spitbol–copyright notice

license–software license for this program

```
m a c r o   s p i t b o l    v e r s i o n  3 . 7
---------------------------------------------------
date of release  -  16 april 2009
permission to use spitbol may be negotiated with
professor robert b. k. dewar.
sites which have obtained such permission may not pass
on copies of the spitbol system or parts of it except
by agreement with dewar.
version 3.7 was maintained by
     mark emmer
     catspaw, inc.
     p.o. box 1123
     salida, colorado 81021
     u.s.a
tel no -  (719) 539 3884
e-mail - marke at snobol4 dot com
versions 2.6 through 3.4 were maintained by
     dr. a. p. mccann
     department of computer studies
     university of leeds
     leeds ls2 9jt
     england.
from 1979 through early 1983 a number of fixes and
enhancements were made by steve duff and robert goldberg.
to assist implementors a revision history based on
version 2.7 is being maintained.
```

spitbol–revision history

```
r e v i s i o n   h i s t o r y
-------------------------------
version 3.6a to 3.7 (november 1, 1991, mark b. emmer)
-----------------------------------------------------
bugs fixed
----------
b3.701  add btkwv and refined test at cdgvl+9 to prevent
        variable names alphabet, lcase, ucase from being
        pre-evaluated because of their associated
        constant keywords.  the code
           alphabet = "abc"; output = size(alphabet)
        returned zero because of pre-evaluation.
b3.702  delay binding to function block of fourth
        argument to trace function.  this permits the
        trace function to be invoked before the 4th
        argument function is defined.  accomplished by
        storing a vrblk pointer in trfnc, and fetching
        its vrfnc entry later, in trxeq.
b3.703  references to keywords with constant pattern
        values (&arb, &bal, etc.) did not work.  a wtb
        instruction had been omitted at acs14+2.
b3.704  if a program employed the code function to
        redefine a label that was the entry location of
        a user-defined function, the function would
        continue to jump to its old function body.  pfcod
        in pfblk was pointing directly to the target code
        block, instead of doing so indirectly through the
        vrblk for the entry label.
b3.705  the test that required a label to be defined
        before it could be used as the entry of a user-
        defined function has been removed.  functions
        may be defined even if the label is yet
        undefined.
b3.706  after a compilation error in the code function,
        the eval function produces spurrious errors.  the
        code offset cwcof was not being reset to the
        beginning of code block.  add line at err04+1 to
        accomplish this reset.
b3.707  inconsistant tests with mxlen corrected.  several
        places were testing with bge instead of bgt,
        resulting in such anomalies as the statement
           &maxlngth = &maxlngth
        failing.  since mxlen is guaranteed to be
        strictly less than dnamb, it is permissible to
        create objects of size mxlen.  bge changed to
        bgt at locations
           s$arr+14, sar07+8, alobf+3, asg14+8, gtar6+10.
b3.708  exit(command string) was not loading ptr to fcb
        chain into wb.  corrected at sext1.
b3.709  change patst to return non-string error for null
        argument.  previously, break(), any(), etc., were
        succeeding, contrary to the language definition.
b3.710  convert function with null second argument
```

```
        crashed system by calling flstg with wa=0.  added
        test at s$cnv, moved error 74 to separate erb at
        scv29.
b3.711  leq(,) crashed system.  lcomp did not obey
        minimal assumption that cmc opcode will always
        be called with wa .gt. 0.  added test at lcmp1.
b3.712  modified line at sdf07+4 to use register wa
        instead of wb.  this corrects problem of define
        function with local variable list that begins
        with comma-  define("f(x),l1,l2")
b3.713  erroneous plc on uninitialised r$cim in listr.
b3.714  erroneous call to flstg possible with null string
        at sdat1.
b3.715  when copy function used with table argument, fix
        problem at cop07.  when copying first teblk on a
        chain, the pseudo-previous block pointer in xr
        is pushed on the stack prior to calling alloc.
        this is not a valid block pointer, as it points
        within the tbblk.  if the subsequent alloc
        invokes gbcol, the heap becomes scrambled.
        recoded to save pointer to start of block, plus
        offset in wb.
b3.716  at iop01, if gtvar triggered garbage collection
        via alost, trap block in wc was not collected.
        save wc on stack to make it collectable across
        gtvar call.
b3.717  at asg10, allow case of variable with more than
        one trblk, as happens with the following stmt -
        output(.output, .output, filename).
b3.718  at senf1, trblk chain search was reloading chain
        head, causing infinite loop if the desired trblk
        was not the first on chain.  system crashed with
        trace(.v1) output(.v2,.v1,file).
b3.719  prototype strings (define, load, data, etc.) were
        allowing blank characters, producing bogus
        variable names.
b3.720  the fact that iofcb destroyed register wc was not
        documented.  b$efc conversion of file argument
        never worked because wc and xt were destroyed by
        call to iofcb.
b3.721  ioput left a trblk attached to filearg1 if sysio
        failed.  subsequent use of this filearg1 variable
        in another i/o call would crash system.
b3.722  add chk at evlp1 to catch recursive pattern error.
b3.723  allow -line to work properly within code function
        by setting cmpln directly in cnc44.  if file name
        absent, decrement scnpt to rescan terminator.
b3.724  when mxlen exceeds start of dynamic memory, round
        it up to multiple of word size prior to storing
        in dnamb at ini06.
b3.725  provide right padding of zero characters to any
        string returned by an external function.
b3.726  reset flptr at bpf17 for undefined function
```

```
        when evalx is evaluating an expression.
b3.727  modify code after read5 for outer nesting of
        an execute-time compile of -include statement.
        create a substring of remainder of original
        code function argument string and return as
        result of readr function
b3.728  the definition of the aov opcode is corrected.
        formerly the definition specified that the branch
        was to be taken if the result of the addition
        exceeded cfp$m, implying a test for overflow
        from signed addition.
        however, address arithmetic must be unsigned to
        allow for systems where the high order address
        bit is set.  therefore, the test must be for
        carry out of the high order bit, if the result
        would exceed cfp$l.
b3.729  a label trace on the entry label for a function
        was undetected, resulting in a transfer to
        b$trt and subsequent crash.  see bpf08 for fix.
b3.730  pop first argument to substr if it is a buffer.
b3.731  pattern replacement with buffer subject returned
        null string instead of new subject value.
        changed to behave as if subject was a string.
b3.732  if convert function was called with a buffer
        first argument and "buffer" second argument,
        it would convert the buffer to a string, and
        then back to a buffer.  this has be corrected
        to simply return the first argument as the
        function result.
b3.733  detect external function returning a null string
        unconverted result at bef12, and jump to exnul.
b3.734  fix problem at ins04 when inserting zero length
        string into buffer.  defend against invoking
        mvc with a zero value in wa, which will cause
        some implementations to wrap the counter.
b3.735  add overflow test for cos and sin to detect
        out-of-range argument.
b3.736  fixed problem introduced with b3.727 not
        restoring r$cim, scnpt and scnil after creating
        substring.
b3.737  fixed tfind to place default value in newly
        allocated teblk.
b3.738  added bl$p0 to p$nth entry point.  the expression
        datatype(convert("","pattern")) would crash when
        the dtype function uses the non-existant type
        word preceding p$nth.
b3.739  bug at gtn35 in the case of overflow during cvm.
        wb can be destroyed by cvm on some platforms.
b3.740  protect scontinue from usage in other than error
        320 case.
b3.741  protect continue from usage following error
        evaluating complex failure goto.
changes
```

```
-------
c3.701  add .culk conditional to include &lcase, &ucase.
c3.702  add -line nn "filename" control card.
c3.703  move .cnld conditional up in routine dffnc to
        omit all tests for b$efc.
c3.704  add conditional .cicc to ignore unrecognized
        control cards.
c3.705  add conditional .cnsc to omit string to numeric
        conversion in sort.  the presence of this
        conversion mode produces a sort result that is
        dependent upon the order of input data.
        for example, given input data "2", 5, "10",
        string comparison yields "10" lt "2", but string
        to integer conversion yields "2" lt 5 lt "10".
c3.706  add seventh return from syshs that allows callee
        to return a string pointer and length.  this is
        done to eliminate the need for the caller to have
        an scblk big enough to accommodate long strings.
c3.707  add eighth return from syshs to force copy of
        block pointed to by xr.
c3.708  made -copy a synonym for -include.
c3.709  add conditional .cbyt for statistics displayed
        in bytes rather than words.
c3.710  dump null valued variables when dump = 3.  core
        dump produced for dump = 4.
c3.711  restrict minimum value to which keyword maxlngth
        can be set to 1,024 via new variable mnlen.
c3.712  add conditional symbol .cmth for extended math
        functions- atan, chop, cos, exp, ln, sin, sqrt,
        tan.  x**y and remdr(x,y) are extended to include
        reals.
c3.713  add bit to syspp to set -print upon entry
c3.714  add conditional .csfn to track source file name
        associated with each code block.
c3.715  add conditional .cinc for -include control card
        feature.  the format of the card is
           -include "filename"
        include control cards may be used during both the
        initial compile and execute-time compile.  the
        filename is saved in a table, and redundant
        includes of that file are ignored.
c3.716  add conditional .csln to include source line
        number in code blocks.  release current ccblk
        after initial compile.
c3.717  changed rilen to 258 (from 120) to provide
        uniform input line length when reading from
        terminal or input.
c3.718  add additional exit to iofcb to distinguish
        argument not convertable to string and argument
        file not open.
c3.719  add fourth and fifth arguments to host function.
c3.720  add &compare keyword to control string
        comparisons.
```

c3.721 setup pfdmp at iniy0 in case osint forced &profile non-zero.
c3.722 add conditional symbol .caex to include up arrow as synonym for exponentiation.
c3.723 add conditional .ccmc and external function syscm to provide string comparison using collation sequence other than strict ordering of character codes (international compares).
c3.724 add conditional .cpol and external function syspl to provide interactive control of spitbol execution.
c3.725 add conditional symbol .cera and external function sysea to provide advice of compilation and runtime errors to osint.
c3.726 add cmpln, rdcln, rdnln to track source line number.
c3.727 converted error messages to upper/lower case.
c3.728 add conditional .cgbc to external routine sysgc. called at the start and end of garbage collection to perform any needed notification to operating system or user.
c3.729 modified last line of s$set from exnul to exint so seek can return final file position after seek.
c3.730 place mov xr,(xs) at s$rmd+4 to allow real second arg to remdr.
c3.731 remove redundant bge xr,=cfp$u,scn07 at scn06+4
c3.732 change definition of cmc and trc such that only xl must be cleared after operation. note, this change was subsequently voided. cmc and trc must clear both xl and xr, because utility routines may preserve xl or xr on the stack, and the stack is collectable by gbcol.
c3.733 remove most branches to exits and exixr. instead, jump directly to next code word.
c3.734 add error 260 for array too large in gtarr.
c3.735 add conditional .cs32 to initialize stlim to 2147483647.
c3.736 add second argument to exit function, allowing user to specify file name of load module being written. if omitted, osint will provide a default name.
c3.737 add conditional .cspr to include spare locations in working area. these may be used in later bug fixes without changing the size of the working storage and obsoleting modules created by exit(). subsuently removed in c3.767.
c3.738 add r$cts to remember last string used to build bit column in patst.
c3.739 change flstg to type e procedure instead of r.
c3.740 standardize on big-endian systems. at the implementors choice, the zgb opcode can also perform a byte swap if necessary to achieve big-

endian byte ordering. this is done so that systems with similar word lengths will produce the same hash code for strings, and hence the same ordering for table entries. the hashs procedure has an additional zgb added to reorder the length word.

c3.741 add conditional .csou to cause assignments to output and terminal variables to be processed through calls to sysou rather than through listing buffer. done to eliminate short record lengths enforced by buffer size. a code of 0 or 1 is passed to sysou instead of an fcblk.

c3.742 increased iniln, inils, rilen to 1024.

c3.743 add bit to syspp to set noerrors mode.

c3.744 add .ccmk conditional to include keyword compare even if syscm is not being included. done to provide identical data regions in systems that implement syscm and those which do not, so that save files can be exchanged in the next release.

c3.745 add wc return parameter to sysil to allow interface to inform spitbol if file about to be read is a binary file. if so, no blank trimming occurs.

c3.746 fold load function argument types to upper case.

c3.747 add .cexp conditional to have sysex pop its arguments.

c3.748 in stopr, do not attempt to display file name and line number if stopping because of stack overflow during garbage collection. pointers to file name table and code block are wrong.

c3.749 add bit to syspp to set case folding mode.

c3.750 add additional return from sysld if insufficient memory to load/call external function.

c3.751 add additional returns from sysex if insufficient memory or bad argument type.

c3.752 ignore leading and trailing blanks in arguments within prototype strings to clear, data, define and load.

c3.753 test for fatal error at err04 and abort if so. force termination on stack overflow by setting errft to 4 in stack overflow section.

c3.754 recode copy loop at srt14 to exchange usage of registers xl and xr. this permits use of the mvw order instead of the explicit loop coding previously employed.

c3.755 add .ceng conditional to include routines needed by text processing engine. add routines enevs and engts for use by engine or debugger. copy xr to xl around call to syspl to allow syspl to trigger garbage collection.

c3.756 add &file, &lastfile, &line, &lastline keywords. for now, line and lastline are maintained in the same manner as stno and lastno, which adds overhead to the statement initialization code. a possible change is to create a stmln procedure that maps statement numbers to line numbers. one simple strategy would be to sweep code blocks in memory looking for the statement number and extracting the line number from that code block. such a procedure would also allow line numbers (and file names) to be added to statement profile reports.

c3.757 change sort to fail instead of producing error message if argument table is null. change sorta to return failure. add another return to gtarr to distinguish null table from bad argument.

c3.758 create procedure prtmm to display memory usage statistics, and call it when producing end-of-run stats.

c3.759 add label scontinue to allow setexit to resume execution exactly where it was interrupted.

c3.760 add snobol4 backspace function and conditional .cbsp.

c3.761 add additional arguments to sysgc to assist virtual memory managers.

c3.762 the method of converting a table to an array has been revised. previously, table elements were copied to the result array in the order they were encountered along the various hash chains. this appeared to the user as a random ordering. however, spitbol/370 as well as sil snobol4 ordered array elements according to their time of entry into the table. user programs that relied upon this behavior malfunctioned when ported to macro spitbol.
to remedy this, the conversion is performed in three steps:
1. convert table to an array placing the address of each teblk in the array instead of the key and value.
2. sort the array of addresses. this orders elements by time of creation (ascending address).
3. scan the array, replacing addresses with the key and value from the referenced teblk.

the affected portions of the program are at s$cnv and in gtarr, which now accepts an additional argument specifying whether to place key/values in the array or teblk addresses.

c3.763 if case-folding is active, fold the function name provided to the load() function before passing it to sysld.

c3.764 add sediment algorithm to garbage collector, conditioned on .csed.

c3.765 add optimization to discard null statements and statements which just have a constant subject (see code at cmp12).

```
c3.766  rearranged order of initial objects in static
        memory so that hash table is the last of the four
        object created by initialization code.  this is
        done so that the print buffer, gts work area, and
        &alphabet keywords do not need to be saved in
        any save file created by osint.  added routine to
        initialize these structures.
c3.767  removed .cspr conditional and spare locations.
c3.768  added .crel conditional and extensive routines
        (reloc et. al.) to perform relocation of data
        in working section, static region, and dynamic
        region after reload of a saved memory image.
        routines relaj, relcr, and reloc are invoked
        by osint after reloading a save file.
        it is now possible to reload such an image even
        if the spitbol compiler and its data structures
        are reloaded to other addresses.  the working
        section has been extensively rearranged to
        accommodate the reloc procedure.
c3.769  zero r$ccb (interim ccblk ptr) in collect,
        convert, eval, and exit functions to release
        unneeded ccblk memory.
c3.770  add exit(4) and exit(-4) to allow execution to
        continue after writing save file or load module.
        revised sysxi interface to detect continuation
        after performance of exit(4) or exit(-4) action.
c3.771  change filnm to preserve registers.
c3.772  addition of .cncr and syscr (real to string
        system routine option).
c3.773  modified replace function to optimize usage
        when second argument is &alphabet.  in this case,
        the third argument can be used as the translate
        table directly.
c3.774  modified conditionals for buffers and reals so
        that their respective block codes are always
        present, even if these data types are conditioned
        out.  this provides consistent block code
        numbering for external functions.
c3.775  modified alobf to test string length against
        kvmxl instead of mxlen.  also, alobf was testing
        total size of bfblk, instead of just string len.
c3.776  move utility routines source up to lie between
        predefined snobol functions (s$xxx) routines and
        utility procedures.  this was done to assist
        translation on platforms such as apple macintosh
        that use 15-bit offsets to store error exits (ppm
        branches).  offsets to labels like exfal were
        just too far away.  similarly, functions tfind,
        tmake, and vmake are located out of alphabetic
        order to satisfy the macintosh's limited range
        for subroutine calls.  move built-in labels
        beyond the block and pattern routines to get it
        within 32k of the error routines.
```

```
c3.777  at scn46, allow colon, right paren and right
        bracket to terminate = operator with default
        null operand.
c3.778  added .ctet conditional for table entry trace.
c3.779  introduce cfp$l, the largest unsigned value
        that may be stored in a one-word integer.  this
        is done to accommodate machines where memory
        addresses have the high-order address bit set.
c3.780  perform replace in place if first arg is buffer.
c3.781  perform reverse in place if first arg is buffer.
c3.782  change sysou to accept buffer as well as string
        to be output.  change code at asg11 to prevent
        conversion of buffer to string.
c3.783  optimize pos and rpos when it is the first node
        of a pattern and has either an integer or simple
        expression variable argument.  if unanchored mode
        and the cursor is zero, it is advanced directly
        to the desired cursor position.
c3.784  perform trim function in place if arg is buffer.
c3.785  add gtstb procedure to get a string or buffer
        argument for replace, reverse, size, trim, etc.
c3.786  change leq, lgt, etc. to perform comparisons
        without converting buffer arguments to strings.
        this is done by changing lcomp to accept buffer
        argument(s).  this also affects sort function,
        which will compare two buffers as strings.
c3.787  change gtnum to use characters in buffer without
        conversion to a string.  this implies that acomp
        will perform arithmetic comparisons of buffers
        without converting to strings first.
c3.788  perform comparisons of strings and buffers in
        sortc.
c3.789  change insbf to allow insertion of a buffer into
        a buffer without first converting it to a string.
        note that this only works when the two buffers
        are not the same.
c3.790  documentation change:  note that all of the block
        move opcodes should have wa .gt. 0.  not all
        implementations avoid moving objects when wa is
        zero.
c3.791  change ident to provide buffer/buffer and
        buffer/string comparisons, to accommodate users
        who perform ident(buf) to check for null string
        in buffer.
c3.792  added fullscan keyword initialized to one.  user
        may set to any non-zero value, will receive an
        error message if attempts to set to zero, since
        quickscan mode is not supported.
c3.793  rewrote statement startup code at stmgo to only
        perform checking of profiling, stcount tracing,
        and statement counting if necessary.
c3.794  add additional exit to sysfc and ioput to signal
        that i/o channel (fcblk) is already in use.
```

```
        added error message numbers 289 and 290.
c3.795  added optional integer argument to date function
        to specify format of date string returned by
        sysdt.
version 3.6 to 3.6a (oct 83)
---------------------------
changes
-------
c3.617  add .cnlf. if defined, then arguments to external
        functions may be declared to have type file.
        such arguments must have been used as second
        arg to input() or output() and a pointer to the
        fcb is passed to the external function.
version 3.5 to 3.6 (jun 83)
---------------------------
codes used to identify authors are (sgd) for duff,
(reg) for  goldberg, and (lds) for shields.
bugs fixed
----------
b3.601  (sgd) to fix multiple trap block problem in asign
b3.602  (sgd) patch in gtarr to fix null convert.
b3.603  (sgd) inserted missing wtb after sysmm calls.
b3.604  (sgd) use string length in hashs.
b3.605  (sgd) fixed serious parser problem
        relating to (x y) on line being viewed as pattern
        match.  fixed by addition of new cmtyp value
        c$cnp (concatenation - not pattern match).
b3.606  (sgd) fixed exit(n) respecification code
        to properly observe header semantics on return.
b3.607  (sgd) bypass prtpg call at initialization
        following compilation if no output generated.
        this prevents output files consisting of the
        headers and a few blank lines when there is no
        source listing and no compilation stats.
        also fix timsx initialization in same code.
b3.608  (sgd) b$efc code did not check for
        unconverted result returning null string.
b3.609  (sgd) load pfvbl field in retrn for
        return tracing. this was causing bug on return
        traces that tried to access the variable name.
b3.610  (sgd) fixed problem relating to compilation of
        goto fields containing small integers
        (in const sec).
b3.611  (reg) prevent clear() from clobbering protected
        variables at label sclr5.
b3.612  (reg) fixed gtexp from accepting trailing
        semicolon or colon. this is not a legal way
        to end an expression.
b3.613  (reg) fixed difficulties with listings during
        execution when no listing generated during
        compilation. -list to code() caused bomb.
        fix is to reset r$ttl and r$stl to nulls not 0
        after compilation.
```

```
        (listr and listt expect nulls)
        when listing and statistics routed to different
        file than execution output, error message is sent
        to execution output (and gets separated from
        ... in statement ... msg). labo1 calls sysax and
        stopr does not call sysax if entered from labo1.
b3.614  (lds) fix misuse of wc just after asg10.
b3.615  (lds) add comment pointing out suspicious code
        after tfn02
b3.616  (lds) fix inconsistent declaration of sorth.
b3.617  (lds) insert missing conditional tests on cnbf.
b3.618  (lds) fix some violations of minimal language
        that had slipped past some translators.
b3.619  (lds) correct error introduced in fixing b3.614.
```

```
changes
-------
c3.601  (sgd) addition of .cnci and sysci (int to string
        system routine option)
c3.602  (reg) changed iniln and and inils to 258
c3.603  (sgd) merged in profiler patches, repaired code.
c3.604  (sgd) added buffer type and symbol cnbf
c3.605  (sgd) added char function.  char(n) returns nth
        character of host machine character set.
c3.606  (reg) added cfp$u to ease translation on smaller
        systems - conditional .cucf
c3.607  (reg) added lower case support, conditional .culc
c3.608  (reg) added set i/o function, conditional .cust
c3.609  (reg) conditionalized page eject after call to
        sysbx and added another before call to sysbx,
        so that, if desired by the implementor,
        standard output will reflect assignments made
        by executing program only.
        conditional .cuej controls - if defined then
        eject is before call to sysbx.
c3.610  (lds) introduce .ctmd to support systm that
        reports elapsed time in deciseconds instead of
        milliseconds.
c3.611  (lds)  provide place for .def or .und for each
        conditional  option, so that settings can be
        changed without changing line numbers.
        current settings are for 808x translation.
c3.612  (lds) obey (new) restriction that operand in
        conditional branch instruction cannot have form
        (x)+ in order to simplify translations for which
        postincrement not readily available.
c3.613  (reg,lds) add op
              flc wreg
        that folds character in wreg to upper case.
        this op is used only if .culc is defined.
        this change also involves addition of keyword
        &case which when nonzero (the initial setting)
        causes the case folding just described to be
        done.
c3.614  (lds) add option .cs16 to permit initialization
        of statement limit values to 32767 for 16 bit
        machines.
c3.615  (lds) permit return point and entry point
        addresses to be distinguished by their parity
        instead of by lying within a certain range
        of values.  introduce conditional symbols
        .crpp  return points have odd parity
        .cepp  entry points have odd parity
c3.616  (lds) introduce new minimal opcodes to branch
        according to parity,
          bev  opn,plbl  branch if address even
          bod  opn,plbl  branch if address odd
        an address is even if it is a multiple of cfp$b.
```

```
documentation revisions
-----------------------
d3.601  (lds) bring minimal machine description up to
        date
```

```
version 3.4 to 3.5 (feb 79)
---------------------------
bugs fixed
----------
b3.401  prtst should be declared as an r type procedure.
b3.402  timing error if spitbol fails in dump.
b3.403  error in handling omitted args of operators.
b3.404  too many lines put on first page of listing.
b3.405  leading unary operator in eval erroneously needed
        preceding blank.
b3.406  identifying name in dump of array or table values
        was omitted.
b3.407  eval unable to return a deferred expression.
b3.408  illegal if setexit code branches to return.
b3.409  illegal on detaching input, output, terminal.
changes
-------
c3.401  -sequ and -nose control cards removed.
c3.402  option provided to suppress system identification
        on listing.
c3.403  description of sysbx slightly revised.
c3.404  permissible to modify scblk length before taking
        error returns from sysin, sysrd, sysri.
c3.405  conditional .cnld may be defined to omit load().
c3.406  conditional .cnex may be defined to omit exit().
c3.407  table now accepts a third argument specifying
        default initial lookup value.
c3.408  routines sort, rsort for sorting arrays and table
        introduced. specification is as in sitbol.
        routines may be omitted by defining .cnsr .
c3.409  error in code(), eval() call now causes statement
        failure but errtext keyword is still set.
c3.410  arg to code() may contain embedded control cards
        and comment delimited by a semicolon.
documentation revisions
-----------------------
d3.401  purpose of restriction 2 in minimal section -6-
        (operations on char values), erroneously stated
        to be for cmc, rather than for ceq, cne.
        descriptions of above opcodes revised.
d3.402  description of ent clarified.
d3.403  descriptions of several opcodes revised to remove
        technically invalid literals e.g. =0 , *1.
d3.405  restricted use of letter z in minimal clarified.
d3.406  divide by zero explicitly mentioned in relation
        to overflow setting.
```

```
version 3.3 to 3.4 (oct 78)
---------------------------
bugs fixed
----------
b3.301  illegal for erroneous eval() arg.
b3.302  address arithmetic overflow in alloc and alocs.
b3.303  -eject and -space ignored -nolist option.
b3.304  erroneous argument scan in load().
b3.305  erroneous plc on uninitialised r$cim in nexts.
b3.306  ldi used instead of mti after prv07.
b3.307  misuse of rmi at erra2.
b3.308  misuse of mti in hashs.
b3.309  bug in -sequ card sequence number checking.
b3.310  stack overflow error message not always printed.
b3.311  corrupt prototype print for traced arrays.
b3.312  pattern first arg in dupl caused error.
b3.313  omitted csc in s$rpd, erroneous csc in convert.
b3.314  misplaced btw in exbld.
b3.315  incorrect code in hashs.
b3.316  failure of load to scan integer arg.
b3.317  table access with negative integer arg. failed.
b3.318  error in returning result of loaded function.
b3.319  =e$srs used after ini01 instead of *e$srs.
b3.320  err used instead of erb after systu
b3.321  label could start with disallowed character.
b3.322  continue after setexit had bad heuristic.
```

```
changes
-------
c3.301  sysax and .csax introduced - see sysax
        in procedures section.
c3.302  variable mxlen introduced. contains the maximum
        size of a spitbol object and is not changeable
        after initialisation. may be defaulted or set
        explicitly by sysmx.
c3.303  syshs returns revised - see syshs.
c3.304  new minimal opcode aov to fix b3.302.
c3.305  inhibit stlimit check if stlimit made negative.
c3.306  cfp$m is required to be of form 2**n - 1.
c3.307  dupl made to conform to sil snobol4 standard.
c3.308  lch and sch actions more closely defined.
c3.309  batch initialisation code omitted if conditional
        assembly symbol .cnbt (no batch) defined.
c3.310  (wa) contains argument count in sysex call.
c3.311  sysfc  may request allocation of static fcblk.
c3.312  if ia,wc overlap, restriction put on dumping/
        restoring these registers.
c3.313  new listing option intermediate between compact
        and extended provided (see syspp).
c3.314  revision of sysxi interface to permit options for
        load module standard o/p file (see sysxi,syspp).
c3.315  last arg of substr may be omitted - treated
        as remainder of string.
```

```
version 3.2 to 3.3 (jan 78)
---------------------------
bugs fixed
----------
b3.201  array reference and external function load
        routines illegally accessed information
        beyond the stack front.
        similar fault in unanchored pattern matching.
b3.202  dump(1) produced dump(2) type output.
b3.203  wtb conversion omitted in code following
        ini01, ini02, exbld.
b3.204  incorrect fail return from tfind in arref.
b3.205  endfile did not detach i/o associated variables.
b3.206  -space with omitted arg. failed
b3.207  looped if dump keyword non-zero after stack
        overflow in garbage collect failure.
b3.208  failure in reading numbers with trailing blanks.
changes
-------
the extensive changes made here mostly result from a
snobol4 implementors meeting held at new york university
in august 1977. they are aimed at
     (1) having spitbol conform to certain snobol4
     language standards  and
     (2) producing a stable definition of minimal by
     carrying out a few essential revisions in the light
     of experience in its use.
changes to spitbol
------------------
c3.201  default values for keywords trim and anchor are
        zero. on systems where records are customarily
        handled without traling blanks, there is no
        obligation to supply such blanks.
c3.202  default value of -inxx control card is -in72.
```

```
c3.203  the second argument of input and output is
        permitted to be an integer as in snobol4.
        in addition input(), output() now give a snobol4
        statement failure if sysio uses the file not
        found return.
        the third argument has a recommended format and
        to override its default delimiter (,) a
        conditional assembly symbol, .ciod, is used.
        interfaces to sysef,sysej,syfc,sysio,sysrw
        are revised.
        wc may now be used to return from sysio, a max
        record length.
c3.204  a new configuration parameter cfp$f (scblk offset
        is introduced. cfp$u is removed.
c3.205  implementation and version identification is
        required - see sysid.
c3.206  routine sysmx returns the maximum length of
        spitbol objects (strings arrays etc).  this
        information is not now needed at time of entry to
        spitbol and hence wc should be zero on entry.
c3.207  a conditional parameter .cnra permits assembly
        of a more compact version with no real
        arithmetic code.
c3.208  terminal is a new pre-associated variable
        capable of performing input and output to an
        online terminal.
        sysri is a new routine used in the implementation
        of this. see also syspp.
c3.209  the environment parameters e$--- are now
        provided by the minimal translator using the
        revised   equ  *   format (see c3.229 and start
        of spitbol definitions section - some reordering
        of symbols has occurred).
c3.210  the interface of sysxi has been slightly revised.
        unavailability of i/o channels after exit(1),
        exit(-1) is documented together with additional
        error return usage for sysin,sysou,syspr,sysrd.
c3.211  spitbol error codes have been frozen - see c3.230
c3.212  the utility routines arref etc. are now
        introduced by rtn statements.
c3.213  sysrl (record length for std input file) is
        removed. since implementation of a general -inxxx
        control card and an ability to specify max record
        length using the third argument of input, sysrl
        has become redundant.
c3.214  sysej and sysxi are now passed a chain linking
        all fcblks in use.
c3.215  a special ending code in sysej is used when
        attempts to use standard output channel fail.
c3.216  restriction c3.233 observed so simplifying
        optimised translation of ent with omitted val.
```

```
changes to minimal
------------------
c3.220  minimal opcodes dec, dim, inc, and bmp
        are withdrawn and replaced by the more consistent
        set dca, dcv, ica, icv.
c3.221  chs has been replaced by the more generally
        useful zgb (still likely to be a no-op for most
        implementations however).
c3.222  the set of character comparisons has been
        reduced to ceq and cne to ease implementation
        problems.
c3.223  opcode irz is removed and dvi, rmi orders are
        redefined to conform to more common usage.
c3.224  new opcodes ssl and sss are defined. their use
        permits return links for n type procedures to be
        placed on a local stack if desired.
c3.225  opcode mnz complements zer. it moves a non-zero
        flag to its destination.
c3.226  for some machines it is preferable for the stack
        to build up rather than down. to permit this
        without need for massive changes in minimal and
        recoding of existing programs, a scheme has been
        devised in which an additional register name, xt,
        is used as a synonym for xl when this register
        is involved in stack manipulation- see section 4.
c3.227  section 0 of a minimal program is renamed the
        procedure section. it now contains, in addition
        to exp, specifications of internal procedures
        and routines by means of the inp and inr opcodes.
c3.228  the literal operand formats =int and *int have
        been withdrawn. =dlbl and *dlbl must be used in
        their stead.
c3.229  the format
        label  equ  *nn
        used to specify values supplied by the minimal
        translator for char. codes etc. is replaced by
        label  equ  *
        where the order in which the definitions are
        supplied by the translator should match the
        order of occurrence in the definitions section.
c3.230  the format of err,erb opcodes is changed to
        require a numeric operand.
c3.231  the rtn opcode is used to introduce routines
        (which are quite distinct from procedures).
c3.232  conditional assembly directives may be nested.
c3.233  minor restriction placed on the omission of
        val with the ent opcode.
```

```
version 3.1 to 3.2 (aug 77)
---------------------------
bugs fixed
----------
b3.101  astonishing this was unnoticed for three years.
        bad code for snobol4 integer divide, /, gave
        wrong result for operands of opposite signs.
        implementations have either wrongly translated
        dvi and got correct result or correctly
        translated dvi and got wrong result - leeds had
        one of each. see also c3.106.
        test program no. 1 now extended to check /
        more thoroughly.
b3.102  garbage collection bug in scane
changes
-------
c3.101  option to use additional characters ch$ht,ch$vt
        (horizontal and vertical tab) with same syntactic
        significance as ch$bl (blank).
c3.102  option to use a set of shifted case alphabetic
        characters ch$$a ... ch$$$.
c3.103  conditional assembly features are introduced into
        minimal on account of the above.
        see minimal documentation section for details
        of above changes.
c3.104  lch and sch may use an x register first
        operand as alternative to a w register.
c3.105  spitbol statement numbers in the listing may
        optionally be padded to 6 or 8 chars instead of 5
        by defining conditional assembly symbols
        .csn6 or .csn8 .
c3.106  to fix bug 3.101. at moderate cost,
        opcode irz (branch if integer divide remainder
        zero) introduced.
c3.107  to handle possible machine dependency in string
        hashing, chs (complete hashing of string) opcode
        is introduced. probably a no-op on most machines
        - not on the dec10.
c3.108  procedures patin,tfind,trace have been
        modified to conform to the minimal standard
        call and return regime.
c3.109  sysfc interface revised slightly to permit
        osint to return a pointer to a privately
        allocated fcblk which spitbol will return on
        subsequent i/o - see sysfc doc.
c3.110  to remove inconsistencies in calling sequences,
        all sys routines having access to a possible
        fcblk have fcblk ptr or zero in reg. wa on entry.
        change affects sysef, sysen, sysil, sysin,
        sysou, sysrw.
c3.111  syspp bit allocated to provide
         -noexec option on entry to spitbol.
```

```
documentation revisions
-----------------------
d3.101  need to preserve registers in syspi, syspr,
        sysrd calls was overstated.
```

```
version 3.0 to 3.1 (mar 77)
---------------------------
bugs fixed
----------
b3.001  replace() could fail during pre-evaluation.
        spitbol now signals an error for null or
        unequally long 2nd and 3rd arguments.
b3.002  negative second arguments to dupl, lpad, rpad
        caused spitbol to signal an error. now causes
        return of null string or first arg respectively.
b3.003  brn-s used instead of ppm-s in s$sub.
b3.004  err used instead of erb after cmp30.
b3.005  b$pfc, s$cnv, s$def, arith and arref kept
        information illegally above the stack top.
b3.006  pre-evaluation of constant parts of
        complex gotos was erroneous.
b3.007  incorrect handling of labels compiled by code().
b3.008  the single use of trc (in s$rpl) was not in
        accord with its definition. some translations of
        trc may need revision now that the use
        has been brought into line with definition.
changes
-------
a debate on a few weaknesses in minimal design has
been resolved by introducing 4 new opcodes.
c3.001  new minimal opcodes bmp and dim introduced
        to augment inc and dec which are applicable
        only to addresses.
c3.002  the opcode szc (store zero characters) had
        a restricted applicability. it has been
        replaced by the more general zer (zeroise).
c3.003  fcblks may be optionally allocated as xrblk-s or
        xnblk-s  - see sysfc for vital information.
c3.004  control card processing has been recoded.
        -inxxx allows specification of standard input
        file record lengths other than 72 or 80, see also
        sysrl. -sequ is ignored unless -in80 is in effect
c3.005  to enable efficient buffering of chars on
        machines without char. handling orders, the
        csc (complete store characters) instruction
        is introduced. current implementations can
        translate it as a no-op if it is of no benefit.
c3.006  integers 0,1,2 are treated specially.
        icblks in static are used instead of
        allocating space in dynamic.
```

```
version 2.7 (june 76) to 3.0 (jan 77)
-------------------------------------
bugs fixed
----------
b2.701  goes illegal if timed out during processing of
        dump() call.
b2.702  goes illegal if spitbol error detected in args of
        code() or eval(). bug fixed so that user now gets
        a spitbol error report (trappable by setexit)
        before statement failure.
b2.703  goes illegal in some circumstances when
        multiple compilation errors occur in a statement
b2.704  goes illegal if garbage collector runs out of
        stack space.
b2.705  control card processing incorrect for cdc 6400.
b2.706  incorrect handling of multiple occurrences of
        chars in replace 2nd and 3rd args.
b2.707  stack overflow in pre-evaluation of replace in
        cdc 6400 version.
b2.708  an explicit call of sysmw was coded in s$dat
        instead of the mvw opcode.
b2.709  call of garbage collector whilst dumping
        caused havoc.
b2.710  size restriction on spitbol objects (size must be
        numerically less than lowest dynamic address)
        was not enforced, with potential for catastrophe.
b2.711  deferred expressions involving alternation or
        negation were incorrectly translated.
b2.712  listing of a compilation error at the end of a
        long line could cause compiler to go illegal.
b2.713  incorrect -nofail code with success goto.
```

```
changes
-------
(it is not anticipated that major revisions on this
scale will be frequent).
c2.701  default value of anchor keyword is set to 1. this
        conflicts with snobol4 practice but is a
        preferable default for most applications.
c2.702  if errtype is out of range the string in keyword
        errtext is printed as the error message.
c2.703  if stlimit is exceeded, up to 10 more statements
        may be obeyed to permit setexit trap to gain
        control.
c2.704  the concept of an interactive channel is
        introduced for implementations where an online
        terminal may be used for spitbol. the standard
        print file may be specified as interactive in
        which case shorter title lines are output.
        alternatively copies of compilation and
        execution errors only may be sent to this channel
c2.705  printing of compilation statistics may be
        suppressed.
c2.706  printing of execution statistics may be
        suppressed.
c2.707  extended or compact listing format may be
        selected.
c2.708  an initial -nolist option may be specified
        before compilation starts.
c2.709  to specify choices implied by c2.704 to c2.708
        syspp interface is revised and syspi is defined.
c2.710  compilation and execution time statistics
        messages have been shortened.
c2.711  the exit function as in sitbol is introduced
        to permit saving load modules - see sysxi, s$ext.
c2.712  diagnostic routines sysgb and sysgd have been
        removed. they were useful in the early debugging
        days but have fallen into disuse now.
c2.713  szc may have an operand of type opn instead of
        type opw
c2.714  input/output association interface has been
        revised. sysif,sysof have been consolidated into
        the new system routine, sysio, and the
        specification of sysfc has been slightly changed.
c2.715  configuration parameter mxlen has been withdrawn
        and the maximum size of a spitbol object which
        was formerly fixed at spitbol compile time by
        reference to it may now be specified as a run
        time option by placing a value in wc before entry
        to spitbol. (see comment on dynamic area in
        basic information section).
c2.716  a function, host, is introduced which yields
        information about the host machine - see syshs
        and s$hst.
```

```
documentation revisions
-----------------------
d2.701  the description of mvc has been revised to
        reflect the fact that some spitbol code sequences
        rely on mvc not destroying wb. minor changes
        have been made to mwb and mvw descriptions to
        emphasise similarities in the implicit loops of
        these orders.
d2.702  descriptions of dvi and rmi have been clarified.
d2.703  implementation of rsx,lsx,ceq,cge,cgt,chi,clo,clt
        is optional at present since they are currently
        unused. their use in later versions is not
        excluded.
d2.704  impossibility of using stack for return links of
        n type procedures is emphasised.
d2.705  notation (xl),(wc) etc in language description is
        clarified.
d2.706  documentation of sysfc, sysio has been improved.
d2.707  opcode descriptions are cross referenced from
        the alphabetical opcode list.
d2.708  general description of compiler has been moved to
        the start of the compiler proper.
d2.709  definitions of environment parameters have been
        put near the front of the definitions section.
```

minimal–machine independent macro assembly lang.

```
the following sections describe the implementation
language originally developed for spitbol but now more
widely used. minimal is an assembly language
for an idealized machine. the following describes the
basic characteristics of this machine.
section 1 - configuration parameters
there are several parameters which may vary with the
target machine. the macro-program is independent of the
actual definitions of these parameters.
the definitions of these parameters are supplied by
the translation program to match the target machine.
cfp$a                 number of distinct characters in
                      internal alphabet in the range
                      64 le cfp$a le mxlen.
cfp$b                 number of bytes in a word where a
                      byte is the amount of storage
                      addressed by the least significant
                      address bit.
cfp$c                 number of characters which can
                      be stored in a single word.
cfp$f                 byte offset from start of a string
                      block to the first character.
                      depends both on target machine and
                      string data structure. see plc, psc
cfp$i                 number of words in a signed
                      integer constant
cfp$l                 the largest unsigned integer
                      of form 2**n - 1 which can be
                      stored in a single word.  n will
                      often be cfp$n but need not be.
cfp$m                 the largest positive signed integer
                      of form 2**n - 1 which can be
                      stored in a single word.  n will
                      often be cfp$n-1 but need not be.
cfp$n                 number of bits which can be stored
                      in a one word bit string.
cfp$r                 number of words in a real constant
cfp$s                 number of significant digits to
                      be output in conversion of a real
                      quantity.
```

---

if **.cncr**

else

```
                      the integer consisting of
                      this number of 9s must not be too
                      large to fit in the integer accum.
```

fi

---

if **.cucf**

```
cfp$u                 realistic upper bound on alphabet.
```

fi

```
cfp$x                 number of digits in real exponent
```

section 2 - memory

memory is organized into words which each contain cfp$b bytes. for word machines cfp$b, which is a configuration parameter, may be one in which case words and bytes are identical. to each word corresponds an address which is a non-negative quantity which is a multiple of cfp$b.

data is organized into words as follows.

1) a signed integer value occupies cfp$i consecutive words (cfp$i is a configuration parameter). the range may include more negative numbers than positive (e.g. the twos complement representation).
2) a signed real value occupies cfp$r consecutive words. (cfp$r is a configuration parameter).
3) cfp$c characters may be stored in a single word (cfp$c is a configuration parameter).
4) a bit string containing cfp$n bits can be stored in a single word (cfp$n is a configuration parameter).
5) a word can contain a unsigned integer value in the range (0 le n le cfp$l). these integer values may represent addresses of other words and some of the instructions use this fact to provide indexing and indirection facilities.
6) program instructions occupy words in an undefined manner. depending on the actual implementation, instructions may occupy several words, or part of a word, or even be split over word boundaries.

the following regions of memory are available to the program. each region consists of a series of words with consecutive addresses.

1)	constant section	assembled constants
2)	working storage section	assembled work areas
3)	program section	assembled instructions
4)	stack area	allocated stack area
5)	data area	allocated data area

section 3 - registers
there are three index registers called xr,xl,xs. in
addition xl may sometimes be referred to by the alias
of xt - see section 4. any of the above registers
may hold a positive unsigned integer in the range
(0 le n le cfp$l). when the index register is used for
indexing purposes, this must be an appropriate address.
xs is special in that it is used to point to the top
item of a stack in memory. the stack may build up or
down in memory.since it is required that xs points to the
stack top but access to items below the top is permitted,
registers xs and xt may be used with suitable offsets
to index stacked items. only xs and xt may be used for
this purpose since the direction of the offset is
target machine dependent. xt is a synonym for xl
which therefore cannot be used in code sequences
referencing xt.
the stack is used for s-r linkage and temporary
data storage for which the stack arrangement is suitable.
xr,xl can also contain a character pointer in conjunction
with the character instructions (see description of plc).

```
there are three work registers called wa,wb,wc which
can contain any data item which can be stored in a
single memory word. in fact, the work registers are just
like memory locations except that they have no addresses
and are referenced in a special way by the instructions.
note that registers wa,wb have special uses in connection
with the cvd, cvm, mvc, mvw, mwb, cmc, trc instructions.
register wc may overlap the integer accumulator (ia) in
some implementations. thus any operation changing the
value in wc leaves (ia) undefined and vice versa
except as noted in the following restriction on simple
dump/restore operations.
    restriction
    -----------
if ia and wc overlap then
    sti  iasav
    ldi  iasav
does not change wc, and
    mov  wc,wcsav
    mov  wcsav,wc
does not change ia.
there is an integer accumulator (ia) which is capable of
holding a signed integer value (cfp$i words long).
register wc may overlap the integer accumulator (ia) in
some implementations. thus any operation changing the
value in wc leaves (ia) undefined and vice versa
except as noted in the above restriction on simple
dump/restore operations.
there is a single real accumulator (ra) which can hold
any real value and is completely separate from any of
the other registers or program accessible locations.
the code pointer register (cp) is a special index
register for use in implementations of interpretors.
it is used to contain a pseudo-code pointer and can
only be affected by icp, lcp, scp and lcw instructions.
```

section 4 - the stack
the following notes are to guide both implementors of systems written in minimal and minimal programmers in dealing with stack manipulation. implementation of a downwards building stack is easiest and in general is to be preferred, in which case it is merely necessary to consider xt as an alternative name for xl.
the minimal virtual machine includes a stack and has operand formats -(xs) and (xs)+ for pushing and popping items with an implication that the stack builds down in memory (a d-stack). however on some target machines it is better for the stack to build up (a u-stack).
a stack addressed only by push and pop operations can build in either direction with no complication but such a pure scheme of stack access proves restrictive. hence it is permitted to access buried items using an integer offset past the index register pointing to the stack top. on target machines this offset will be positive/negative for d-stacks/u-stacks and this must be allowed for in the translation.
a further restriction is that at no time may an item be placed above the stack top. for some operations this makes it convenient to advance the stack pointer and then address items below it using a second index register. the problem of signed offsets past such a register then arises. to distinguish stack offsets, which in some implementations may be negative, from non-stack offsets which are invariably positive, xt, an alias or synonym for xl is used. for a u-stack implementation, the minimal translator should negate the sign of offsets applied to both (xs) and (xt).
programmers should note that since xt is not a separate register, xl should not be used in code where xt is referenced. other modifications needed in u-stack translations are in the add, sub, ica, dca opcodes applied to xs, xt. for example

```
minimal           d-stack trans.  u-stack trans.
mov  wa,-(xs)     sbi  xs,1       adi  xs,1
                  sto  wa,(xs)    sto  wa,(xs)
mov  (xt)+,wc     lod  wc,(xl)    lod  wc,(xl)
                  adi  xl,1       sbi  xl,1
add  =seven,xs    adi  xs,7       sbi  xs,7
mov  2(xt),wa     lod  wa,2(xl)   lod  wa,-2(xl)
ica  xs           adi  xs,1       sbi  xs,1
```

note that forms such as

```
mov  -(xs),wa
add  wa,(xs)+
```

are illegal, since they assume information storage above the stack top.

section 5 - internal character set
the internal character set is represented by a set of contiguous codes from 0 to cfp$a-1. the codes for the digits 0-9 must be contiguous and in sequence. other than this, there are no restraints.
the following symbols are automatically defined to have the value of the corresponding internal character code.

```
ch$la               letter a
ch$lb               letter b
.                   .
ch$l$               letter z
ch$d0               digit 0
.                   .
ch$d9               digit 9
ch$am               ampersand
ch$as               asterisk
ch$at               at
ch$bb               left bracket
ch$bl               blank
ch$br               vertical bar
ch$cl               colon
ch$cm               comma
ch$dl               dollar sign
ch$dt               dot (period)
ch$dq               double quote
ch$eq               equal sign
ch$ex               exclamation mark
ch$mn               minus
ch$nm               number sign
ch$nt               not
ch$pc               percent
ch$pl               plus
ch$pp               left paren
ch$rb               right bracket
ch$rp               right paren
ch$qu               question mark
ch$sl               slash
ch$sm               semi-colon
ch$sq               single quote
ch$un               underline
```

the following optional symbols are incorporated by defining the conditional assembly symbol named.
26 shifted letters incorporated by defining .casl

```
ch$$a               shifted a
ch$$b               shifted b
.                   .
ch$$$               shifted z
ch$ht               horizontal tab - define .caht
ch$vt               vertical tab   - define .cavt
ch$ey               up arrow       - define .caex
```

section 6 - conditional assembly features

some features of the interpreter are applicable to only certain target machines. they may be incorporated or omitted by use of conditional assembly. the full form of a condition is -

```
.if    conditional assembly symbol    (cas)
.then
       minimal statements1   (ms1)
.else
       minimal statements2   (ms2)
.fi
```

the following rules apply

1. the directives .if, .then, .else, .fi must start in column 1.
2. the conditional assembly symbol must start with a dot in column 8 followed by 4 letters or digits e.g.
   .ca$1
3. .then is redundant and may be omitted if wished.
4. ms1, ms2 are arbitrary sequences of minimal statements either of which may be null or may contain further conditions.
5. if ms2 is omitted, .else may also be omitted.
6. .fi is required.
7. conditions may be nested to a depth determined by the translator (not less than 20, say).

selection of the alternatives ms1, ms2 is by means of the define and undefine directives of form -

```
.def   cas
.undef cas
```

which obey rules 1. and 2. above and may occur at any point in a minimal program, including within a condition.
multiply defining a symbol is an error.
undefining a symbol which is not defined is not an error.
the effect is that if a symbol is currently defined, then in any condition depending on it, ms1 will be processed and ms2 omitted. conversely if it is undefined, ms1 will be omitted and ms2 processed.
nesting of conditions is such that conditions in a section not selected for processing must not be evaluated. nested conditions must remember their environment whilst being processed. effectively this implies use of a scheme based on a stack with .if, .fi matching by the condition processor of the translator.

section 7 - operand formats
the following section describes the various possibilities
for operands of instructions and assembly operations.

```
01   int              unsigned integer le cfp$l
02   dlbl             symbol defined in definitions sec
03   wlbl             label in working storage section
04   clbl             label in constant section
05   elbl             program section entry label
06   plbl             program section label (non-entry)
07   x                one of the three index registers
08   w                one of the three work registers
09   (x)              location indexed by x
10   (x)+             like (x) but post increment x
11   -(x)             like (x) but predecrement x
12   int(x)           location int words beyond addr in x
13   dlbl(x)          location dlbl words past addr in x
14   clbl(x)          location (x) bytes beyond clbl
15   wlbl(x)          location (x) bytes beyond wlbl
16   integer          signed integer (dic)
17   real             signed real (drc)
18   =dlbl            location containing dac dlbl
19   *dlbl            location containing dac cfp$b*dlbl
20   =wlbl            location containing dac wlbl
21   =clbl            location containing dac clbl
22   =elbl            location containing dac elbl
23   pnam             procedure label (on prc instruc)
24   eqop             operand for equ instruction
25   ptyp             procedure type (see prc)
26   text             arbitrary text (erb,err,ttl)
27   dtext            delimited text string (dtc)
```

the numbers in the above list are used in subsequent
description and in some of the minimal translators.

```
operand formats (continued)
the following special symbols refer to a collection of
the listed possibilities
val  01,02                      predefined value
     val is used to refer to a predefined one word
     integer value in the range 0 le n le cfp$l.
reg  07,08                      register
     reg is used to describe an operand which can be
     any of the registers (xl,xr,xs,xt,wa,wb,wc). such
     an operand can hold a one word integer (address).
opc  09,10,11                   character
     opc is used to designate a specific character
     operand for use in the lch and sch instructions.
     the index register referenced must be either xr or
     xl (not xs,xt). see section on character operations.
ops  03,04,09,12,13,14,15       memory reference
     ops is used to describe an operand which is in
     memory. the operand may be one or more words long
     depending on the data type. in the case of multiword
     operands, the address given is the first word.
opw  as for ops + 08,10,11      full word
     opw is used to refer to an operand whose capacity is
     that of a full memory word. opw includes all the
     possibilities for ops (the referenced word is used)
     plus the use of one of the three work registers
     (wa,wb,wc). in addition, the formats (x)+ and -(x)
     allow indexed operations in which the index register
     is popped by one word after the reference (x)+,
     or pushed by one word before the reference -(x)
     these latter two formats provide a facility for
     manipulation of stacks. the format does not imply
     a particular direction in which stacks must build -
     it is used for compactness. note that there is a
     restriction which disallows an instruction to use
     an index register in one of these formats
     in some other manner in the same instruction.
     e.g.   mov  xl,(xl)+   is illegal.
     the formats -(x) and (x)+ may also be used in
     pre-decrementation, post-incrementation to access
     the adjacent character of a string.
```

```
operand formats (continued)
opn  as for opw + 07            one word integer
     opn is used to represent an operand location which
     can contain a one word integer (e.g. an address).
     this includes all the possibilities for opw plus
     the use of one of the index registers (xl,xr,xt,
     xs). the range of integer values is 0 le n le cfp$l.
opv  as for opn + 18-22         one word integer value
     opv is used for an operand which can yield a one
     word integer value (e.g. an address). it includes
     all the possibilities for opn (the current value of
     the location is used) plus the use of literals. note
     that although the literal formats are described in
     terms of a reference to a location containing an
     address constant, this location may not actually
     exist in some implementations since only the value
     is required. a restriction is placed on literals
     which may consist only of defined symbols and
     certain labels. consequently small integers to be
     used as literals must be pre-defined, a discipline
     aiding program maintenance and revision.
addr 01,02,03,04,05             address
     addr is used to describe an explicit address value
     (one word integer value) for use with dac.
     ****************************************************
     *   in the following descriptions the usage --     *
     *      (xl),(xr), ... ,(ia)                         *
     *   in the descriptive text signifies the          +
     *   contents of the stated register.                *
     ****************************************************
```

section 8 - list of instruction mnemonics
the following list includes all instruction and
assembly operation mnemonics in alphabetical order.
the mnemonics are preceded by a number identifying
the following section where the instruction is described.
a star (*) is appended to the mnemonic if the last
operand may optionally be omitted.
see section -15- for details of statement format and
comment conventions.

```
 2.1  add  opv,opn      add address
 4.2  adi  ops          add integer
 5.3  adr  ops          add real
 7.1  anb  opw,w        and bit string
 2.17 aov  opv,opn,plbl add address, fail if overflow
 5.16 atn               arctangent of real accum
 2.16 bct  w,plbl       branch and count
 2.5  beq  opn,opv,plbl branch if address equal
 2.18 bev  opn,plbl     branch if address even
 2.8  bge  opn,opv,plbl branch if address greater or equl
 2.7  bgt  opn,opv,plbl branch if address greater
 2.12 bhi  opn,opv,plbl branch if address high
 2.10 ble  opn,opv,plbl branch if address less or equal
 2.11 blo  opn,opv,plbl branch if address low
 2.9  blt  opn,opv,plbl branch if address less than
 2.6  bne  opn,opv,plbl branch if address not equal
 2.13 bnz  opn,plbl     branch if address non-zero
 2.19 bod  opn,plbl     branch if address odd
 1.2  brn  plbl         branch unconditional
 1.7  bri  opn          branch indirect
 1.3  bsw* x,val,plbl   branch on switch value
 8.2  btw  reg          convert bytes to words
 2.14 bze  opn,plbl     branch if address zero
 6.6  ceq  opw,opw,plbl branch if characters equal
10.1  chk               check stack overflow
 5.17 chp               integer portion of real accum
 7.4  cmb  w            complement bit string
 6.8  cmc  plbl,plbl    compare character strings
 6.7  cne  opw,opw,plbl branch if characters not equal
 6.5  csc  x            complete store characters
 5.18 cos               cosine of real accum
 8.8  ctb  w,val        convert character count to bytes
 8.7  ctw  w,val        convert character count to words
 8.10 cvd               convert by division
 8.9  cvm  plbl         convert by multiplication
11.1  dac  addr         define address constant
11.5  dbc  val          define bit string constant
 2.4  dca  opn          decrement address by one word
 1.17 dcv  opn          decrement value by one
11.2  dic  integer      define integer constant
```

```
alphabetical list of mnemonics (continued)
11.3  drc  real         define real constant
11.4  dtc  dtext        define text (character) constant
 4.5  dvi  ops          divide integer
 5.6  dvr  ops          divide real
13.1  ejc               eject assembly listing
14.2  end               end of assembly
 1.13 enp               define end of procedure
 1.6  ent* val          define entry point
12.1  equ  eqop         define symbolic value
 1.15 erb  int,text     assemble error code and branch
 1.14 err  int,text     assemble error code
 1.5  esw               end of switch list for bsw
 5.19 etx               e to the power in the real accum
 1.12 exi* int          exit from procedure
12.2  exp               define external procedure
 6.10 flc  w            fold character to upper case
 2.3  ica  opn          increment address by one word
 3.4  icp               increment code pointer
 1.16 icv  opn          increment value by one
 4.11 ieq  plbl         jump if integer zero
 1.4  iff  val,plbl     specify branch for bsw
 4.12 ige  plbl         jump if integer non-negative
 4.13 igt  plbl         jump if integer positive
 4.14 ile  plbl         jump if integer negative or zero
 4.15 ilt  plbl         jump if integer negative
 4.16 ine  plbl         jump if integer non-zero
 4.9  ino  plbl         jump if no integer overflow
12.3  inp  ptyp,int     internal procedure
12.4  inr               internal routine
 4.10 iov  plbl         jump if integer overflow
 8.5  itr               convert integer to real
 1.9  jsr  pnam         call procedure
 6.3  lch  reg,opc      load character
 2.15 lct  w,opv        load counter for loop
 3.1  lcp  reg          load code pointer register
 3.3  lcw  reg          load next code word
 4.1  ldi  ops          load integer
 5.1  ldr  ops          load real
 1.8  lei  x            load entry point id
 5.20 lnf               natural logorithm of real accum
 7.6  lsh  w,val        left shift bit string
 7.8  lsx  w,(x)        left shift indexed
 9.4  mcb               move characterswords backwards
 8.4  mfi* opn,plbl     convert (ia) to address value
 4.3  mli  ops          multiply integer
 5.5  mlr  ops          multiply real
 1.19 mnz  opn          move non-zero
 1.1  mov  opv,opn      move
 8.3  mti  opn          move address value to (ia)
 9.1  mvc               move characters
 9.2  mvw               move words
 9.3  mwb               move words backwards
```

4.8 ngi negate integer

```
alphabetical list of mnemonics (continued)
 5.9  ngr              negate real
 7.9  nzb  w,plbl      jump if not all zero bits
 7.2  orb  opw,w       or bit strings
 6.1  plc* x,opv       prepare to load characters
 1.10 ppm* plbl        provide procedure exit parameter
 1.11 prc  ptyp,val    define start of procedure
 6.2  psc* x,opv       prepare to store characters
 5.10 req  plbl        jump if real zero
 5.11 rge  plbl        jump if real positive or zero
 5.12 rgt  plbl        jump if real positive
 5.13 rle  plbl        jump if real negative or zero
 5.14 rlt  plbl        jump if real negative
 4.6  rmi  ops         remainder integer
 5.15 rne  plbl        jump if real non-zero
 5.8  rno  plbl        jump if no real overflow
 5.7  rov  plbl        jump if real overflow
 7.5  rsh  w,val       right shift bit string
 7.7  rsx  w,(x)       right shift indexed
 8.6  rti* plbl        convert real to integer
 1.22 rtn              define start of routine
 4.4  sbi  ops         subtract integer
 5.4  sbr  ops         subtract reals
 6.4  sch  reg,opc     store character
 3.2  scp  reg         store code pointer
14.1  sec              define start of assembly section
 5.21 sin              sine of real accum
 5.22 sqr              square root of real accum
 1.20 ssl  opw         subroutine stack load
 1.21 sss  opw         subroutine stack store
 4.7  sti  ops         store integer
 5.2  str  ops         store real
 2.2  sub  opv,opn     subtract address
 5.23 tan              tangent of real accum
 6.9  trc              translate character string
13.2  ttl  text        supply assembly title
 8.1  wtb  reg         convert words to bytes
 7.3  xob  opw,w       exclusive or bit strings
 1.18 zer  opn         zeroise integer location
 7.11 zgb  opn         zeroise garbage bits
 7.10 zrb  w,plbl      jump if all zero bits
```

```
section 9 - minimal instructions
the following descriptions assume the definitions -
zeroe  equ  0
unity  equ  1
-1-  basic instruction set
1.1  mov  opv,opn     move one word value
     mov causes the value of operand opv to be set as
     the new contents of operand location opn. in the
     case where opn is not an index register, any value
     which can normally occupy a memory word (including
     a part of a multiword real or integer value)
     can be transferred using mov. if the target location
     opn is an index register, then opv must specify an
     appropriate one word value or operand containing
     such an appropriate value.
1.2  brn  plbl        unconditional branch
     brn causes control to be passed to the indicated
     label in the program section.
1.3  bsw  x,val,plbl  branch on switch value
1.4  iff  val,plbl    provide branch for switch
     iff  val,plbl     ...
     ...
     ...
1.5  esw              end of branch switch table
     bsw,iff,esw provide a capability for a switched
     branch similar to a fortran computed goto. the
     val on the bsw instruction is the maximum number
     of branches. the value in x ranges from zero up to
     but not including this maximum. each iff provides a
     branch. val must be less than that given on the bsw
     and control goes to plbl if the value in x matches.
     if the value in x does not correspond to any of the
     iff entries, then control passes to the plbl on the
     bsw. this plbl operand may be omitted if there are
     no values missing from the list.
     iff and esw may only be used in this context.
     execution of bsw may destroy the contents of x.
     the iff entries may be in any order and since
     a translator may thus need to store and sort them,
     the comment field is restricted in length (sec 11).
```

```
-1-  basic instructions (continued)
1.6  ent  val         define program entry point
     the symbol appearing in the label field is defined
     to be a program entry point which can subsequently
     be used in conjunction with the bri instruction,
     which provides the only means of entering the
     code. it is illegal to fall into code
     identified by an entry point. the
     entry symbol is assigned an address which need not
     be a multiple of cfp$b but which must be in the
     range 0 le cfp$l and the address must not lie within
     the address range of the allocated data area.
     furthermore, addresses of successive entry points
     must be assigned in some ascending sequence so
     that the address comparison instructions can be
     used to test the order in which two entry points
     occur. the symbol val gives an identifying value
     to the entry point which can be accessed with the
     lei instruction.
     note - subject to the restriction below, val may
     be omitted if no such identification is needed i.e.
     if no lei references the entry point. for this
     case, a translation optimisation is possible in
     which no memory need be reserved for a null
     identification which is never to be referenced, but
     only provided this is done so as not to interfere
     with the strictly ascending sequence of entry point
     addresses. to simplify this optimisation for all
     implementors, the following restriction is observed
          val may only be omitted if the entry point is
          separated from a following entry point by a
          non-null minimal code sequence.
     entry point addresses are accessible only by use of
     literals (=elbl, section 7) or dac constants
     (section 8-11.1).
1.7  bri  opn         branch indirect
     opn contains the address of a program entry point
     (see ent). control is passed to the executable
     code starting at the entry point address. opn is
     left unchanged.
1.8  lei  x           load entry point identification
     x contains the address of an entry point for which
     an identifying value was given on the the ent line.
     lei replaces the contents of x by this value.
```

```
-1-  basic instructions (continued)
1.9  jsr  pnam        call procedure pnam
1.10 ppm  plbl        provide exit parameter
     ppm  plbl         ...
     ...
     ppm  plbl         ...
     jsr causes control to be passed to the named
     procedure. pnam is the label on a prc statement
     elsewhere in the program section (see prc)
     or has been defined using an exp instruction.
     the ppm exit parameters following the call give
     names of program locations (plbl-s) to which
     alternative exi returns of the called procedure may
     pass control. they may optionally be replaced by
     error returns (see err). the number of exit
     parameters following a jsr must equal the int in the
     procedure definition. the operand of ppm may be
     omitted if the corresponding exi return is certain
     not to be taken.
1.11 prc  ptyp,int    define start of procedure
     the symbol appearing in the label field is defined
     to be the name of a procedure for use with jsr.
     a procedure is a contiguous section of instructions
     to which control may be passed with a jsr
     instruction. this is the only way in which the
     instructions in a procedure may be executed. it is
     not permitted to fall into a procedure.
     all procedures should be named in section 0
     inp  statements.
     int is the number of exit parameters (ppm-s) to
     be used in jsr calls.
     there are three possibilities for ptyp, each
     consisting of a single letter as follows.
     r                recursive
     the return point (one or more words) is stored on
     the stack as though one or more mov ...,-(xs)
     instructions were executed.
```

```
-1-  basic instructions (continued)
     n                non-recursive
     the return point is to be stored either
     (1) in a local storage word associated
     with the procedure and not directly
     available to the program in any other manner or
     (2) on a subroutine link stack quite distinct from
     the minimal stack addressed by xs.
     it is an error to use the stack for n-links, since
     procedure parameters or results may be passed via
     the stack.
     if method (2) is used for links, error exits
     (erb,err) from a procedure will necessitate link
     stack resetting. the ssl and sss orders provided
     for this may be regarded as no-ops for
     implementations using method (1).
     e                either
     the return point may be stored in either manner
     according to efficiency requirements of the actual
     physical machine used for the implementation. note
     that programming of e type procedures must be
     independent of the actual implementation.
     the actual form of the return point is undefined.
     however, each word stored on the stack for an
     r-type call must meet the following requirements.
     1)               it can be handled as an address
                      and placed in an index register.
     2)               when used as an operand in an
                      address comparison instruction, it
                      must not appear to lie within
                      the allocated data area.
     3)               it is not required to appear
                      to lie within the program section.
```

```
-1-  basic instructions (continued)
1.12 exi  int         exit from procedure
     the ppm and err parameters following a jsr are
     numbered starting from 1.  exi  int causes control
     to be returned to the int-th such param. exi 1 gives
     control to the plbl of the first ppm after the jsr.
     if int is omitted, control is passed back past the
     last exit parameter (or past the jsr if there are
     none). for r and e type procedures, the
     stack pointer xs must be set to its appropriate
     entry value before executing an exi instruction.
     in this case, exi removes return points from the
     stack if any are stored there so that the stack
     pointer is restored to its calling value.
1.13 enp              define end of procedure body
     enp delimits a procedure body and may not actually
     be executed, hence it must have no label.
1.14 err  int,text    provide error return
     err may replace an exit parameter (ppm) in
     any procedure call. the int argument is a unique
     error code in 0 to 899.
     the text supplied as the other operand is
     arbitrary text in the fortran character set and
     may be used in constructing a file of error
     messages for documenting purposes or for building
     a direct access or other file of messages to be
     used by the error handling code.
     in the event that an exi attempts
     to return control via an exit parameter to
     an err, control is instead passed to the first
     instruction in the error section (which follows the
     program section) with the error code in wa.
1.15 erb  int,text    error branch
     this instruction resembles err except that it may
     occur at any point where a branch is permitted.
     it effects a transfer of control to the error
     section with the error code in wa.
1.16 icv  opn         increment value by one
     icv increments the value of the operand by unity.
     it is equivalent to   add  =unity,opn
1.17 dcv  opn         decrement value by one
     dcv decrements the value of the operand by unity.
     it is equivalent to   sub  =unity,opn
```

```
basic instructions (continued)
1.18 zer  opn         zeroise opn
     zer is equivalent to  mov  =zeroe,opn
1.19 mnz  opn         move non-zero to opn
     any non-zero collectable value may used, for which
     the opcodes bnz/bze will branch/fail to branch.
1.20 ssl  opw         subroutine stack load
1.21 sss  opw         subroutine stack store
     this pair of operations is provided to make possible
     the use of a local stack to hold subroutine (s-r)
     return links for n-type procedures. sss stores the
     s-r stack pointer in opw and ssl loads the s-r
     stack pointer from opw. by using sss in the main
     program or on entry to a procedure which should
     regain control on occurrence of an err or erb and by
     use of ssl in the error processing sections the
     s-r stack pointer can be restored giving a link
     stack cleaned up ready for resumed execution.
     the form of the link stack pointer is undefined in
     minimal (it is likely to be a private register
     known to the translator) and the only requirement
     is that it should fit into a single full word.
     ssl and sss are no-ops if a private link stack is
     not used.
1.22 rtn              define start of routine
     a routine is a code chunk used for similar purposes
     to a procedure. however it is entered by any type of
     conditional or unconditional branch (not by jsr). on
     termination it passes control by a branch (often
     bri through a code word) or even permits control
     to drop through to another routine. no return link
     exists and the end of a routine is not marked by
     an explicit opcode (compare enp).
     all routines should be named in section 0
     inr  statements.
```

```
-2-  operations on one word integer values (addresses)
2.1  add  opv,opn     adds opv to the value in opn and
                      stores the result in opn. undefined
                      if the result exceeds cfp$l.
2.2  sub  opv,opn     subtracts opv from opn. stores the
                      result in opn. undefined if the
                      result is negative.
2.3  ica  opn         increment address in opn
                      equivalent to add *unity,opn
2.4  dca  opn         decrement address in opn
                      equivalent to sub *unity,opn
2.5  beq  opn,opv,plbl branch to plbl if opn eq opv
2.6  bne  opn,opv,plbl branch to plbl if opn ne opv
2.7  bgt  opn,opv,plbl branch to plbl if opn gt opv
2.8  bge  opn,opv,plbl branch to plbl if opn ge opv
2.9  blt  opn,opv,plbl branch to plbl if opn lt opv
2.10 ble  opn,opv,plbl branch to plbl if opn le opv
2.11 blo  opn,opv,plbl equivalent to blt or ble
2.12 bhi  opn,opv,plbl equivalent to bgt or bge
     the above instructions compare two address
     values as unsigned integer values.
     the blo and bhi instructions are used in cases where
     the equal condition either does not occur or can
     result either in a branch or no branch. this avoids
     inefficient translations in some implementations.
2.13 bnz  opn,plbl    equivalent to bne opn,=zeroe,plbl
2.14 bze  opn,plbl    equivalent to beq opn,=zeroe,plbl
2.15 lct  w,opv       load counter for bct
     lct loads a counter value for use with the bct
     instruction. the value in opv is the number of loops
     to be executed. the value in w after this operation
     is an undefined one word integer quantity.
2.16 bct  w,plbl      branch and count
     bct uses the counter value in w to branch the
     required number of times and then finally to fall
     through to the next instruction. bct can only be
     used following an appropriate lct instruction.
     the value in w after execution of bct is undefined.
2.17 aov  opv,opn,plbl add with carry test
     adds opv to the value in opn and stores result in
     opn. branches to plbl if result exceeds cfp$l
     with result in opn undefined. cf. add.
2.18 bev  opn,plbl     branch if even
2.19 bod  opn,plbl     branch if odd
     these operations are used only if .cepp or .crpp is
     defined.  on some implementations, a more efficient
     implementation is possible by noting that address of
     blocks must always be a multiple of cfp$b. we call
     such addresses even.  thus return address on the
     stack (.crpp) and entry point addresses (.cepp) can
     be distinguished from block addresses if they are
     forced to be odd (not a multiple of cfp$b).
     bev and bod branch according as operand is even
```

or odd, respectively.

```
-3-  operations on the code pointer register (cp)
     the code pointer register provides a psuedo
     instruction counter for use in an interpretor. it
     may be implemented as a real register or as a
     memory location, but in either case it is separate
     from any other register. the value in the code
     pointer register is always a word address (i.e.
     a one word integer which is a multiple of cfp$b).
3.1  lcp  reg          load code pointer register
                       this instruction causes the code
                       pointer register to be set from
                       the value in reg which is unchanged
3.2  scp  reg          store code pointer register
                       this instruction loads the current
                       value in the code pointer register
                       into reg. (cp) is unchanged.
3.3  lcw  reg          load next code word
                       this instruction causes the word
                       pointed to by cp to be loaded into
                       the indicated reg. the value in cp
                       is then incremented by one word.
                       execution of lcw may destroy xl.
3.4  icp               increment cp by one word
     on machines with more than three index registers,
     cp can be treated simply as an index register.
     in this case, the following equivalences apply.
     lcp reg is like mov reg,cp
     scp reg is like mov cp,reg
     lcw reg is like mov (cp)+,reg
     icp     is like ica cp
     since lcw is allowed to destroy xl, the following
     implementation using a work location cp$$$ can
     also be used.
     lcp  reg          mov  reg,cp$$$
     scp  reg          mov  cp$$$,reg
     lcw  reg          mov  cp$$$,xl
                       mov  (xl)+,reg
                       mov  xl,cp$$$
     icp               ica  cp$$$
```

```
-4-  operations on signed integer values
4.1  ldi  ops        load integer accumulator from ops
4.2  adi  ops        add ops to integer accumulator
4.3  mli  ops        multiply integer accumulator by ops
4.4  sbi  ops        subtract ops from int accumulator
4.5  dvi  ops        divide integer accumulator by ops
4.6  rmi  ops        set int accum to mod(intacc,ops)
4.7  sti  ops        store integer accumulator at ops
4.8  ngi             negate the value in the integer
                     accumulator (change its sign)
     the equation satisfied by operands and results of
     dvi and rmi is
            div = qot * ops + rem          where
     div = dividend in integer accumulator
     qot = quotient left in ia by div
     ops = the divisor
     rem = remainder left in ia by rmi
     the sign of the result of dvi is + if (ia) and (ops)
     have the same sign and is - if they have opposite
     signs. the sign of (ia) is always used as the sign
     of the result of rem.
     assuming in each case that ia contains the number
     specified in parentheses and that seven and msevn
     hold +7 and -7 resp. the algorithm is illustrated
     below.
     (ia = 13)
     dvi  seven      ia = 1
     rmi  seven      ia = 6
     dvi  msevn      ia = -1
     rmi  msevn      ia = 6
     (ia = -13)
     dvi  seven      ia = -1
     rmi  seven      ia = -6
     dvi  msevn      ia = 1
     rmi  msevn      ia = -6
```

```
     the above instructions operate on a full range of
     signed integer values. with the exception of ldi and
     sti, these instructions may cause integer overflow
     by attempting to produce an undefined or out of
     range result in which case integer overflow is set,
     the result in (ia) is undefined and the following
     instruction must be iov or ino.
     particular care may be needed on target machines
     having distinct overflow and divide by zero
     conditions.
4.9  ino  plbl       jump to plbl if no integer overflow
4.10 iov  plbl       jump to plbl if integer overflow
     these instructions can only occur immediately
     following an instruction which can cause integer
     overflow (adi, sbi, mli, dvi, rmi, ngi) and
     test the result of the preceding instruction.
     iov and ino may not have labels.
4.11 ieq  plbl       jump to plbl if (ia) eq 0
4.12 ige  plbl       jump to plbl if (ia) ge 0
4.13 igt  plbl       jump to plbl if (ia) gt 0
4.14 ile  plbl       jump to plbl if (ia) le 0
4.15 ilt  plbl       jump to plbl if (ia) lt 0
4.16 ine  plbl       jump to plbl if (ia) ne 0
     the above conditional jump instructions do
     not change the contents of the accumulator.
     on a ones complement machine, it is permissible to
     produce negative zero in ia provided these
     instructions operate correctly with such a value.
```

```
-5-  operations on real values
5.1  ldr  ops         load real accumulator from ops
5.2  str  ops         store real accumulator at ops
5.3  adr  ops         add ops to real accumulator
5.4  sbr  ops         subtract ops from real accumulator
5.5  mlr  ops         multiply real accumulator by ops
5.6  dvr  ops         divide real accumulator by ops
     if the result of any of the above operations causes
     underflow, the result yielded is 0.0.
     if the result of any of the above operations is
     undefined or out of range, real overflow is set,
     the contents of (ra) are undefined and the following
     instruction must be either rov or rno.
     particular care may be needed on target machines
     having distinct overflow and divide by zero
     conditions.
5.7  rov  plbl        jump to plbl if real overflow
5.8  rno  plbl        jump to plbl if no real overflow
     these instructions can only occur immediately
     following an instruction which can cause real
     overflow (adr,sbr,mlr,dvr).
5.9  ngr              negate real accum (change sign)
5.10 req  plbl        jump to plbl if (ra) eq 0.0
5.11 rge  plbl        jump to plbl if (ra) ge 0.0
5.12 rgt  plbl        jump to plbl if (ra) gt 0.0
5.13 rle  plbl        jump to plbl if (ra) le 0.0
5.14 rlt  plbl        jump to plbl if (ra) lt 0.0
5.15 rne  plbl        jump to plbl if (ra) ne 0.0
     the above conditional instructions do not affect
     the value stored in the real accumulator.
     on a ones complement machine, it is permissible to
     produce negative zero in ra provided these
     instructions operate correctly with such a value.
```

---

if **.cmth**

```
5.16 atn              arctangent of real accum
5.17 chp              integer portion of real accum
5.18 cos              cosine of real accum
5.19 etx              e to the power in the real accum
5.20 lnf              natural logorithm of real accum
5.21 sin              sine of real accum
5.22 sqr              square root of real accum
5.23 tan              tangent of real accum
     the above orders operate upon the real accumulator,
     and replace the contents of the accumulator with the
     result.
     if the result of any of the above operations is
     undefined or out of range, real overflow is set,
     the contents of (ra) are undefined and the following
     instruction must be either rov or rno.
```

fi

```
-6-  operations on character values
     character operations employ the concept of a
     character pointer which uses either
     index register xr or xl (not xs).
     a character pointer points to a specific character
     in a string of characters stored cfp$c chars to a
     word. the only operations permitted on a character
     pointer are lch and sch. in particular, a character
     pointer may not even be moved with mov.
        restriction 1.
        --------------
     it is important when coding in minimal to ensure
     that no action occurring between the initial use of
     plc or psc and the eventual clearing of xl or xr on
     completion of character operations can initiate a
     garbage collection. the latter of course could cause
     the addressed characters to be moved leaving the
     character pointers pointing to rubbish.
        restriction 2.
        --------------
     a further restriction to be observed in code
     handling character strings, is that strings built
     dynamically should be right padded with zero
     characters to a full word boundary to permit easy
     hashing and use of ceq or cne in testing strings
     for equality.
6.1  plc  x,opv       prepare ch ptr for lch,cmc,mvc,trc,
                      mcb.
6.2  psc  x,opv       prepare char. ptr for sch,mvc,mcb.
     opv can be omitted if it is zero.
     the char. initially addressed is determined by the
     word address in x and the integer offset opv.
     there is an automatic implied offset of cfp$f bytes.
     cfp$f is used to formally introduce into minimal a
     value needed in translating these opcodes which,
     since minimal itself does not prescribe a string
     structure in detail, depends on the choice of a data
     structure for strings in the minimal program.
     e.g. if cfp$b = cfp$c = 3, cfp$f = 6, num01 = 1, xl
     points to a series of 4 words, abc/def/ghi/jkl, then
         plc  xl,=num01
     points to h.
```

```
-6- operations on character values (continued)
6.3  lch  reg,opc     load character into reg
6.4  sch  reg,opc     store character from reg
     these operations are defined such that the character
     is right justified in register reg with zero bits to
     the left. after lch for example, it is legitimate
     to regard reg as containing the ordinal integer
     corresponding to the character.
     opc is one of the following three possibilities.
     (x)              the character pointed to by the
                      character pointer in x. the
                      character pointer is not changed.
     (x)+             same character as (x) but the
                      character pointer is incremented
                      to point to the next character
                      following execution.
     -(x)             the character pointer is decre-
                      mented before accessing the
                      character so that the previous
                      character is referenced.
6.5  csc  x           complete store characters
     this instruction marks completion of a
     psc,sch,sch,...,sch sequence initiated by
     a  psc  x instruction. no more sch instructions
     using  x  should be obeyed until another psc is
     obeyed. it is provided solely as an efficiency aid
     on machines without character orders since it
     permits use of register buffering of chars in sch
     sequences. where csc is not a no-op, it must observe
     restriction 2. (e.g. in spitbol, alocs zeroises the
     last word of a string frame prior to sch sequence
     being started so csc must not nullify this action.)
     the following instructions are used to compare
     two words containing cfp$c characters.
     comparisons distinct from beq,bne are provided as
     on some target machines, the possibility of the sign
     bit being set may require special action.
     note that restriction 2 above, eases use of these
     orders in testing complete strings for equality,
     since whole word tests are possible.
6.6  ceq  opw,opw,plbl jump to plbl if opw eq opw
6.7  cne  opw,opw,plbl jump to plbl if opw ne opw
```

```
-6- operations on character values (continued)
6.8  cmc  plbl,plbl   compare characters
     cmc is used to compare two character strings. before
     executing cmc, registers are set up as follows.
     (xl)             character ptr for first string
     (xr)             character pointer for second string
     (wa)             character count (must be .gt. zero)
     xl and xr should have been prepared by plc.
     control passes to first plbl if the first string
     is lexically less than the second string, and to
     the second plbl if the first string is lexically
     greater. control passes to the following instruction
     if the strings are identical. after executing this
     instruction, the values of xr and xl are set to
     zero and the value in (wa) is undefined.
     arguments to cmc may be complete or partial
     strings, so making optimisation to use whole word
     comparisons difficult (dependent in general on
     shifts and masking).
6.9  trc              translate characters
     trc is used to translate a character string using a
     supplied translation table. before executing trc the
     registers are set as follows.
     (xl)             char ptr to string to be translated
     (xr)             char ptr to translate table
     (wa)             length of string to be translated
     xl and xr should have been prepared by plc.
     the translate table consists of cfp$a contiguous
     characters giving the translations of the cfp$a
     characters in the alphabet. on completion, (xr) and
     (xl) are set to zero and (wa) is undefined.
6.10 flc  w           fold character to upper case
     flc is used only if .culc is defined. the character
     code value in w is translated to upper case if it
     corresponds to a lower case character.
```

```
-7-  operations on bit string values
7.1  anb  opw,w       and bit string values
7.2  orb  opw,w       or bit string values
7.3  xob  opw,w       exclusive or bit string values
     in the above operations, the logical connective is
     applied separately to each of the cfp$n bits.
     the result is stored in the second operand location.
7.4  cmb  w           complement all bits in opw
7.5  rsh  w,val       right shift by val bits
7.6  lsh  w,val       left shift by val bits
7.7  rsx  w,(x)       right shift w number of bits in x
7.8  lsx  w,(x)       left shift w number of bits in x
     the above shifts are logical shifts in which bits
     shifted out are lost and zero bits supplied as
     required. the shift count is in the range 0-cfp$n.
7.9  nzb  w,plbl      jump to plbl if w is not
                      all zero bits.
7.10 zrb  w,plbl      jump to plbl if w is all zero bits
7.11 zgb  opn         zeroise garbage bits
     opn contains a bit string representing a word
     of characters from a string or some function
     formed from such characters (e.g. as a result of
     hashing). on a machine where the word size is not a
     multiple of the character size, some bits in reg may
     be undefined. this opcode replaces such bits by the
     zero bit. zgb is a no-op if the word size is a
     multiple of the character size.
```

```
-8-  conversion instructions
     the following instructions provide for conversion
     between lengths in bytes and lengths in words.
8.1  wtb  reg         convert reg from words to bytes.
                      that is, multiply by cfp$b. this is
                      a no-op if cfp$b is one.
8.2  btw  reg         convert reg from bytes to words
                      by dividing reg by cfp$b discarding
                      the fraction. no-op if cfp$b is one
     the following instructions provide for conversion
     of one word integer values (addresses) to and
     from the full signed integer format.
8.3  mti  opn         the value of opn (an address)
                      is moved as a positive integer
                      to the integer accumulator.
8.4  mfi  opn,plbl    the value currently stored in the
                      integer accumulator is moved
                      to opn as an address if it is in
                      the range 0 to cfp$m inclusive.
                      if the accumulator value is
                      outside this range, then the result
                      in opn is undefined and control is
                      passed to plbl. mfi destroys the
                      value of (ia) whether or not
                      integer overflow is signalled.
                      plbl may be omitted if overflow
                      is impossible.
     the following instructions provide for conversion
     between real values and integer values.
8.5  itr              convert integer value in integer
                      accumulator to real and store in
                      real accumulator (may lose
                      precision in some cases)
8.6  rti  plbl        convert the real value in ra to
                      an integer and place result in ia.
                      conversion is by truncation of the
                      fraction - no rounding occurs.
                      jump to plbl if out of range. (ra)
                      is not changed in either case.
                      plbl may be omitted if overflow
                      is impossible.
```

```
-8-  conversion instructions (continued)
     the following instructions provide for computing
     the length of storage required for a text string.
8.7  ctw  w,val       this instruction computes the sum
                      (number of words required to store
                      w characters) + (val). the sum
                      is stored in w.
                      for example, if cfp$c is 5, and wa
                      contains 32, then ctw wa,2
                      gives a result of 9 in wa.
8.8  ctb  w,val       ctb is exactly like ctw except that
                      the result is in bytes. it has the
                      same effect as ctw w,val  wtb w
     the following instructions provide for conversion
     from integers to and from numeric digit characters
     for use in numeric conversion routines. they employ
     negative integer values to allow for proper
     conversion of numbers which cannot be complemented.
8.9  cvm  plbl        convert by multiplication
     the integer accumulator, which is zero or negative,
     is multiplied by 10. wb contains the character
     code for a digit. the value of this digit is then
     subtracted from the result. if the result is out of
     range, then control is passed to plbl with the
     result in (ia) undefined. execution of cvm leaves
     the result in (wb) undefined.
8.10 cvd              convert by division
     the integer accumulator, which is zero or negative,
     is divided by 10. the quotient (zero or negative)
     is replaced in the accumulator. the remainder is
     converted to the character code of a digit and
     placed in wa. for example, an operand of -523 gives
     a quotient of -52 and a remainder in wa of ch$d3.
```

```
-9-  block move instructions
the following instructions are used for transferring
data from one area of memory to another in blocks.
they can be implemented with the indicated series of
other macro-instructions, but more efficient imple-
mentations will be possible on most machines.
note that in the equivalent code sequence shown below, a
zero value in wa will move at least one item, and may
may wrap the counter causing a core dump in some imple-
mentations.  thus wa should be .gt. 0 prior to invoking
any of these block move instructions.
9.1  mvc              move characters
     before obeying this order wa,xl,xr should have been
     set up, the latter two by plc, psc resp.
     mvc is equivalent to the sequence
            mov  wb,dumpb
            lct  wa,wa
     loopc  lch  wb,(xl)+
            sch  wb,(xr)+
            bct  wa,loopc
            csc  xr
            mov  dumpb,wb
     the character pointers are bumped as indicated
     and the final value of wa is undefined.
9.2  mvw              move words
     mvw is equivalent to the sequence
     loopw  mov  (xl)+,(xr)+
            dca  wa               wa = bytes to move
            bnz  wa,loopw
     note that this implies that the value in wa is the
     length in bytes which is a multiple of cfp$b.
     the initial addresses in xr,xl are word addresses.
     as indicated, the final xr,xl values point past the
     new and old regions of memory respectively.
     the final value of wa is undefined.
     wa,xl,xr must be set up before obeying mvw.
9.3  mwb              move words backwards
     mwb is equivalent to the sequence
     loopb  mov  -(xl),-(xr)
            dca  wa               wa = bytes to move
            bnz  wa,loopb
     there is a requirement that the initial value in xl
     be at least 256 less than the value in xr. this
     allows an implementation in which chunks of 256
     bytes are moved forward (ibm 360, icl 1900).
     the final value of wa is undefined.
     wa,xl,xr must be set up before obeying mwb.
9.4  mcb              move characters backwards
     mcb is equivalent to the sequence
            mov  wb,dumpb
            lct  wa,wa
     loopc  lch  wb,-(xl)
            sch  wb,-(xr)
```

```
       bct  wa,loopc
       csc  xr
       mov  dumpb,wb
```

there is a requirement that the initial value in xl
be at least 256 less than the value in xr. this
allows an implementation in which chunks of 256
bytes are moved forward (ibm 360, icl 1900).
the final value of wa is undefined.
wa,xl,xr must be set up before obeying mcb.

-10- operations connected with the stack
the stack is an area in memory which is dedicated for use in conjunction with the stack pointer register (xs). as previously described, it is used by the jsr and exi instructions and may be used for storage of any other data as required.
the stack builds either way in memory and an important restriction is that the value in (xs) must be the address of the stack front at all times since
some implementations may randomly destroy stack locations beyond (xs).
the starting stack base address is passed
in (xs) at the start of execution. during execution it is necessary to make sure that the stack does not overflow. this is achieved by executing the following instruction periodically.

```
10.1 chk              check stack overflow
```

after successfully executing chk, it is permissible to use up to 100 additional words before issuing another chk thus chk need not be issued every time the stack is expanded. in some implementations, the checking may be automatic and chk will have no effect. following the above rule makes sure that the program will operate correctly in implementations with no automatic check.
if stack overflow occurs (detected either automatically or by a chk instruction), then control is passed to the stack overflow section (see program form). note that this transfer may take place following any instruction which stores data at a new location on the stack.
after stack overflow, stack is arbitrarily popped
to give some space in which the error procedure may operate. otherwise a loop of stack overflows may occur.

```
-11- data generation instructions
the following instructions are used to generate constant
values in the constant section and also to assemble
initial values in the working storage section. they
may not appear except in these two sections.
11.1 dac  addr       assemble address constant.
                     generates one word containing the
                     specified one word integer
                     value (address).
11.2 dic  integer    generates an integer value which
                     occupies cfp$i consecutive words.
                     the operand is a digit string with
                     a required leading sign.
11.3 drc  real       assembles a real constant which
                     occupies cfp$r consecutive words.
                     the operand form must obey the
                     rules for a fortran real constant
                     with the extra requirement that a
                     leading sign be present.
11.4 dtc  dtext      define text constant. dtext
                     is started and ended with any
                     character not contained in the
                     characters to be assembled. the
                     constant occupies consecutive words
                     as dictated by the configuration
                     parameter cfp$c. any unused chars
                     in the last word are right filled
                     with zeros (i.e. the character
                     whose internal code is zero).
                     the string contains a sequence of
                     letters, digits, blanks and any of
                     the following special characters.
                     =,$.(*)/+-
                     no other characters
                     may be used in a dtext operand.
11.5 dbc  val        assemble bit string constant. the
                     operand is a positive integer
                     value which is interpreted in
                     binary, right justified and left
                     filled with zero bits. thus 5 would
                     imply the bit string value 00...101.
```

```
-12- symbol definition instructions
the following instruction is used to define symbols
in the definitions section. it may not be used elsewhere.
12.1 equu  eqop        define symbol
     the symbol which appears in the label field is
     defined to have the absolute value given
     by the eqop operand. a given symbol may be defined
     only once in this manner, and any symbols occuring
     in eqop must be previously defined.
     the following are the possibilities for eqop
     val              the indicated value is used
     val+val          the sum of the two values is used.
                      this sum must not exceed cfp$m
     val-val          the difference between the two
                      values (must be positive) is used.
     *                this format defines the label by
                      using a value supplied by the
                      minimal translator. values are
                      required for the
     cfp$x            (configuration parameters)
     e$xxx            (environment parameters)
     ch$xx            (character codes).
                      in order for a translator to
                      handle this format correctly the
                      definitions section must be
                      consulted for details of required
                      symbols as listed at the front of
                      the section.
```

```
symbol definition instructions (continued)
the following instructions may be used to define symbols
in the procedure section. they may not be used in
any other part of the program.
12.2 exp              define external procedure
     exp defines the symbol appearing in the label field
     to be the name of an external procedure which can be
     referenced in a subsequent jsr instruction. the
     coding for the procedure is external to the
     coding of the source program in this language.
     the code for external procedures may be
     referred to collectively as the operating system
     interface, or more briefly, osint, and will
     frequently be a separately compiled segment of code
     loaded with spitbol to produce a complete system.
12.3 inp  ptyp,int    define internal procedure
     inp defines the symbol appearing in the label field
     to be the name of an internal procedure and gives
     its type and number of exit parameters. the label
     can be referenced in jsr instructions and
     it must appear labelling a prc instruction in the
     program section.
12.4 inr              define internal routine
     inr defines the symbol appearing in the label
     field to be the name of an internal routine. the
     label may be referenced in any type of branch order
     and it must appear labelling a rtn instruction in
     the program section.
```

```
-13- assembly listing layout instructions
13.1 ejc              eject to next page
13.2 ttl  text        set new assembly title
     ttl implies an immediate eject of the
     assembly listing to print the new title.
     the use of ttl and ejc cards is such that the
     program will list neatly if the printer prints
     as many as 58 lines per page. in the event that
     the printer depth is less than this, or if the
     listing contains interspersed lines (such as actual
     generated code), then the format may be upset.
     lines starting with an asterisk are comment lines
     which cause no code to be generated and may occur
     freely anywhere in the program. the format for
     comment lines is given in section -15-.
```

```
-14- program form
     the program consists of separate sections separated
     by sec operations. the sections must appear in the
     following specified order.
14.1 sec              start of procedure section
     (procedure section)
     sec              start of definitions section
     (definitions section)
     sec              start of constant storage section
     (constant storage section)
     sec              start of working storage section
     (working storage section)
     sec              start of program section
     (program section)
     sec              start of stack overflow section
     (stack overflow section)
     sec              start of error section
     (error section)
14.2 end              end of assembly
```

```
section 10 - program form
procedure section
     the procedure section contains all the exp
     instructions for externally available procedures
     and inp,inr opcodes for internal procedures,routines
     so that a single pass minimal translator has advance
     knowledge of procedure types when translating calls.
definitions section
     the definitions section contains equ instructions
     which define symbols referenced later on in the
     program, constant and work sections.
constant storage section
     the constant storage section consists entirely
     of constants assembled with the dac,dic,drc,dtc,dbc
     assembly operations. these constants can be freely
     referenced by the program instructions.
working storage section
     the working storage section consists entirely of
     dac,dic,drc,dbc,dtc instructions to define a fixed
     length work area. the work locations in this area
     can be directly referenced in program instructions.
     the area is initialized in accordance with the
     values assembled in the instructions.
program section
     the program section contains program instructions
     and associated operations (such as prc, enp, ent).
     control is passed to the first instruction in this
     section when execution is initiated.
stack overflow section
     the stack overflow section contains instructions
     like the program section. control is passed to the
     first instruction in this section following the
     occurrence of stack overflow, see chk instruction.
error section
     the error section contains instructions like the
     program section. control is passed to the first
     instruction in this section when a procedure exit
     corresponds to an error parameter (see err)
     or when an erb opcode is obeyed. the error code
     must clean up the main stack and cater for the
     possibility that a subroutine stack may need clean
     up.
```

osint

though not part of the minimal source, it is useful to refer to the collection of initialisation and exp routines as osint (operating system interface). errors occurring within osint procedures are usually handled by making an error return. if this is not feasible or appropriate, osint may use the minimal error section to report errors directly by branching to it with a suitable numeric error code in wa.

```
section 11 - statement format
all labels are exactly five characters long and start
with three letters (abcdefghijklmnopqrstuvwxy$) followed
by two letters or digits.
the letter z may not be used in minimal symbols but $ is
permitted.
for implementations where $ may not appear in the
target code , a simple substitution of z for $
may thus be made without risk of producing non-unique
symbols.
the letter z is however permitted in opcode mnemonics and
in comments.
minimal statements are in a fixed format as follows.
cols 1-5             label if any (else blank)
cols 6-7             always blank
cols 8-10            operation mnemonic
cols 11-12           blanks
cols 13-28           operand field, terminated by a
                     blank. may occasionally
                     extend past column 28.
cols 30-64           comment. always separated from the
                     operand field by at least one blank
                     may occasionally start after column
                     30 if the operand extends past 28.
                     a special exception occurs for the
                     iff instruction, whose comment may
                     be only 20 characters long (30-49).
cols 65 on           unused
comment lines have the following format
col 1                asterisk
cols 2-7             blank
cols 8-64            arbitrary text, restricted to the
                     fortran character set.
the fortran character set is a-z 0-9 =,$.(*)-/+
```

```
section 12 - program execution
execution of the program begins with the first
instruction in the program section.
in addition to the fixed length memory regions defined
by the assembly, there are two dynamically allocated
memory regions as follows.
data area             this is an area available to the
                      program for general storage of data
                      any data value may be stored in
                      this area except instructions.
                      in some implementations, it may be
                      possible to increase the size of
                      this area dynamically by adding
                      words at the top end with a call
                      to a system procedure.
stack area            this region of memory holds
                      the stack used for subroutine calls
                      and other storage of one word
                      integer values (addresses). this
                      is the stack associated with
                      index register xs.
the locations and sizes of these areas are specified
by the values in the registers at the start of program
execution as follows.
(xs)                  address one past the stack base.
                      e.g. if xs is 23456, a d-stack will
                      occupy words 23455,23454,...
                      whereas a u-stack will occupy
                      23457,23458,...
(xr)                  address of the first word
                      in the data area
(xl)                  address of the last word in the
                      data area.
(wa)                  initial stack pointer
(wb,wc,ia,ra,cp)      zero
there is no explicit way to terminate the execution of a
program. this function is performed by an appropriate
system procedure referenced with the sysej instruction.
```

```
general structure
-----------------
this program is a translator for a version of the snobol4
programming language. language details are contained in
the manual macro spitbol by dewar and mccann, technical
report 90, university of leeds 1976.
the implementation is discussed in dewar and mccann,
macro spitbol - a snobol4 compiler, software practice and
experience, 7, 95-113, 1977.
the language is as implemented by the btl translator
(griswold, poage and polonsky, prentice hall, 1971)
with the following principal exceptions.
1)   redefinition of standard system functions and
     operators is not permitted.
2)   the value function is not provided.
3)   access tracing is provided in addition to the
     other standard trace modes.
4)   the keyword stfcount is not provided.
5)   the keyword fullscan is not provided and all pattern
     matching takes place in fullscan mode (i.e. with no
     heuristics applied).
6)   a series of expressions separated by commas may
     be grouped within parentheses to provide a selection
     capability. the semantics are that the selection
     assumes the value of the first expression within it
     which succeeds as they are evaluated from the left.
     if no expression succeeds the entire statement fails
7)   an explicit pattern matching operator is provided.
     this is the binary query (see gimpel sigplan oct 74)
8)   the assignment operator is introduced as in the
     gimpel reference.
9)   the exit function is provided for generating load
     modules - cf. gimpels sitbol.
the method used in this program is to translate the
source code into an internal pseudo-code (see following
section). an interpretor is then used to execute this
generated pseudo-code. the nature of the snobol4 language
is such that the latter task is much more complex than
the actual translation phase. accordingly, nearly all the
code in the program section is concerned with the actual
execution of the snobol4 program.
```

interpretive code format
----

the interpretive pseudo-code consists of a series of address pointers. the exact format of the code is described in connection with the cdblk format. the purpose of this section is to give general insight into the interpretive approach involved.

the basic form of the code is related to reverse polish. in other words, the operands precede the operators which are zero address operators. there are some exceptions to these rules, notably the unary not operator and the selection construction which clearly require advance knowledge of the operator involved.

the operands are moved to the top of the main stack and the operators are applied to the top stack entries. like other versions of spitbol, this processor depends on knowing whether operands are required by name or by value and moves the appropriate object to the stack. thus no name/value checks are included in the operator circuits.

the actual pointers in the code point to a block whose first word is the address of the interpretor routine to be executed for the code word.

in the case of operators, the pointer is to a word which contains the address of the operator to be executed. in the case of operands such as constants, the pointer is to the operand itself. accordingly, all operands contain a field which points to the routine to load the value of the operand onto the stack. in the case of a variable, there are three such pointers. one to load the value, one to store the value and a third to jump to the label.

the handling of failure returns deserves special comment. the location flptr contains the pointer to the location on the main stack which contains the failure return which is in the form of a byte offset in the current code block (cdblk or exblk). when a failure occurs, the stack is popped as indicated by the setting of flptr and control is passed to the appropriate location in the current code block with the stack pointer pointing to the failure offset on the stack and flptr unchanged.

```
internal data representations
-----------------------------
representation of values
a value is represented by a pointer to a block which
describes the type and particulars of the data value.
in general, a variable is a location containing such a
pointer (although in the case of trace associations this
is modified, see description of trblk).
the following is a list of possible datatypes showing the
type of block used to hold the value. the details of
each block format are given later.
datatype             block type
--------             ----------
array                arblk or vcblk
code                 cdblk
expression           exblk or seblk
integer              icblk
name                 nmblk
pattern              p0blk or p1blk or p2blk
real                 rcblk
string               scblk
table                tbblk
program datatype     pdblk
```

```
representation of variables
---------------------------
during the course of evaluating expressions, it is
necessary to generate names of variables (for example
on the left side of a binary equals operator). these are
not to be confused with objects of datatype name which
are in fact values.
from a logical point of view, such names could be simply
represented by a pointer to the appropriate value cell.
however in the case of arrays and program defined
datatypes, this would violate the rule that there must be
no pointers into the middle of a block in dynamic store.
accordingly, a name is always represented by a base and
offset. the base points to the start of the block
containing the variable value and the offset is the
offset within this block in bytes. thus the address
of the actual variable is determined by adding the base
and offset values.
the following are the instances of variables represented
in this manner.
1)   natural variable base is ptr to vrblk
                      offset is *vrval
2)   table element    base is ptr to teblk
                      offset is *teval
3)   array element    base is ptr to arblk
                      offset is offset to element
4)   vector element   base is ptr to vcblk
                      offset is offset to element
5)   prog def dtp     base is ptr to pdblk
                      offset is offset to field value
in addition there are two cases of objects which are
like variables but cannot be handled in this manner.
these are called pseudo-variables and are represented
with a special base pointer as follows=
expression variable   ptr to evblk (see evblk)
keyword variable      ptr to kvblk (see kvblk)
pseudo-variables are handled as special cases by the
access procedure (acess) and the assignment procedure
(asign). see these two procedures for details.
```

```
organization of data area
-------------------------
the data area is divided into two regions.
static area
the static area builds up from the bottom and contains
data areas which are allocated dynamically but are never
deleted or moved around. the macro-program itself
uses the static area for the following.
1)   all variable blocks (vrblk).
2)   the hash table for variable blocks.
3)   miscellaneous buffers and work areas (see program
     initialization section).
in addition, the system procedures may use this area for
input/output buffers, external functions etc. space in
the static region is allocated by calling procedure alost
the following global variables define the current
location and size of the static area.
statb                address of start of static area
state                address+1 of last word in area.
the minimum size of static is given approximately by
     12 + *e$hnb + *e$sts + space for alphabet string
     and standard print buffer.
```

dynamic area
the dynamic area is built upwards in memory after the static region. data in this area must all be in standard block formats so that it can be processed by the garbage collector (procedure gbcol). gbcol compacts blocks down in this region as required by space exhaustion and can also move all blocks up to allow for expansion of the static region.
with the exception of tables and arrays, no spitbol object once built in dynamic memory is ever subsequently modified. observing this rule necessitates a copying action during string and pattern concatenation.
garbage collection is fundamental to the allocation of space for values. spitbol uses a very efficient garbage collector which insists that pointers into dynamic store should be identifiable without use of bit tables, marker bits etc. to satisfy this requirement, dynamic memory must not start at too low an address and lengths of arrays, tables, strings, code and expression blocks may not exceed the numerical value of the lowest dynamic address.
to avoid either penalizing users with modest requirements or restricting those with greater needs on host systems where dynamic memory is allocated in low addresses, the minimum dynamic address may be specified sufficiently high to permit arbitrarily large spitbol objects to be created (with the possibility in extreme cases of wasting large amounts of memory below the start address). this minimum value is made available in variable mxlen by a system routine, sysmx.
alternatively sysmx may indicate that a default may be used in which dynamic is placed at the lowest possible address following static.
the following global work cells define the location and length of the dynamic area.

```
dnamb                start of dynamic area
dnamp                next available location
dname                last available location + 1
```

dnamb is always higher than state since the alost procedure maintains some expansion space above state.
*** dnamb must never be permitted to have a value less than that in mxlen ***
space in the dynamic region is allocated by the alloc procedure. the dynamic region may be used by system procedures provided that all the rules are obeyed.
some of the rules are subtle so it is preferable for osint to manage its own memory needs. spitbol procs obey rules to ensure that no action can cause a garbage collection except at such times as contents of xl, xr and the stack are +clean+ (see comment before utility procedures and in gbcol for more detail). note that calls of alost may cause garbage collection (shift of memory to free space). spitbol procs which call

system routines assume that they cannot precipitate collection and this must be respected.

```
register usage
--------------
(cp)                    code pointer register. used to
                        hold a pointer to the current
                        location in the interpretive pseudo
                        code (i.e. ptr into a cdblk).
(xl,xr)                 general index registers. usually
                        used to hold pointers to blocks in
                        dynamic storage. an important
                        restriction is that the value in
                        xl must be collectable for
                        a garbage collect call. a value
                        is collectable if it either points
                        outside the dynamic area, or if it
                        points to the start of a block in
                        the dynamic area.
(xs)                    stack pointer. used to point to
                        the stack front. the stack may
                        build up or down and is used
                        to stack subroutine return points
                        and other recursively saved data.
(xt)                    an alternative name for xl during
                        its use in accessing stacked items.
(wa,wb,wc)              general work registers. cannot be
                        used for indexing, but may hold
                        various types of data.
(ia)                    used for all signed integer
                        arithmetic, both that used by the
                        translator and that arising from
                        use of snobol4 arithmetic operators
(ra)                    real accumulator. used for all
                        floating point arithmetic.
```

```
spitbol conditional assembly symbols
------------------------------------
in the spitbol translator, the following conditional
assembly symbols are referred to. to incorporate the
features referred to, the minimal source should be
prefaced by suitable conditional assembly symbol
definitions.
in all cases it is permissible to default the definitions
in which case the additional features will be omitted
from the target code.
.caex                define to allow up arrow for expon.
.caht                define to include horizontal tab
.casl                define to include 26 shifted lettrs
.cavt                define to include vertical tab
.cbyt                define for statistics in bytes
.ccmc                define to include syscm function
.ccmk                define to include compare keyword
.cepp                define if entrys have odd parity
.cera                define to include sysea function
.cexp                define if spitbol pops sysex args
.cgbc                define to include sysgc function
.cicc                define to ignore bad control cards
.cinc                define to add -include control card
.ciod                define to not use default delimiter
                       in processing 3rd arg of input()
                       and output()
.cmth                define to include math functions
.cnbf                define to omit buffer extension
.cnbt                define to omit batch initialisation
.cnci                define to enable sysci routine
.cncr                define to enable syscr routine
.cnex                define to omit exit() code.
.cnld                define to omit load() code.
.cnlf                define to add file type for load()
.cnpf                define to omit profile stuff
.cnra                define to omit all real arithmetic
.cnsc                define to no numeric-string compare
.cnsr                define to omit sort, rsort
.cpol                define if interface polling desired
.crel                define to include reloc routines
.crpp                define if returns have odd parity
.cs16                define to initialize stlim to 32767
.cs32                define to init stlim to 2147483647
                     omit to take default of 50000
.csax                define if sysax is to be called
.csed                define to use sediment in gbcol
.csfn                define to track source file names
.csln                define if line number in code block
.csn5                define to pad stmt nos to 5 chars
.csn6                define to pad stmt nos to 6 chars
.csn8                define to pad stmt nos to 8 chars
.csou                define if output, terminal to sysou
.ctet                define to table entry trace wanted
```

```
   .ctmd              define if systm unit is decisecond
   .cucf              define to include cfp$u
   .cuej              define to suppress needless ejects
   .culk              define to include &l/ucase keywords
   .culc              define to include &case (lc names)
                      if cucl defined, must support
                      minimal op flc wreg that folds
                      argument to upper case
   .cust              define to include set() code
                      conditional options
                      since .undef not allowed if symbol
                      not defined, a full comment line
                      indicates symbol initially not
                      defined.
.def   .ca                            define to allow up arrow for expon.
.def   .ca                            define to include horizontal tab
.def   .ca                            define to include 26 shifted lettrs
.def   .ca                            define to include vertical tab
   .cbyt              define for statistics in bytes
   .ccmc              define to include syscm function
   .ccmk              define to include compare keyword
   .cepp              define if entrys have odd parity
   .cera              define to include sysea function
   .cexp              define if spitbol pops sysex args
.def   .cg                            define to include sysgc function
   .cicc              define to ignore bad control cards
   .cinc              define to add -include control card
.def   .ci                            define to not use default delimiter
                      in processing 3rd arg of input()
                      and output()
   .cmth              define to include math functions
.def   .cn                            define to omit buffer extension
.def   .cn                            define to omit batch initialisation
   .cnci              define to enable sysci routine
   .cncr              define to enable syscr routine
   .cnex              define to omit exit() code.
.def   .cn                            define to omit load() code.
   .cnlf              define to add file type to load()
   .cnpf              define to omit profile stuff
   .cnra              define to omit all real arithmetic
   .cnsc              define if no numeric-string compare
   .cnsr              define to omit sort, rsort
   .cpol              define if interface polling desired
   .crel              define to include reloc routines
   .crpp              define if returns have odd parity
   .cs16              define to initialize stlim to 32767
   .cs32              define to init stlim to 2147483647
.def   .cs                            define if sysax is to be called
   .csed              define to use sediment in gbcol
   .csfn              define to track source file names
   .csln              define if line number in code block
   .csn5              define to pad stmt nos to 5 chars
   .csn6              define to pad stmt nos to 6 chars
```

```
.def    .cs                                 define to pad stmt nos to 8 chars
   .csou           define if output, terminal to sysou
.def    .ct                                 define to table entry trace wanted
   .ctmd           define if systm unit is decisecond
.def    .cu                                 define to include cfp$u
.def    .cu                                 define to suppress needless ejects
.def    .cu                                 define to include &l/ucase keywords
.def    .cu                                 define to include &case (lc names)
.def    .cu                                 define to include set() code
   force definition of .ccmk if .ccmc is defined
```

---

if **.ccmc**

```
.def    .cc
```

fi

```
this section starts with descriptions of the operating
system dependent procedures which are used by the spitbol
translator. all such procedures have five letter names
beginning with sys. they are listed in alphabetical
order.
all procedures have a  specification consisting of a
model call, preceded by a possibly empty list of register
contents giving parameters available to the procedure and
followed by a possibly empty list of register contents
required on return from the call or which may have had
their contents destroyed. only those registers explicitly
mentioned in the list after the call may have their
values changed.
the segment of code providing the external procedures is
conveniently referred to as osint (operating system
interface). the sysxx procedures it contains provide
facilities not usually available as primitives in
assembly languages. for particular target machines,
implementors may choose for some minimal opcodes which
do not have reasonably direct translations, to use calls
of additional procedures which they provide in osint.
e.g. mwb or trc might be translated as jsr sysmb,
jsr systc in some implementations.
in the descriptions, reference is made to --blk
formats (-- = a pair of letters). see the spitbol
definitions section for detailed descriptions of all
such block formats except fcblk for which sysfc should
be consulted.
section 0 contains inp,inr specifications of internal
procedures,routines. this gives a single pass translator
information making it easy to generate alternative calls
in the translation of jsr-s for procedures of different
types if this proves necessary.
```

sec		start of procedures section

---

if **.csax**

```
    sysax -- after execution
sysax   exp                                   define external entry point
    if the conditional assembly symbol .csax is defined,
    this routine is called immediately after execution and
    before printing of execution statistics or dump output.
    purpose of call is for implementor to determine and
    if the call is not required it will be omitted if .csax
    is undefined. in this case sysax need not be coded.
    jsr  sysax            call after execution
```

else

fi

if **.cbsp**

```
    sysbs -- backspace file
sysbs   exp                                    define external entry point
    sysbs is used to implement the snobol4 function backspace
    if the conditional assembly symbol .cbsp is defined.
    the meaning is system dependent.  in general, backspace
    repositions the file one record closer to the beginning
    of file, such that a subsequent read or write will
    operate on the previous record.
    (wa)                 ptr to fcblk or zero
    (xr)                 backspace argument (scblk ptr)
    jsr  sysbs           call to backspace
    ppm  loc             return here if file does not exist
    ppm  loc             return here if backspace not allowed
    ppm  loc             return here if i/o error
    (wa,wb)              destroyed
    the second error return is used for files for which
    backspace is not permitted. for example, it may be expected
    files on character devices are in this category.
```

fi

```
    sysbx -- before execution
sysbx   exp                                    define external entry point
    called after initial spitbol compilation and before
    commencing execution in case osint needs
    to assign files or perform other necessary services.
    osint may also choose to send a message to online
    terminal (if any) indicating that execution is starting.
    jsr  sysbx            call before execution starts
```

if **.cnci**

```
    sysci -- convert integer
sysci   exp
    sysci is an optional osint routine that causes spitbol to
    call sysci to convert integer values to strings, rather
    than using the internal spitbol conversion code.  this
    code may be less efficient on machines with hardware
    conversion instructions and in such cases, it may be an
    advantage to include sysci.  the symbol .cnci must be
    defined if this routine is to be used.
    the rules for converting integers to strings are that
    positive values are represented without any sign, and
    there are never any leading blanks or zeros, except in
    the case of zero itself which is represented as a single
    zero digit.  negative numbers are represented with a
    preceeding minus sign.  there are never any trailing
    blanks, and conversion cannot fail.
    (ia)                 value to be converted
    jsr  sysci           call to convert integer value
    (xl)                 pointer to pseudo-scblk with string
```

fi

---

if **.ccmc**

```
    syscm -- general string comparison function
syscm   exp                                     define external entry point
    provides string comparison determined by interface.
    used for international string comparison.
    (xr)                character pointer for first string
    (xl)                character pointer for second string
    (wb)                character count of first string
    (wa)                character count of second string
    jsr  syscm          call to syscm function
    ppm  loc            string too long for syscm
    ppm  loc            first string lexically gt second
    ppm  loc            first string lexically lt second
    ---                 strings equal
    (xl)                zero
    (xr)                destroyed
```

fi

---

if **.cnra**
else

---

if **.cncr**

```
    syscr -- convert real
syscr   exp
    syscr is an optional osint routine that causes spitbol to
    call syscr to convert real values to strings, rather
    than using the internal spitbol conversion code.  this
    code may be desired on machines where the integer size
    is too small to allow production of a sufficient number
    of significant digits.  the symbol .cncr must be defined
    if this routine is to be used.
    the rules for converting reals to strings are that
    positive values are represented without any sign, and
    there are never any leading blanks or zeros, except in
    the case of zero itself which is represented as a single
    zero digit.  negative numbers are represented with a
    preceeding minus sign.  there are never any trailing
    blanks, or trailing zeros in the fractional part.
    conversion cannot fail.
    (ra)                 value to be converted
    (wa)                 no. of significant digits desired
    (wb)                 conversion type:
                          negative for e-type conversion
                          zero for g-type conversion
                          positive for f-type conversion
    (wc)                 character positions in result scblk
    (xr)                 scblk for result
    jsr  syscr           call to convert real value
    (xr)                 result scblk
    (wa)                 number of result characters
```

fi
fi

```
    sysdc -- date check
sysdc   exp                                   define external entry point
    sysdc is called to check that the expiry date for a trial
    version of spitbol is unexpired.
    jsr  sysdc            call to check date
    return only if date is ok
```

```
    sysdm  -- dump core
sysdm   exp                                   define external entry point
    sysdm is called by a spitbol program call of dump(n) with
    n ge 4.  its purpose is to provide a core dump.
    n could hold an encoding of the start adrs for dump and
    amount to be dumped e.g.  n = 256*a + s , s = start adrs
    in kilowords,  a = kilowords to dump
    (xr)                  parameter n of call dump(n)
    jsr  sysdm            call to enter routine
```

```
    sysdt -- get current date
sysdt   exp                                     define external entry point
    sysdt is used to obtain the current date. the date is
    returned as a character string in any format appropriate
    to the operating system in use. it may also contain the
    current time of day. sysdt is used to implement the
    snobol4 function date().
    (xr)                  parameter n of call date(n)
    jsr  sysdt            call to get date
    (xl)                  pointer to block containing date
    the format of the block is like an scblk except that
    the first word need not be set. the result is copied
    into spitbol dynamic memory on return.
```

---

if **.cera**

```
    sysea -- inform osint of compilation and runtime errors
sysea   exp                                     define external entry point
    provides means for interface to take special actions on
    errors
    (wa)                  error code
    (wb)                  line number
    (wc)                  column number
    (xr)                  system stage
```

---

if **.csfn**

```
    (xl)                  file name (scblk)
```

fi

```
    jsr  sysea            call to sysea function
    ppm  loc              suppress printing of error message
    (xr)                  message to print (scblk) or 0
    sysea may not return if interface chooses to retain
    control.  closing files via the fcb chain will be the
    responsibility of the interface.
    all registers preserved
```

fi

```
    sysef -- eject file
sysef   exp                                   define external entry point
    sysef is used to write a page eject to a named file. it
    may only be used for files where this concept makes
    sense. note that sysef is not normally used for the
    standard output file (see sysep).
    (wa)                 ptr to fcblk or zero
    (xr)                 eject argument (scblk ptr)
    jsr  sysef           call to eject file
    ppm  loc             return here if file does not exist
    ppm  loc             return here if inappropriate file
    ppm  loc             return here if i/o error
```

```
    sysej -- end of job
sysej   exp                                        define external entry point
    sysej is called once at the end of execution to
    terminate the run. the significance of the abend and
    code values is system dependent. in general, the code
    value should be made available for testing, and the
    abend value should cause some post-mortem action such as
    a dump. note that sysej does not return to its caller.
    see sysxi for details of fcblk chain
    (wa)                 value of abend keyword
    (wb)                 value of code keyword
    (xl)                 o or ptr to head of fcblk chain
    jsr  sysej           call to end job
    the following special values are used as codes in (wb)
    999  execution suppressed
    998  standard output file full or unavailable in a sysxi
         load module. in these cases (wa) contains the number
         of the statement causing premature termination.
```

```
    sysem -- get error message text
sysem   exp                                 define external entry point
    sysem is used to obtain the text of err, erb calls in the
    source program given the error code number. it is allowed
    to return a null string if this facility is unavailable.
    (wa)                  error code number
    jsr  sysem            call to get text
    (xr)                  text of message
    the returned value is a pointer to a block in scblk
    format except that the first word need not be set. the
    string is copied into dynamic memory on return.
    if the null string is returned either because sysem does
    not provide error message texts or because wa is out of
    range, spitbol will print the string stored in errtext
    keyword.
```

```
    sysen -- endfile
sysen   exp                                       define external entry point
    sysen is used to implement the snobol4 function endfile.
    the meaning is system dependent. in general, endfile
    implies that no further i/o operations will be performed,
    but does not guarantee this to be the case. the file
    should be closed after the call, a subsequent read
    or write may reopen the file at the start or it may be
    necessary to reopen the file via sysio.
    (wa)                 ptr to fcblk or zero
    (xr)                 endfile argument (scblk ptr)
    jsr  sysen           call to endfile
    ppm  loc             return here if file does not exist
    ppm  loc             return here if endfile not allowed
    ppm  loc             return here if i/o error
    (wa,wb)              destroyed
    the second error return is used for files for which
    endfile is not permitted. for example, it may be expected
    that the standard input and output files are in this
    category.
```

```
    sysep -- eject printer page
sysep   exp                                     define external entry point
    sysep is called to perform a page eject on the standard
    printer output file (corresponding to syspr output).
    jsr  sysep            call to eject printer output
```

```
    sysex -- call external function
sysex   exp                                  define external entry point
    sysex is called to pass control to an external function
    previously loaded with a call to sysld.
    (xs)                  pointer to arguments on stack
    (xl)                  pointer to control block (efblk)
    (wa)                  number of arguments on stack
    jsr  sysex            call to pass control to function
    ppm  loc              return here if function call fails
    ppm  loc              return here if insufficient memory
    ppm  loc              return here if bad argument type
```

---

if **.cexp**
else

```
    (xs)                  popped past arguments
```

fi

```
    (xr)                  result returned
    the arguments are stored on the stack with
    the last argument at 0(xs). on return, xs
    is popped past the arguments.
    the form of the arguments as passed is that used in the
    spitbol translator (see definitions and data structures
    section). the control block format is also described
    (under efblk) in this section.
    there are two ways of returning a result.
    1)   return a pointer to a block in dynamic storage. this
         block must be in exactly correct format, including
         the first word. only functions written with intimate
         knowledge of the system will return in this way.
    2)   string, integer and real results may be returned by
         pointing to a pseudo-block outside dynamic memory.
         this block is in icblk, rcblk or scblk format except
         that the first word will be overwritten
         by a type word on return and so need not
         be correctly set. such a result is
         copied into main storage before proceeding.
         unconverted results may similarly be returned in a
         pseudo-block which is in correct format including
         type word recognisable by garbage collector since
         block is copied into dynamic memory.
```

```
    sysfc -- file control block routine
sysfc   exp                                         define external entry point
    see also sysio
    input and output have 3 arguments referred to as shown
         input(variable name,file arg1,file arg2)
         output(variable name,file arg1,file arg2)
    file arg1 may be an integer or string used to identify
    an i/o channel. it is converted to a string for checking.
    the exact significance of file arg2
    is not rigorously prescribed but to improve portability,
    the scheme described in the spitbol user manual
    should be adopted when possible. the preferred form is
    a string $f$,r$r$,c$c$,i$i$,...,z$z$  where
    $f$ is an optional file name which is placed first.
     remaining items may be omitted or included in any order.
    $r$ is maximum record length
    $c$ is a carriage control character or character string
    $i$ is some form of channel identification used in the
       absence of $f$ to associate the variable
       with a file allocated dynamically by jcl commands at
       spitbol load time.
    ,...,z$z$ are additional fields.
    if , (comma) cannot be used as a delimiter, .ciod
    should be defined to introduce by conditional assembly
    another delimiter (see
      iodel  equ  *
    early in definitions section).
    sysfc is called when a variable is input or output
    associated to check file arg1 and file arg2 and
    to  report whether an fcblk (file control
    block) is necessary and if so what size it should be.
    this makes it possible for spitbol rather than osint to
    allocate such a block in dynamic memory if required
    or alternatively in static memory.
    the significance of an fcblk , if one is requested, is
    entirely up to the system interface. the only restriction
    is that if the fcblk should appear to lie in dynamic
    memory, pointers to it should be proper pointers to
    the start of a recognisable and garbage collectable
    block (this condition will be met if sysfc requests
    spitbol to provide an fcblk).
    an option is provided for osint to return a pointer in
    xl to an fcblk which it privately allocated. this ptr
    will be made available when i/o occurs later.
    private fcblks may have arbitrary contents and spitbol
    stores nothing in them.
```

```
the requested size for an fcblk in dynamic memory
should allow a 2 word overhead for block type and
length fields. information subsequently stored in the
remaining words may be arbitrary if an xnblk (external
non-relocatable block) is requested. if the request is
for an xrblk (external relocatable block) the
contents of words should be collectable (i.e. any
apparent pointers into dynamic should be genuine block
pointers). these restrictions do not apply if an fcblk
is allocated outside dynamic or is not allocated at all.
if an fcblk is requested, its fields will be initialised
to zero before entry to sysio with the exception of
words 0 and 1 in which the block type and length
fields are placed for fcblks in dynamic memory only.
for the possible use of sysej and sysxi, if fcblks
are used, a chain is built so that they may all be
found - see sysxi for details.
if both file arg1 and file arg2 are null, calls of sysfc
and sysio are omitted.
if file arg1 is null (standard input/output file), sysfc
is called to check non-null file arg2 but any request
for an fcblk will be ignored, since spitbol handles the
standard files specially and cannot readily keep fcblk
pointers for them.
filearg1 is type checked by spitbol so further checking
may be unneccessary in many implementations.
file arg2 is passed so that sysfc may analyse and
check it. however to assist in this, spitbol also passes
on the stack the components of this argument with
file name, $f$ (otherwise null) extracted and stacked
first.
the other fields, if any, are extracted as substrings,
pointers to them are stacked and a count of all items
stacked is placed in wc. if an fcblk was earlier
allocated and pointed to via file arg1, sysfc is also
passed a pointer to this fcblk.
(xl)                  file arg1 scblk ptr (2nd arg)
(xr)                  filearg2 (3rd arg) or null
-(xs)...-(xs)         scblks for $f$,$r$,$c$,...
(wc)                  no. of stacked scblks above
(wa)                  existing file arg1 fcblk ptr or 0
(wb)                  0/3 for input/output assocn
jsr  sysfc            call to check need for fcblk
ppm  loc              invalid file argument
ppm  loc              fcblk already in use
(xs)                  popped (wc) times
(wa non zero)         byte size of requested fcblk
(wa=0,xl non zero)    private fcblk ptr in xl
(wa=xl=0)             no fcblk wanted, no private fcblk
(wc)                  0/1/2 request alloc of xrblk/xnblk
                      /static block for use as fcblk
(wb)                  destroyed
```

if **.cgbc**

```
    sysgc -- inform interface of garbage collections
sysgc   exp                                       define external entry point
    provides means for interface to take special actions
    prior to and after a garbage collection.
    possible usages-
    1. provide visible screen icon of garbage collection
       in progress
    2. inform virtual memory manager to ignore page access
       patterns during garbage collection.  such accesses
       typically destroy the page working set accumulated
       by the program.
    3. inform virtual memory manager that contents of memory
       freed by garbage collection can be discarded.
    (xr)                  non-zero if beginning gc
                          =0 if completing gc
    (wa)                  dnamb=start of dynamic area
    (wb)                  dnamp=next available location
    (wc)                  dname=last available location + 1
    jsr  sysgc            call to sysgc function
    all registers preserved
fi
```

```
    syshs -- give access to host computer features
syshs   exp                                   define external entry point
    provides means for implementing special features
    on different host computers. the only defined entry is
    that where all arguments are null in which case syshs
    returns an scblk containing name of computer,
    name of operating system and name of site separated by
    colons. the scblk need not have a correct first field
    as this is supplied on copying string to dynamic memory.
    spitbol does no argument checking but does provide a
    single error return for arguments checked as erroneous
    by osint. it also provides a single execution error
    return. if these are inadequate, use may be made of the
    minimal error section direct as described in minimal
    documentation, section 10.
    several non-error returns are provided. the first
    corresponds to the defined entry or, for implementation
    defined entries, any string may be returned. the others
    permit respectively,  return a null result, return with a
    result to be stacked which is pointed at by xr, and a
    return causing spitbol statement failure. if a returned
    result is in dynamic memory it must obey garbage
    collector rules. the only results copied on return
    are strings returned via ppm loc3 return.
    (wa)                  argument 1
    (xl)                  argument 2
    (xr)                  argument 3
    (wb)                  argument 4
    (wc)                  argument 5
    jsr  syshs            call to get host information
    ppm  loc1             erroneous arg
    ppm  loc2             execution error
    ppm  loc3             scblk ptr in xl or 0 if unavailable
    ppm  loc4             return a null result
    ppm  loc5             return result in xr
    ppm  loc6             cause statement failure
    ppm  loc7             return string at xl, length wa
    ppm  loc8             return copy of result in xr
```

```
    sysid -- return system identification
sysid   exp                                   define external entry point
    this routine should return strings to head the standard
    printer output. the first string will be appended to
    a heading line of the form
         macro spitbol version v.v
    supplied by spitbol itself. v.v are digits giving the
    major version number and generally at least a minor
    version number relating to osint should be supplied to
    give say
         macro spitbol version v.v(m.m)
    the second string should identify at least the machine
    and operating system.  preferably it should include
    the date and time of the run.
    optionally the strings may include site name of the
    the implementor and/or machine on which run takes place,
    unique site or copy number and other information as
    appropriate without making it so long as to be a
    nuisance to users.
    the first words of the scblks pointed at need not be
    correctly set.
    jsr  sysid            call for system identification
    (xr)                  scblk ptr for addition to header
    (xl)                  scblk ptr for second header
```

if **.cinc**

```
    sysif -- switch to new include file
sysif   exp                                 define external entry point
    sysif is used for include file processing, both to inform
    the interface when a new include file is desired, and
    when the end of file of an include file has been reached
    and it is desired to return to reading from the previous
    nested file.
    it is the responsibility of sysif to remember the file
    access path to the present input file before switching to
    the new include file.
    (xl)                  ptr to scblk or zero
    (xr)                  ptr to vacant scblk of length cswin
                          (xr not used if xl is zero)
    jsr  sysif            call to change files
    ppm  loc              unable to open file
    (xr)                  scblk with full path name of file
                          (xr not used if input xl is zero)
    register xl points to an scblk containing the name of the
    include file to which the interface should switch.  data
    is fetched from the file upon the next call to sysrd.
    sysif may have the ability to search multiple libraries
    for the include file named in (xl).  it is therefore
    required that the full path name of the file where the
    file was finally located be returned in (xr).  it is this
    name that is recorded along with the source statements,
    and will accompany subsequent error messages.
    register xl is zero to mark conclusion of use of an
    include file.
```

fi

```
    sysil -- get input record length
sysil   exp                                   define external entry point
    sysil is used to get the length of the next input record
    from a file previously input associated with a sysio
    call. the length returned is used to establish a buffer
    for a subsequent sysin call.  sysil also indicates to the
    caller if this is a binary or text file.
    (wa)                  ptr to fcblk or zero
    jsr  sysil            call to get record length
    (wa)                  length or zero if file closed
    (wc)                  zero if binary, non-zero if text
    no harm is done if the value returned is too long since
    unused space will be reclaimed after the sysin call.
    note that it is the sysil call (not the sysio call) which
    causes the file to be opened as required for the first
    record input from the file.
```

```
    sysin -- read input record
sysin   exp                                  define external entry point
    sysin is used to read a record from the file which was
    referenced in a prior call to sysil (i.e. these calls
    always occur in pairs). the buffer provided is an
    scblk for a string of length set from the sysil call.
    if the actual length read is less than this, the length
    field of the scblk must be modified before returning
    unless buffer is right padded with zeroes.
    it is also permissible to take any of the alternative
    returns after scblk length has been modified.
    (wa)                  ptr to fcblk or zero
    (xr)                  pointer to buffer (scblk ptr)
    jsr  sysin            call to read record
    ppm  loc              endfile or no i/p file after sysxi
    ppm  loc              return here if i/o error
    ppm  loc              return here if record format error
    (wa,wb,wc)            destroyed
```

```
    sysio -- input/output file association
sysio   exp                                          define external entry point
    see also sysfc.
    sysio is called in response to a snobol4 input or output
    function call except when file arg1 and file arg2
    are both null.
    its call always follows immediately after a call
    of sysfc. if sysfc requested allocation
    of an fcblk, its address will be in wa.
    for input files, non-zero values of $r$ should be
    copied to wc for use in allocating input buffers. if $r$
    is defaulted or not implemented, wc should be zeroised.
    once a file has been opened, subsequent input(),output()
    calls in which the second argument is identical with that
    in a previous call, merely associate the additional
    variable name (first argument) to the file and do not
    result in re-opening the file.
    in subsequent associated accesses to the file a pointer
    to any fcblk allocated will be made available.
    (xl)                  file arg1 scblk ptr (2nd arg)
    (xr)                  file arg2 scblk ptr (3rd arg)
    (wa)                  fcblk ptr (0 if none)
    (wb)                  0 for input, 3 for output
    jsr  sysio            call to associate file
    ppm  loc              return here if file does not exist
    ppm  loc              return if input/output not allowed
    (xl)                  fcblk pointer (0 if none)
    (wc)                  0 (for default) or max record lngth
    (wa,wb)               destroyed
    the second error return is used if the file named exists
    but input/output from the file is not allowed. for
    example, the standard output file may be in this category
    as regards input association.
```

```
    sysld -- load external function
sysld   exp                                   define external entry point
    sysld is called in response to the use of the snobol4
    load function. the named function is loaded (whatever
    this means), and a pointer is returned. the pointer will
    be used on subsequent calls to the function (see sysex).
    (xr)                  pointer to function name (scblk)
    (xl)                  pointer to library name (scblk)
    jsr  sysld            call to load function
    ppm  loc              return here if func does not exist
    ppm  loc              return here if i/o error
    ppm  loc              return here if insufficient memory
    (xr)                  pointer to loaded code
    the significance of the pointer returned is up to the
    system interface routine. the only restriction is that
    if the pointer is within dynamic storage, it must be
    a proper block pointer.
```

```
    sysmm -- get more memory
sysmm   exp                                    define external entry point
    sysmm is called in an attempt to allocate more dynamic
    memory. this memory must be allocated contiguously with
    the current dynamic data area.
    the amount allocated is up to the system to decide. any
    value is acceptable including zero if allocation is
    impossible.
    jsr  sysmm              call to get more memory
    (xr)                    number of additional words obtained
```

sysmx -- supply mxlen

sysmx **exp** define external entry point

```
because of the method of garbage collection, no spitbol
object is allowed to occupy more bytes of memory than
the integer giving the lowest address of dynamic
(garbage collectable) memory. mxlen is the name used to
refer to this maximum length of an object and for most
users of most implementations, provided dynamic memory
starts at an address of at least a few thousand words,
there is no problem.
if the default starting address is less than say 10000 or
20000, then a load time option should be provided where a
user can request that he be able to create larger
objects. this routine informs spitbol of this request if
any. the value returned is either an integer
representing the desired value of mxlen (and hence the
minimum dynamic store address which may result in
non-use of some store) or zero if a default is acceptable
in which mxlen is set to the lowest address allocated
to dynamic store before compilation starts.
if a non-zero value is returned, this is used for keyword
maxlngth. otherwise the initial low address of dynamic
memory is used for this keyword.
jsr  sysmx            call to get mxlen
(wa)                  either mxlen or 0 for default
```

```
    sysou -- output record
sysou   exp                                  define external entry point
    sysou is used to write a record to a file previously
    associated with a sysio call.
    (wa)                 ptr to fcblk
```

---

if **.csou**

```
                         or 0 for terminal or 1 for output
```

fi

---

if **.cnbf**

```
    (xr)                 record to be written (scblk)
```

else

```
    (xr)                 record to write (bcblk or scblk)
```

fi

```
    jsr  sysou           call to output record
    ppm  loc             file full or no file after sysxi
    ppm  loc             return here if i/o error
    (wa,wb,wc)           destroyed
    note that it is the sysou call (not the sysio call) which
    causes the file to be opened as required for the first
    record output to the file.
```

```
    syspi -- print on interactive channel
syspi   exp                                   define external entry point
    if spitbol is run from an online terminal, osint can
    request that messages such as copies of compilation
    errors be sent to the terminal (see syspp). if relevant
    reply was made by syspp then syspi is called to send such
    messages to the interactive channel.
    syspi is also used for sending output to the terminal
    through the special variable name, terminal.
    (xr)                  ptr to line buffer (scblk)
    (wa)                  line length
    jsr  syspi            call to print line
    ppm  loc              failure return
    (wa,wb)               destroyed
```

---

if **.cpol**

```
    syspl -- provide interactive control of spitbol
syspl   exp                                       define external entry point
    provides means for interface to take special actions,
    such as interrupting execution, breakpointing, stepping,
    and expression evaluation.  these last three options are
    not presently implemented by the code calling syspl.
    (wa)                  opcode as follows-
                          =0 poll to allow osint to interrupt
                          =1 breakpoint hit
                          =2 completion of statement stepping
                          =3 expression evaluation result
    (wb)                  statement number
    r$fcb                 o or ptr to head of fcblk chain
    jsr  syspl            call to syspl function
    ppm  loc              user interruption
    ppm  loc              step one statement
    ppm  loc              evaluate expression
    ---                   resume execution
                          (wa) = new polling interval
fi
```

```
    syspp -- obtain print parameters
syspp   exp                                    define external entry point
    syspp is called once during compilation to obtain
    parameters required for correct printed output format
    and to select other options. it may also be called again
    after sysxi when a load module is resumed. in this
    case the value returned in wa may be less than or equal
    to that returned in initial call but may not be
    greater.
    the information returned is -
    1.   line length in chars for standard print file
    2.   no of lines/page. 0 is preferable for a non-paged
         device (e.g. online terminal) in which case listing
         page throws are suppressed and page headers
         resulting from -title,-stitl lines are kept short.
    3.   an initial -nolist option to suppress listing unless
         the program contains an explicit -list.
    4.   options to suppress listing of compilation and/or
         execution stats (useful for established programs) -
         combined with 3. gives possibility of listing
         file never being opened.
    5.   option to have copies of errors sent to an
         interactive channel in addition to standard printer.
    6.   option to keep page headers short (e.g. if listing
         to an online terminal).
    7.   an option to choose extended or compact listing
         format. in the former a page eject and in the latter
         a few line feeds precede the printing of each
         of-- listing, compilation statistics, execution
         output and execution statistics.
    8.   an option to suppress execution as though a
         -noexecute card were supplied.
    9.   an option to request that name /terminal/  be pre-
         associated to an online terminal via syspi and sysri
    10.  an intermediate (standard) listing option requiring
         that page ejects occur in source listings. redundant
         if extended option chosen but partially extends
         compact option.
    11.  option to suppress sysid identification.
    jsr  syspp            call to get print parameters
    (wa)                  print line length in chars
    (wb)                  number of lines/page
    (wc)                  bits value ...mlkjihgfedcba where
                          a = 1 to send error copy to int.ch.
                          b = 1 means std printer is int. ch.
                          c = 1 for -nolist option
                          d = 1 to suppress compiln. stats
                          e = 1 to suppress execn. stats
                          f = 1/0 for extnded/compact listing
                          g = 1 for -noexecute
                          h = 1 pre-associate /terminal/
                          i = 1 for standard listing option.
                          j = 1 suppresses listing header
```

```
k = 1 for -print
l = 1 for -noerrors
```

---

if **.culc**

```
m = 1 for -case 1
```

fi

```
    syspr -- print line on standard output file
syspr   exp                                      define external entry point
    syspr is used to print a single line on the standard
    output file.
    (xr)                  pointer to line buffer (scblk)
    (wa)                  line length
    jsr  syspr            call to print line
    ppm  loc              too much o/p or no file after sysxi
    (wa,wb)               destroyed
    the buffer pointed to is the length obtained from the
    syspp call and is filled out with trailing blanks. the
    value in wa is the actual line length which may be less
    than the maximum line length possible. there is no space
    control associated with the line, all lines are printed
    single spaced. note that null lines (wa=0) are possible
    in which case a blank line is to be printed.
    the error exit is used for systems which limit the amount
    of printed output. if possible, printing should be
    permitted after this condition has been signalled once to
    allow for dump and other diagnostic information.
    assuming this to be possible, spitbol may make more syspr
    calls. if the error return occurs another time, execution
    is terminated by a call of sysej with ending code 998.
```

```
    sysrd -- read record from standard input file
sysrd  exp                                  define external entry point
    sysrd is used to read a record from the standard input
    file. the buffer provided is an scblk for a string the
    length of which in characters is given in wc, this
    corresponding to the maximum length of string which
    spitbol is prepared to receive. at compile time it
    corresponds to xxx in the most recent -inxxx card
    (default 72) and at execution time to the most recent
    ,r$r$ (record length) in the third arg of an input()
    statement for the standard input file (default 80).
    if fewer than (wc) characters are read, the length
    field of the scblk must be adjusted before returning
    unless the buffer is right padded with zeroes.
    it is also permissible to take the alternative return
    after such an adjustment has been made.
    spitbol may continue to make calls after an endfile
    return so this routine should be prepared to make
    repeated endfile returns.
    (xr)                  pointer to buffer (scblk ptr)
    (wc)                  length of buffer in characters
    jsr  sysrd            call to read line
    ppm  loc              endfile or no i/p file after sysxi
```

---

if **.csfn**

```
                          or input file name change.  if
                          the former, scblk length is zero.
                          if input file name change, length
                          is non-zero. caller should re-issue
                          sysrd to obtain input record.
```

fi

```
    (wa,wb,wc)            destroyed
```

```
    sysri -- read record from interactive channel
sysri   exp                                   define external entry point
    reads a record from online terminal for spitbol variable,
    terminal. if online terminal is unavailable then code the
    endfile return only.
    the buffer provided is of length 258 characters. sysri
    should replace the count in the second word of the scblk
    by the actual character count unless buffer is right
    padded with zeroes.
    it is also permissible to take the alternative
    return after adjusting the count.
    the end of file return may be used if this makes
    sense on the target machine (e.g. if there is an
    eof character.)
    (xr)                  ptr to 258 char buffer (scblk ptr)
    jsr  sysri            call to read line from terminal
    ppm  loc              end of file return
    (wa,wb,wc)            may be destroyed
```

```
    sysrw -- rewind file
sysrw   exp                                 define external entry point
    sysrw is used to rewind a file i.e. reposition the file
    at the start before the first record. the file should be
    closed and the next read or write call will open the
    file at the start.
    (wa)                  ptr to fcblk or zero
    (xr)                  rewind arg (scblk ptr)
    jsr  sysrw            call to rewind file
    ppm  loc              return here if file does not exist
    ppm  loc              return here if rewind not allowed
    ppm  loc              return here if i/o error
```

if **.cust**

```
    sysst -- set file pointer
sysst   exp                                  define external entry point
    sysst is called to change the position of a file
    pointer. this is accomplished in a system dependent
    manner, and thus the 2nd and 3rd arguments are passed
    unconverted.
    (wa)                 fcblk pointer
    (wb)                 2nd argument
    (wc)                 3rd argument
    jsr  sysst           call to set file pointer
    ppm  loc             return here if invalid 2nd arg
    ppm  loc             return here if invalid 3rd arg
    ppm  loc             return here if file does not exist
    ppm  loc             return here if set not allowed
    ppm  loc             return here if i/o error
```

fi

```
    systm -- get execution time so far
systm   exp                                   define external entry point
    systm is used to obtain the amount of execution time
    used so far since spitbol was given control. the units
    are described as milliseconds in the spitbol output, but
    the exact meaning is system dependent. where appropriate,
    this value should relate to processor rather than clock
    timing values.
    if the symbol .ctmd is defined, the units are described
    as deciseconds (0.1 second).
    jsr  systm            call to get timer value
    (ia)                  time so far in milliseconds
                          (deciseconds if .ctmd defined)
```

```
    systt -- trace toggle
systt   exp                              define external entry point
    called by spitbol function trace() with no args to
    toggle the system trace switch.  this permits tracing of
    labels in spitbol code to be turned on or off.
    jsr  systt            call to toggle trace switch
```

```
    sysul -- unload external function
sysul   exp
```
define external entry point

```
    sysul is used to unload a function previously
    loaded with a call to sysld.
    (xr)                ptr to control block (efblk)
    jsr  sysul          call to unload function
    the function cannot be called following a sysul call
    until another sysld call is made for the same function.
    the efblk contains the function code pointer and also a
    pointer to the vrblk containing the function name (see
    definitions and data structures section).
```

---

if **.cnex**
else

```
    sysxi -- exit to produce load module
sysxi   exp                                    define external entry point
    when sysxi is called, xl contains either a string pointer
    or zero. in the former case, the string gives the
    character name of a program. the intention is that
    spitbol execution should be terminated forthwith and
    the named program loaded and executed. this type of chain
    execution is very system dependent and implementors may
    choose to omit it or find it impossible to provide.
    if (xl) is zero,ia contains one of the following integers
    -1, -2, -3, -4
         create if possible a load module containing only the
         impure area of memory which needs to be loaded with
         a compatible pure segment for subsequent executions.
         version numbers to check compatibility should be
         kept in both segments and checked on loading.
         to assist with this check, (xr) on entry is a
         pointer to an scblk containing the spitbol major
         version number v.v (see sysid).  the file thus
         created is called a save file.
    0    if possible, return control to job control
         command level. the effect if available will be
         system dependent.
    +1, +2, +3, +4
         create if possible a load module from all of
         memory. it should be possible to load and execute
         this module directly.
    in the case of saved load modules, the status of open
    files is not preserved and implementors may choose to
    offer means of attaching files before execution of load
    modules starts or leave it to the user to include
    suitable input(), output() calls in his program.
    sysxi should make a note that no i/o channels,
    including standard files, have files attached so that
    calls of sysin, sysou, syspr, sysrd should fail unless
    new associations are made for the load module.
    at least in the case of the standard output file, it is
    recommended that either the user be required to attach
    a file or that a default file is attached, since the
    problem of error messages generated by the load module
    is otherwise severe. as a last resort, if spitbol
    attempts to write to the standard output file and gets a
    reply indicating that such ouput is unacceptable it stops
    by using an entry to sysej with ending code 998.
    as described below, passing of some arguments makes it
    clear that load module will use a standard output file.
    if use is made of fcblks for i/o association, spitbol
    builds a chain so that those in use may be found in sysxi
    and sysej. the nodes are 4 words long. third word
    contains link to next node or 0, fourth word contains
    fcblk pointer.
```

```
sysxi (continued)
(xl)                  zero or scblk ptr to first argument
(xr)                  ptr to v.v scblk
(ia)                  signed integer argument
(wa)                  scblk ptr to second argument
(wb)                  0 or ptr to head of fcblk chain
jsr  sysxi            call to exit
ppm  loc              requested action not possible
ppm  loc              action caused irrecoverable error
(wb,wc,ia,xr,xl,cp)   should be preserved over call
(wa)                  0 in all cases except sucessful
                      performance of exit(4) or exit(-4),
                      in which case 1 should be returned.
loading and running the load module or returning from
jcl command level causes execution to resume at the point
after the error returns which follow the call of sysxi.
the value passed as exit argument is used to indicate
options required on resumption of load module.
+1 or -1 require that on resumption, sysid and syspp be
called and a heading printed on the standard output file.
+2 or -2 indicate that syspp will be called but not sysid
and no heading will be put on standard output file.
above options have the obvious implication that a
standard o/p file must be provided for the load module.
+3, +4, -3 or -4 indicate calls of neither sysid nor
syspp and no heading will be placed on standard output
file.
+4 or -4 indicate that execution is to continue after
creation of the save file or load module, although all
files will be closed by the sysxi action.  this permits
the user to checkpoint long-running programs while
continuing execution.
no return from sysxi is possible if another program
is loaded and entered.
```

fi

    introduce the internal procedures.

`acess` **inp**
`acomp` **inp**
`alloc` **inp**

---

if **.cnbf**
else
`alobf` **inp**
fi
`alocs` **inp**
`alost` **inp**

---

if **.cnbf**
else
`apndb` **inp**
fi

---

if **.cnra**
`arith` **inp**
else
`arith` **inp**
fi
`asign` **inp**
`asinp` **inp**
`blkln` **inp**
`cdgcg` **inp**
`cdgex` **inp**
`cdgnm` **inp**
`cdgvl` **inp**
`cdwrd` **inp**
`cmgen` **inp**
`cmpil` **inp**
`cncrd` **inp**
`copyb` **inp**
`dffnc` **inp**
`dtach` **inp**
`dtype` **inp**
`dumpr` **inp**

---

if **.ceng**
`enevs` **inp**
`engts` **inp**
fi
`ermsg` **inp**
`ertex` **inp**
`evali` **inp**
`evalp` **inp**
`evals` **inp**
`evalx` **inp**
`exbld` **inp**
`expan` **inp**
`expap` **inp**
`expdm` **inp**
`expop` **inp**

---

if **.csfn**

`filnm` **inp**
fi

---

if **.culc**
`flstg` **inp**
fi
`gbcol` **inp**
`gbcpf` **inp**
`gtarr` **inp**

`gtcod` **inp**
`gtexp` **inp**
`gtint` **inp**
`gtnum` **inp**
`gtnvr` **inp**
`gtpat` **inp**

---

if **.cnra**
else
`gtrea` **inp**
fi
`gtsmi` **inp**

---

if **.cnbf**
else
`gtstb` **inp**
fi
`gtstg` **inp**
`gtvar` **inp**
`hashs` **inp**
`icbld` **inp**
`ident` **inp**
`inout` **inp**

---

if **.cnbf**
else
`insbf` **inp**
fi
`insta` **inp**
`iofcb` **inp**
`ioppf` **inp**
`ioput` **inp**
`ktrex` **inp**
`kwnam` **inp**
`lcomp` **inp**
`listr` **inp**
`listt` **inp**

---

if **.csfn**
`newfn` **inp**
fi
`nexts` **inp**
`patin` **inp**
`patst` **inp**
`pbild` **inp**
`pconc` **inp**
`pcopy` **inp**

---

if **.cnpf**
else
`prflr` **inp**
`prflu` **inp**
fi
`prpar` **inp**
`prtch` **inp**
`prtic` **inp**

`prtis`	**inp**
`prtin`	**inp**
`prtmi`	**inp**
`prtmm`	**inp**
`prtmx`	**inp**
`prtnl`	**inp**
`prtnm`	**inp**
`prtnv`	**inp**
`prtpg`	**inp**
`prtps`	**inp**
`prtsn`	**inp**
`prtst`	**inp**

```
prttr   inp
prtvl   inp
prtvn   inp
```

---

if **.cnra**
else
`rcbld   inp`
fi
`readr   inp`

---

if **.crel**

```
relaj   inp
relcr   inp
reldn   inp
reloc   inp
relst   inp
relws   inp
```

fi
`rstrt   inp`

---

if **.c370**
`sbool   inp`
fi

```
sbstr   inp
scane   inp
scngf   inp
setvr   inp
```

---

if **.cnsr**
else

```
sorta   inp
sortc   inp
sortf   inp
sorth   inp
```

fi

```
start   inp
stgcc   inp
tfind   inp
tmake   inp
trace   inp
trbld   inp
trimr   inp
trxeq   inp
vmake   inp
xscan   inp
xscni   inp
    introduce the internal routines
arref   inr
cfunc   inr
exfal   inr
exint   inr
exits   inr
exixr   inr
exnam   inr
exnul   inr
```

---

if **.cnra**
else
`exrea` **inr**
fi
`exsid` **inr**
`exvnm` **inr**
`failp` **inr**
`flpop` **inr**
`indir` **inr**
`match` **inr**
`retrn` **inr**
`stcov` **inr**
`stmgo` **inr**
`stopr` **inr**
`succp` **inr**
`sysab` **inr**
`systu` **inr**

```
    this section contains all symbol definitions and also
    pictures of all data structures used in the system.
       sec                                       start of definitions section
    definitions of machine parameters
    the minimal translator should supply appropriate values
    for the particular target machine for all the
    equ  *
    definitions given at the start of this section.
    note that even if conditional assembly is used to omit
    some feature (e.g. real arithmetic) a full set of cfp$-
    values must be supplied. use dummy values if genuine
    ones are not needed.
cfp$a   equ   *                                  number of characters in alphabet
cfp$b   equ   *                                  bytes/word addressing factor
cfp$c   equ   *                                  number of characters per word
cfp$f   equ   *                                  offset in bytes to chars in
                      scblk. see scblk format.
cfp$i   equ   *                                  number of words in integer constant
cfp$m   equ   *                                  max positive integer in one word
cfp$n   equ   *                                  number of bits in one word
    the following definitions require the supply of either
    a single parameter if real arithmetic is omitted or
    three parameters if real arithmetic is included.
```

---

if **.cnra**

```
nstmx   equ   *                                  no. of decimal digits in cfp$m
```

else

```
cfp$r   equ   *                                  number of words in real constant
cfp$s   equ   *                                  number of sig digs for real output
cfp$x   equ   *                                  max digits in real exponent
```

---

if **.cncr**

```
nstmx   equ   *                                  no. of decimal digits in cfp$m
mxdgs   equ   cfp$s+cfp$x                        max digits in real number
    max space for real (for +0.e+) needs five more places
nstmr   equ   mxdgs+5                            max space for real
```

else

```
mxdgs   equ   cfp$s+cfp$x                        max digits in real number
    max space for real (for +0.e+) needs five more places
nstmx   equ   mxdgs+5                            max space for real
```

fi

fi

---

if **.cucf**

```
    the following definition for cfp$u supplies a realistic
    upper bound on the size of the alphabet.  cfp$u is used
    to save space in the scane bsw-iff-esw table and to ease
    translation storage requirements.
cfp$u   equ   *                                  realistic upper bound on alphabet
```

fi

```
    environment parameters
    the spitbol program is essentially independent of
    the definitions of these parameters. however, the
    efficiency of the system may be affected. consequently,
    these parameters may require tuning for a given version
    the values given in comments have been successfully used.
    e$srs is the number of words to reserve at the end of
    storage for end of run processing. it should be
    set as small as possible without causing memory overflow
    in critical situations (e.g. memory overflow termination)
    and should thus reserve sufficient space at least for
    an scblk containing say 30 characters.
e$srs   equ   *                                  30 words
    e$sts is the number of words grabbed in a chunk when
    storage is allocated in the static region. the minimum
    permitted value is 256/cfp$b. larger values will lead
    to increased efficiency at the cost of wasting memory.
e$sts   equ   *                                  500 words
    e$cbs is the size of code block allocated initially and
    the expansion increment if overflow occurs. if this value
    is too small or too large, excessive garbage collections
    will occur during compilation and memory may be lost
    in the case of a too large value.
e$cbs   equ   *                                  500 words
    e$hnb is the number of bucket headers in the variable
    hash table. it should always be odd. larger values will
    speed up compilation and indirect references at the
    expense of additional storage for the hash table itself.
e$hnb   equ   *                                  127 bucket headers
    e$hnw is the maximum number of words of a string
    name which participate in the string hash algorithm.
    larger values give a better hash at the expense of taking
    longer to compute the hash. there is some optimal value.
e$hnw   equ   *                                  6 words
    e$fsp.  if the amount of free space left after a garbage
    collection is small compared to the total amount of space
    in use garbage collector thrashing is likely to occur as
    this space is used up.  e$fsp is a measure of the
    minimum percentage of dynamic memory left as free space
    before the system routine sysmm is called to try to
    obtain more memory.
e$fsp   equ   *                                  15 percent
```

---

if **.csed**

```
    e$sed.  if the amount of free space left in the sediment
    after a garbage collection is a significant fraction of
    the new sediment size, the sediment is marked for
    collection on the next call to the garbage collector.
e$sed   equ   *                                  25 percent
```

fi

```
    definitions of codes for letters
ch$la   equ   *                                        letter a
ch$lb   equ   *                                        letter b
ch$lc   equ   *                                        letter c
ch$ld   equ   *                                        letter d
ch$le   equ   *                                        letter e
ch$lf   equ   *                                        letter f
ch$lg   equ   *                                        letter g
ch$lh   equ   *                                        letter h
ch$li   equ   *                                        letter i
ch$lj   equ   *                                        letter j
ch$lk   equ   *                                        letter k
ch$ll   equ   *                                        letter l
ch$lm   equ   *                                        letter m
ch$ln   equ   *                                        letter n
ch$lo   equ   *                                        letter o
ch$lp   equ   *                                        letter p
ch$lq   equ   *                                        letter q
ch$lr   equ   *                                        letter r
ch$ls   equ   *                                        letter s
ch$lt   equ   *                                        letter t
ch$lu   equ   *                                        letter u
ch$lv   equ   *                                        letter v
ch$lw   equ   *                                        letter w
ch$lx   equ   *                                        letter x
ch$ly   equ   *                                        letter y
ch$l$   equ   *                                        letter z
    definitions of codes for digits
ch$d0   equ   *                                        digit 0
ch$d1   equ   *                                        digit 1
ch$d2   equ   *                                        digit 2
ch$d3   equ   *                                        digit 3
ch$d4   equ   *                                        digit 4
ch$d5   equ   *                                        digit 5
ch$d6   equ   *                                        digit 6
ch$d7   equ   *                                        digit 7
ch$d8   equ   *                                        digit 8
ch$d9   equ   *                                        digit 9
```

```
    definitions of codes for special characters
    the names of these characters are related to their
    original representation in the ebcdic set corresponding
    to the description in standard snobol4 manuals and texts.
ch$am   equu  *                 keyword operator (ampersand)
ch$as   equ  *                  multiplication symbol (asterisk)
ch$at   equ  *                  cursor position operator (at)
ch$bb   equ  *                  left array bracket (less than)
ch$bl   equ  *                  blank
ch$br   equ  *                  alternation operator (vertical bar)
ch$cl   equ  *                  goto symbol (colon)
ch$cm   equ  *                  comma
ch$dl   equ  *                  indirection operator (dollar)
ch$dt   equ  *                  name operator (dot)
ch$dq   equ  *                  double quote
ch$eq   equ  *                  equal sign
ch$ex   equ  *                  exponentiation operator (exclm)
ch$mn   equ  *                  minus sign / hyphen
ch$nm   equ  *                  number sign
ch$nt   equ  *                  negation operator (not)
ch$pc   equ  *                  percent
ch$pl   equ  *                  plus sign
ch$pp   equ  *                  left parenthesis
ch$rb   equ  *                  right array bracket (grtr than)
ch$rp   equ  *                  right parenthesis
ch$qu   equ  *                  interrogation operator (question)
ch$sl   equ  *                  slash
ch$sm   equ  *                  semicolon
ch$sq   equ  *                  single quote
ch$un   equ  *                  special identifier char (underline)
ch$ob   equ  *                  opening bracket
ch$cb   equ  *                  closing bracket
```

```
    remaining chars are optional additions to the standards.
```

---

if **.caht**

```
    tab characters - syntactically equivalent to blank
ch$ht   equ   *                                   horizontal tab
```

fi

---

if **.cavt**

```
ch$vt   equ   *                                   vertical tab
```

fi

---

if **.caex**

```
    up arrow same as exclamation mark for exponentiation
ch$ey   equ   *                                   up arrow
```

fi

---

if **.casl**

```
    lower case or shifted case alphabetic chars
ch$$a   equ   *                                   shifted a
ch$$b   equ   *                                   shifted b
ch$$c   equ   *                                   shifted c
ch$$d   equ   *                                   shifted d
ch$$e   equ   *                                   shifted e
ch$$f   equ   *                                   shifted f
ch$$g   equ   *                                   shifted g
ch$$h   equ   *                                   shifted h
ch$$i   equ   *                                   shifted i
ch$$j   equ   *                                   shifted j
ch$$k   equ   *                                   shifted k
ch$$l   equ   *                                   shifted l
ch$$m   equ   *                                   shifted m
ch$$n   equ   *                                   shifted n
ch$$o   equ   *                                   shifted o
ch$$p   equ   *                                   shifted p
ch$$q   equ   *                                   shifted q
ch$$r   equ   *                                   shifted r
ch$$s   equ   *                                   shifted s
ch$$t   equ   *                                   shifted t
ch$$u   equ   *                                   shifted u
ch$$v   equ   *                                   shifted v
ch$$w   equ   *                                   shifted w
ch$$x   equ   *                                   shifted x
ch$$y   equ   *                                   shifted y
ch$$$   equ   *                                   shifted z
```

fi

```
    if a delimiter other than ch$cm must be used in
    the third argument of input(),output() then .ciod should
    be defined and a parameter supplied for iodel.
```

---

if **.ciod**

```
iodel   equ   *
```

else

```
iodel   equ   *
```

fi

data block formats and definitions

the following sections describe the detailed format of all possible data blocks in static and dynamic memory. every block has a name of the form xxblk where xx is a unique two character identifier. the first word of every block must contain a pointer to a program location in the interpretor which is immediately preceded by an address constant containing the value bl$xx where xx is the block identifier. this provides a uniform mechanism for distinguishing between the various block types.

in some cases, the contents of the first word is constant for a given block type and merely serves as a pointer to the identifying address constant. however, in other cases there are several possibilities for the first word in which case each of the several program entry points must be preceded by the appropriate constant.

in each block, some of the fields are relocatable. this means that they may contain a pointer to another block in the dynamic area. (to be more precise, if they contain a pointer within the dynamic area, then it is a pointer to a block). such fields must be modified by the garbage collector (procedure gbcol) whenever blocks are compacted in the dynamic region. the garbage collector (actually procedure gbcpf) requires that all such relocatable fields in a block must be contiguous.

```
the description format uses the following scheme.
1)   block title and two character identifier
2)   description of basic use of block and indication
     of circumstances under which it is constructed.
3)   picture of the block format. in these pictures low
     memory addresses are at the top of the page. fixed
     length fields are surrounded by i (letter i). fields
     which are fixed length but whose length is dependent
     on a configuration parameter are surrounded by *
     (asterisk). variable length fields are surrounded
     by / (slash).
4)   definition of symbolic offsets to fields in
     block and of the size of the block if fixed length
     or of the size of the fixed length fields if the
     block is variable length.
     note that some routines such as gbcpf assume
     certain offsets are equal. the definitions
     given here enforce this.  make changes to
     them only with due care.
definitions of common offsets
```

```
offs1   equ   *
offs2   equ   *
offs3   equ   *
```

```
5)   detailed comments on the significance and formats
     of the various fields.
the order is alphabetical by identification code.
```

```
    definitions of block codes
    this table provides a unique identification code for
    each separate block type. the first word of a block in
    the dynamic area always contains the address of a program
    entry point. the block code is used as the entry point id
    the order of these codes dictates the order of the table
    used by the datatype function (scnmt in the constant sec)
    block codes for accessible datatypes
    note that real and buffer types are always included, even
    if they are conditionally excluded elsewhere.  this main-
    tains block type codes across all versions of spitbol,
    providing consistancy for external functions.  but note
    that the bcblk is out of alphabetic order, placed at the
    end of the list so as not to change the block type
    ordering in use in existing external functions.
bl$ar   equ   0                                       arblk array
bl$cd   equ   bl$ar+1                                 cdblk code
bl$ex   equ   bl$cd+1                                 exblk expression
bl$ic   equ   bl$ex+1                                 icblk integer
bl$nm   equ   bl$ic+1                                 nmblk name
bl$p0   equ   bl$nm+1                                 p0blk pattern
bl$p1   equ   bl$p0+1                                 p1blk pattern
bl$p2   equ   bl$p1+1                                 p2blk pattern
bl$rc   equ   bl$p2+1                                 rcblk real
bl$sc   equ   bl$rc+1                                 scblk string
bl$se   equ   bl$sc+1                                 seblk expression
bl$tb   equ   bl$se+1                                 tbblk table
bl$vc   equ   bl$tb+1                                 vcblk array
bl$xn   equ   bl$vc+1                                 xnblk external
bl$xr   equ   bl$xn+1                                 xrblk external
bl$bc   equ   bl$xr+1                                 bcblk buffer
bl$pd   equ   bl$bc+1                                 pdblk program defined datatype
bl$$d   equ   bl$pd+1                                 number of block codes for data
    other block codes
bl$tr   equ   bl$pd+1                                 trblk
bl$bf   equ   bl$tr+1                                 bfblk
bl$cc   equ   bl$bf+1                                 ccblk
bl$cm   equ   bl$cc+1                                 cmblk
bl$ct   equ   bl$cm+1                                 ctblk
bl$df   equ   bl$ct+1                                 dfblk
bl$ef   equ   bl$df+1                                 efblk
bl$ev   equ   bl$ef+1                                 evblk
bl$ff   equ   bl$ev+1                                 ffblk
bl$kv   equ   bl$ff+1                                 kvblk
bl$pf   equ   bl$kv+1                                 pfblk
bl$te   equ   bl$pf+1                                 teblk
bl$$i   equ   0                                       default identification code
bl$$t   equ   bl$tr+1                                 code for data or trace block
bl$$$   equ   bl$te+1                                 number of block codes
```

field references
references to the fields of data blocks are symbolic
(i.e. use the symbolic offsets) with the following
exceptions.

1) references to the first word are usually not
   symbolic since they use the (x) operand format.
2) the code which constructs a block is often not
   symbolic and should be changed if the corresponding
   block format is modified.
3) the plc and psc instructions imply an offset
   corresponding to the definition of cfp$f.
4) there are non-symbolic references (easily changed)
   in the garbage collector (procedures gbcpf, blkln).
5) the fields idval, fargs appear in several blocks
   and any changes must be made in parallel to all
   blocks containing the fields. the actual references
   to these fields are symbolic with the above
   listed exceptions.
6) several spots in the code assume that the
   definitions of the fields vrval, teval, trnxt are
   the same (these are sections of code which search
   out along a trblk chain from a variable).
7) references to the fields of an array block in the
   array reference routine arref are non-symbolic.

apart from the exceptions listed, references are symbolic
as far as possible and modifying the order or number
of fields will not require changes.

```
common fields for function blocks
blocks which represent callable functions have two
common fields at the start of the block as follows.
     +------------------------------------+
     i                fcode               i
     +------------------------------------+
     i                fargs               i
     +------------------------------------+
     /                                    /
     /       rest of function block       /
     /                                    /
     +------------------------------------+
```

fcode **equ** 0 pointer to code for function
fargs **equ** 1 number of arguments

```
fcode is a pointer to the location in the interpretor
program which processes this type of function call.
fargs is the expected number of arguments. the actual
number of arguments is adjusted to this amount by
deleting extra arguments or supplying trailing nulls
for missing ones before transferring though fcode.
a value of 999 may be used in this field to indicate a
variable number of arguments (see svblk field svnar).
the block types which follow this scheme are.
ffblk                field function
dfblk                datatype function
pfblk                program defined function
efblk                external loaded function
```

```
    identification field
    id   field
    certain program accessible objects (those which contain
    other data values and can be copied) are given a unique
    identification number (see exsid). this id value is an
    address integer value which is always stored in word two.
idval   equ  1                                  id value field
    the blocks containing an idval field are.
    arblk                 array
```

---

if **.cnbf**

else

```
    bcblk                 buffer control block
```

fi

```
    pdblk                 program defined datatype
    tbblk                 table
    vcblk                 vector block (array)
    note that a zero idval means that the block is only
    half built and should not be dumped (see dumpr).
```

```
array block (arblk)
an array block represents an array value other than one
with one dimension whose lower bound is one (see vcblk).
an arblk is built with a call to the functions convert
(s$cnv) or array (s$arr).
     +-----------------------------------+
     i               artyp               i
     +-----------------------------------+
     i               idval               i
     +-----------------------------------+
     i               arlen               i
     +-----------------------------------+
     i               arofs               i
     +-----------------------------------+
     i               arndm               i
     +-----------------------------------+
     *               arlbd               *
     +-----------------------------------+
     *               ardim               *
     +-----------------------------------+
     *                                   *
     * above 2 flds repeated for each dim *
     *                                   *
     +-----------------------------------+
     i               arpro               i
     +-----------------------------------+
     /                                   /
     /               arvls               /
     /                                   /
     +-----------------------------------+
```

```
    array block (continued)
artyp   equ   0                 pointer to dummy routine b$art
arlen   equ   idval+1           length of arblk in bytes
arofs   equ   arlen+1           offset in arblk to arpro field
arndm   equ   arofs+1           number of dimensions
arlbd   equ   arndm+1           low bound (first subscript)
ardim   equ   arlbd+cfp$i       dimension (first subscript)
arlb2   equ   ardim+cfp$i       low bound (second subscript)
ardm2   equ   arlb2+cfp$i       dimension (second subscript)
arpro   equ   ardim+cfp$i       array prototype (one dimension)
arvls   equ   arpro+1           start of values (one dimension)
arpr2   equ   ardm2+cfp$i       array prototype (two dimensions)
arvl2   equ   arpr2+1           start of values (two dimensions)
arsi$   equ   arlbd             number of standard fields in block
ardms   equ   arlb2-arlbd       size of info for one set of bounds
    the bounds and dimension fields are signed integer
    values and each occupy cfp$i words in the arblk.
    the length of an arblk in bytes may not exceed mxlen.
    this is required to keep name offsets garbage collectable
    the actual values are arranged in row-wise order and
    can contain a data pointer or a pointer to a trblk.
```

---

if **.cnbf**

else

```
    buffer control block (bcblk)
    a bcblk is built for every bfblk.
         +------------------------------------+
         i               bctyp                i
         +------------------------------------+
         i               idval                i
         +------------------------------------+
         i               bclen                i
         +------------------------------------+
         i               bcbuf                i
         +------------------------------------+
bctyp   equ   0                 ptr to dummy routine b$bct
bclen   equ   idval+1           defined buffer length
bcbuf   equ   bclen+1           ptr to bfblk
bcsi$   equ   bcbuf+1           size of bcblk
    a bcblk is an indirect control header for bfblk.
    the reason for not storing this data directly
    in the related bfblk is so that the bfblk can
    maintain the same skeletal structure as an scblk
    thus facilitating transparent string operations
    (for the most part).  specifically, cfp$f is the
    same for a bfblk as for an scblk.  by convention,
    whereever a buffer value is employed, the bcblk
    is pointed to.
    the corresponding bfblk is pointed to by the
    bcbuf pointer in the bcblk.
    bclen is the current defined size of the character
    array in the bfblk.  characters following the offset
    of bclen are undefined.
```

```
    string buffer block (bfblk)
    a bfblk is built by a call to buffer(...)
         +------------------------------------+
         i               bftyp                i
         +------------------------------------+
         i               bfalc                i
         +------------------------------------+
         /                                    /
         /               bfchr                /
         /                                    /
         +------------------------------------+
```

bftyp	**equ**	0	ptr to dummy routine b$bft
bfalc	**equ**	bftyp+1	allocated size of buffer
bfchr	**equ**	bfalc+1	characters of string
bfsi$	**equ**	bfchr	size of standard fields in bfblk

```
    the characters in the buffer are stored left justified.
    the final word of defined characters is always zero
    (character) padded.  any trailing allocation past the
    word containing the last character contains
    unpredictable contents and is never referenced.
    note that the offset to the characters of the string
    is given by cfp$f, as with an scblk.  however, the
    offset which is occupied by the length for an scblk
    is the total char space for bfblks, and routines which
    deal with both must account for this difference.
    the value of bfalc may not exceed mxlen.  the value of
    bclen is always less than or equal to bfalc.
```

fi

```
    code construction block (ccblk)
    at any one moment there is at most one ccblk into
    which the compiler is currently storing code (cdwrd).
         +------------------------------------+
         i                cctyp               i
         +------------------------------------+
         i                cclen               i
```

if **.csln**

```
         +------------------------------------+
         i                ccsln               i
```

fi

```
         +------------------------------------+
         i                ccuse               i
         +------------------------------------+
         /                                    /
         /                cccod               /
         /                                    /
         +------------------------------------+
```

cctyp	**equ**	0	pointer to dummy routine b$cct
cclen	**equ**	cctyp+1	length of ccblk in bytes

if **.csln**

ccsln	**equ**	cclen+1	source line number
ccuse	**equ**	ccsln+1	offset past last used word (bytes)

else

ccuse	**equ**	cclen+1	offset past last used word (bytes)

fi

cccod	**equ**	ccuse+1	start of generated code in block

```
    the reason that the ccblk is a separate block type from
    the usual cdblk is that the garbage collector must
    only process those fields which have been set (see gbcpf)
```

```
    code block (cdblk)
    a code block is built for each statement compiled during
    the initial compilation or by subsequent calls to code.
         +------------------------------------+
         i                cdjmp               i
         +------------------------------------+
         i                cdstm               i
```
if **.csln**

```
         +------------------------------------+
         i                cdsln               i
```
fi

```
         +------------------------------------+
         i                cdlen               i
         +------------------------------------+
         i                cdfal               i
         +------------------------------------+
         /                                    /
         /                cdcod               /
         /                                    /
         +------------------------------------+
```

cdjmp	**equ**	0	ptr to routine to execute statement
cdstm	**equ**	cdjmp+1	statement number

if **.csln**

cdsln	**equ**	cdstm+1	source line number
cdlen	**equ**	cdsln+1	length of cdblk in bytes
cdfal	**equ**	cdlen+1	failure exit (see below)

else

cdlen	**equ**	offs2	length of cdblk in bytes
cdfal	**equ**	offs3	failure exit (see below)

fi

cdcod	**equ**	cdfal+1	executable pseudo-code
cdsi$	**equ**	cdcod	number of standard fields in cdblk

```
    cdstm is the statement number of the current statement.
    cdjmp, cdfal are set as follows.
    1)   if the failure exit is the next statement
         cdjmp = b$cds
         cdfal = ptr to cdblk for next statement
    2)   if the failure exit is a simple label name
         cdjmp = b$cds
         cdfal is a ptr to the vrtra field of the vrblk
    3)   if there is no failure exit (-nofail mode)
         cdjmp = b$cds
         cdfal = o$unf
    4)   if the failure exit is complex or direct
         cdjmp = b$cdc
         cdfal is the offset to the o$gof word
```

```
code block (continued)
cdcod is the start of the actual code. first we describe
the code generated for an expression. in an expression,
elements are fetched by name or by value. for example,
the binary equal operator fetches its left argument
by name and its right argument by value. these two
cases generate quite different code and are described
separately. first we consider the code by value case.
generation of code by value for expressions elements.
expression            pointer to exblk or seblk
integer constant      pointer to icblk
null constant         pointer to nulls
pattern               (resulting from preevaluation)
                      =o$lpt
                      pointer to p0blk,p1blk or p2blk
real constant         pointer to rcblk
string constant       pointer to scblk
variable              pointer to vrget field of vrblk
addition              value code for left operand
                      value code for right operand
                      =o$add
affirmation           value code for operand
                      =o$aff
alternation           value code for left operand
                      value code for right operand
                      =o$alt
array reference       (case of one subscript)
                      value code for array operand
                      value code for subscript operand
                      =o$aov
                      (case of more than one subscript)
                      value code for array operand
                      value code for first subscript
                      value code for second subscript
                      ...
                      value code for last subscript
                      =o$amv
                      number of subscripts
```

```
code block (continued)
assignment            (to natural variable)
                      value code for right operand
                      pointer to vrsto field of vrblk
                      (to any other variable)
                      name code for left operand
                      value code for right operand
                      =o$ass
compile error         =o$cer
complementation       value code for operand
                      =o$com
concatenation         (case of pred func left operand)
                      value code for left operand
                      =o$pop
                      value code for right operand
                      (all other cases)
                      value code for left operand
                      value code for right operand
                      =o$cnc
cursor assignment     name code for operand
                      =o$cas
division              value code for left operand
                      value code for right operand
                      =o$dvd
exponentiation        value code for left operand
                      value code for right operand
                      =o$exp
function call         (case of call to system function)
                      value code for first argument
                      value code for second argument
                      ...
                      value code for last argument
                      pointer to svfnc field of svblk
```

```
code block (continued)
function call          (case of non-system function 1 arg)
                       value code for argument
                       =o$fns
                       pointer to vrblk for function
                       (non-system function, gt 1 arg)
                       value code for first argument
                       value code for second argument
                       ...
                       value code for last argument
                       =o$fnc
                       number of arguments
                       pointer to vrblk for function
immediate assignment   value code for left operand
                       name code for right operand
                       =o$ima
indirection            value code for operand
                       =o$inv
interrogation          value code for operand
                       =o$int
keyword reference      name code for operand
                       =o$kwv
multiplication         value code for left operand
                       value code for right operand
                       =o$mlt
name reference         (natural variable case)
                       pointer to nmblk for name
                       (all other cases)
                       name code for operand
                       =o$nam
negation               =o$nta
                       cdblk offset of o$ntc word
                       value code for operand
                       =o$ntb
                       =o$ntc
```

```
code block (continued)
pattern assignment    value code for left operand
                      name code for right operand
                      =o$pas
pattern match         value code for left operand
                      value code for right operand
                      =o$pmv
pattern replacement   name code for subject
                      value code for pattern
                      =o$pmn
                      value code for replacement
                      =o$rpl
selection             (for first alternative)
                      =o$sla
                      cdblk offset to next o$slc word
                      value code for first alternative
                      =o$slb
                      cdblk offset past alternatives
                      (for subsequent alternatives)
                      =o$slc
                      cdblk offset to next o$slc,o$sld
                      value code for alternative
                      =o$slb
                      offset in cdblk past alternatives
                      (for last alternative)
                      =o$sld
                      value code for last alternative
subtraction           value code for left operand
                      value code for right operand
                      =o$sub
```

```
code block (continued)
generation of code by name for expression elements.
variable             =o$lvn
                     pointer to vrblk
expression           (case of *natural variable)
                     =o$lvn
                     pointer to vrblk
                     (all other cases)
                     =o$lex
                     pointer to exblk
array reference      (case of one subscript)
                     value code for array operand
                     value code for subscript operand
                     =o$aon
                     (case of more than one subscript)
                     value code for array operand
                     value code for first subscript
                     value code for second subscript
                     ...
                     value code for last subscript
                     =o$amn
                     number of subscripts
compile error        =o$cer
function call        (same code as for value call)
                     =o$fne
indirection          value code for operand
                     =o$inn
keyword reference    name code for operand
                     =o$kwn
any other operand is an error in a name position
note that in this description, =o$xxx refers to the
generation of a word containing the address of another
word which contains the entry point address o$xxx.
```

```
code block (continued)
now we consider the overall structure of the code block
for a statement with possible goto fields.
first comes the code for the statement body.
the statement body is an expression to be evaluated
by value although the value is not actually required.
normal value code is generated for the body of the
statement except in the case of a pattern match by
value, in which case the following is generated.
                      value code for left operand
                      value code for right operand
                      =o$pms
next we have the code for the success goto. there are
several cases as follows.
1)   no success goto  ptr to cdblk for next statement
2)   simple label     ptr to vrtra field of vrblk
3)   complex goto     (code by name for goto operand)
                      =o$goc
4)   direct goto      (code by value for goto operand)
                      =o$god
following this we generate code for the failure goto if
it is direct or if it is complex, simple failure gotos
having been handled by an appropriate setting of the
cdfal field of the cdblk. the generated code is one
of the following.
1)   complex fgoto    =o$fif
                      =o$gof
                      name code for goto operand
                      =o$goc
2)   direct fgoto     =o$fif
                      =o$gof
                      value code for goto operand
                      =o$god
an optimization occurs if the success and failure gotos
are identical and either complex or direct. in this case,
no code is generated for the success goto and control
is allowed to fall into the failure goto on success.
```

```
compiler block (cmblk)
a compiler block (cmblk) is built by expan to represent
one node of a tree structured expression representation.
     +------------------------------------+
     i                cmidn               i
     +------------------------------------+
     i                cmlen               i
     +------------------------------------+
     i                cmtyp               i
     +------------------------------------+
     i                cmopn               i
     +------------------------------------+
     /           cmvls or cmrop           /
     /                                    /
     /                cmlop               /
     /                                    /
     +------------------------------------+
```

cmidn	**equ**	0	pointer to dummy routine b$cmt
cmlen	**equ**	cmidn+1	length of cmblk in bytes
cmtyp	**equ**	cmlen+1	type (c$xxx, see list below)
cmopn	**equ**	cmtyp+1	operand pointer (see below)
cmvls	**equ**	cmopn+1	operand value pointers (see below)
cmrop	**equ**	cmvls	right (only) operator operand
cmlop	**equ**	cmvls+1	left operator operand
cmsi$	**equ**	cmvls	number of standard fields in cmblk
cmus$	**equ**	cmsi$+1	size of unary operator cmblk
cmbs$	**equ**	cmsi$+2	size of binary operator cmblk
cmar1	**equ**	cmvls+1	array subscript pointers

```
the cmopn and cmvls fields are set as follows
array reference      cmopn = ptr to array operand
                     cmvls = ptrs to subscript operands
function call        cmopn = ptr to vrblk for function
                     cmvls = ptrs to argument operands
selection            cmopn = zero
                     cmvls = ptrs to alternate operands
unary operator       cmopn = ptr to operator dvblk
                     cmrop = ptr to operand
binary operator      cmopn = ptr to operator dvblk
                     cmrop = ptr to right operand
                     cmlop = ptr to left operand
```

```
    cmtyp is set to indicate the type of expression element
    as shown by the following table of definitions.
c$arr   equ   0                                 array reference
c$fnc   equ   c$arr+1                           function call
c$def   equ   c$fnc+1                           deferred expression (unary *)
c$ind   equ   c$def+1                           indirection (unary $)
c$key   equ   c$ind+1                           keyword reference (unary ampersand)
c$ubo   equ   c$key+1                           undefined binary operator
c$uuo   equ   c$ubo+1                           undefined unary operator
c$uo$   equ   c$uuo+1                           test value (=c$uuo+1=c$ubo+2)
c$$nm   equ   c$uuo+1                           number of codes for name operands
    the remaining types indicate expression elements which
    can only be evaluated by value (not by name).
c$bvl   equ   c$uuo+1                           binary op with value operands
c$uvl   equ   c$bvl+1                           unary operator with value operand
c$alt   equ   c$uvl+1                           alternation (binary bar)
c$cnc   equ   c$alt+1                           concatenation
c$cnp   equ   c$cnc+1                           concatenation, not pattern match
c$unm   equ   c$cnp+1                           unary op with name operand
c$bvn   equ   c$unm+1                           binary op (operands by value, name)
c$ass   equ   c$bvn+1                           assignment
c$int   equ   c$ass+1                           interrogation
c$neg   equ   c$int+1                           negation (unary not)
c$sel   equ   c$neg+1                           selection
c$pmt   equ   c$sel+1                           pattern match
c$pr$   equ   c$bvn                             last preevaluable code
c$$nv   equ   c$pmt+1                           number of different cmblk types
```

```
character table block (ctblk)
a character table block is used to hold logical character
tables for use with any,notany,span,break,breakx
patterns. each character table can be used to store
cfp$n distinct tables as bit columns. a bit column
allocated for each argument of more than one character
in length to one of the above listed pattern primitives.
     +------------------------------------+
     i                cttyp               i
     +------------------------------------+
     *                                    *
     *                                    *
     *                ctchs               *
     *                                    *
     *                                    *
     +------------------------------------+
```

cttyp	**equ**	0	pointer to dummy routine b$ctt
ctchs	**equ**	cttyp+1	start of character table words
ctsi$	**equ**	ctchs+cfp$a	number of words in ctblk

```
ctchs is cfp$a words long and consists of a one word
bit string value for each possible character in the
internal alphabet. each of the cfp$n possible bits in
a bitstring is used to form a column of bit indicators.
a bit is set on if the character is in the table and off
if the character is not present.
```

```
datatype function block (dfblk)
a datatype function is used to control the construction
of a program defined datatype object. a call to the
system function data builds a dfblk for the datatype name
note that these blocks are built in static because pdblk
length is got from dflen field.  if dfblk was in dynamic
store this would cause trouble during pass two of garbage
collection.  scblk referred to by dfnam field is also put
in static so that there are no reloc. fields. this cuts
garbage collection task appreciably for pdblks which are
likely to be present in large numbers.
     +------------------------------------+
     i               fcode                i
     +------------------------------------+
     i               fargs                i
     +------------------------------------+
     i               dflen                i
     +------------------------------------+
     i               dfpdl                i
     +------------------------------------+
     i               dfnam                i
     +------------------------------------+
     /                                    /
     /               dffld                /
     /                                    /
     +------------------------------------+
```

dflen	**equ**	fargs+1	length of dfblk in bytes
dfpdl	**equ**	dflen+1	length of corresponding pdblk
dfnam	**equ**	dfpdl+1	pointer to scblk for datatype name
dffld	**equ**	dfnam+1	start of vrblk ptrs for field names
dfflb	**equ**	dffld-1	offset behind dffld for field func
dfsi$	**equ**	dffld	number of standard fields in dfblk

```
the fcode field points to the routine b$dfc
fargs (the number of arguments) is the number of fields.
```

```
dope vector block (dvblk)
a dope vector is assembled for each possible operator in
the snobol4 language as part of the constant section.
     +------------------------------------+
     i                dvopn               i
     +------------------------------------+
     i                dvtyp               i
     +------------------------------------+
     i                dvlpr               i
     +------------------------------------+
     i                dvrpr               i
     +------------------------------------+
```

dvopn	**equ**	0	entry address (ptr to o$xxx)
dvtyp	**equ**	dvopn+1	type code (c$xxx, see cmblk)
dvlpr	**equ**	dvtyp+1	left precedence (llxxx, see below)
dvrpr	**equ**	dvlpr+1	right precedence (rrxxx, see below)
dvus$	**equ**	dvlpr+1	size of unary operator dv
dvbs$	**equ**	dvrpr+1	size of binary operator dv
dvubs	**equ**	dvus$+dvbs$	size of unop + binop (see scane)

```
the contents of the dvtyp field is copied into the cmtyp
field of the cmblk for the operator if it is used.
the cmopn field of an operator cmblk points to the dvblk
itself, providing the required entry address pointer ptr.
for normally undefined operators, the dvopn (and cmopn)
fields contain a word offset from r$uba of the function
block pointer for the operator (instead of o$xxx ptr).
for certain special operators, the dvopn field is not
required at all and is assembled as zero.
the left precedence is used in comparing an operator to
the left of some other operator. it therefore governs the
precedence of the operator towards its right operand.
the right precedence is used in comparing an operator to
the right of some other operator. it therefore governs
the precedence of the operator towards its left operand.
higher precedence values correspond to a tighter binding
capability. thus we have the left precedence lower
(higher) than the right precedence for right (left)
associative binary operators.
the left precedence of unary operators is set to an
arbitrary high value. the right value is not required and
consequently the dvrpr field is omitted for unary ops.
```

```
    table of operator precedence values
rrass   equ   10                        right equal
llass   equ   00                        left equal
rrpmt   equ   20                        right question mark
llpmt   equ   30                        left question mark
rramp   equ   40                        right ampersand
llamp   equ   50                        left ampersand
rralt   equ   70                        right vertical bar
llalt   equ   60                        left vertical bar
rrcnc   equ   90                        right blank
llcnc   equ   80                        left blank
rrats   equ   110                       right at
llats   equ   100                       left at
rrplm   equ   120                       right plus, minus
llplm   equ   130                       left plus, minus
rrnum   equ   140                       right number
llnum   equ   150                       left number
rrdvd   equ   160                       right slash
lldvd   equ   170                       left slash
rrmlt   equ   180                       right asterisk
llmlt   equ   190                       left asterisk
rrpct   equ   200                       right percent
llpct   equ   210                       left percent
rrexp   equ   230                       right exclamation
llexp   equ   220                       left exclamation
rrdld   equ   240                       right dollar, dot
lldld   equ   250                       left dollar, dot
rrnot   equ   270                       right not
llnot   equ   260                       left not
lluno   equ   999                       left all unary operators
    precedences are the same as in btl snobol4 with the
    following exceptions.
    1)   binary question mark is lowered and made left assoc-
         iative to reflect its new use for pattern matching.
    2)   alternation and concatenation are made right
         associative for greater efficiency in pattern
         construction and matching respectively. this change
         is transparent to the snobol4 programmer.
    3)   the equal sign has been added as a low precedence
         operator which is right associative to reflect its
         more general usage in this version of snobol4.
```

```
    external function block (efblk)
    an external function block is used to control the calling
    of an external function. it is built by a call to load.
         +------------------------------------+
         i                fcode               i
         +------------------------------------+
         i                fargs               i
         +------------------------------------+
         i                eflen               i
         +------------------------------------+
         i                efuse               i
         +------------------------------------+
         i                efcod               i
         +------------------------------------+
         i                efvar               i
         +------------------------------------+
         i                efrsl               i
         +------------------------------------+
         /                                    /
         /                eftar               /
         /                                    /
         +------------------------------------+
```

eflen	**equ**	fargs+1	length of efblk in bytes
efuse	**equ**	eflen+1	use count (for opsyn)
efcod	**equ**	efuse+1	ptr to code (from sysld)
efvar	**equ**	efcod+1	ptr to associated vrblk
efrsl	**equ**	efvar+1	result type (see below)
eftar	**equ**	efrsl+1	argument types (see below)
efsi$	**equ**	eftar	number of standard fields in efblk

```
    the fcode field points to the routine b$efc.
    efuse is used to keep track of multiple use when opsyn
    is employed. the function is automatically unloaded
    when there are no more references to the function.
    efrsl and eftar are type codes as follows.
         0                type is unconverted
         1                type is string
         2                type is integer
```

---

if **.cnra**

---

if **.cnlf**

```
         3                type is file
```

fi
else

```
         3                type is real
```

---

if **.cnlf**

```
         4                type is file
```

fi
fi

```
expression variable block (evblk)
in this version of spitbol, an expression can be used in
any position which would normally expect a name (for
example on the left side of equals or as the right
argument of binary dot). this corresponds to the creation
of a pseudo-variable which is represented by a pointer to
an expression variable block as follows.
     +------------------------------------+
     i                evtyp               i
     +------------------------------------+
     i                evexp               i
     +------------------------------------+
     i                evvar               i
     +------------------------------------+
```

evtyp	**equ**	0	pointer to dummy routine b$evt
evexp	**equ**	evtyp+1	pointer to exblk for expression
evvar	**equ**	evexp+1	pointer to trbev dummy trblk
evsi$	**equ**	evvar+1	size of evblk

```
the name of an expression variable is represented by a
base pointer to the evblk and an offset of evvar. this
value appears to be trapped by the dummy trbev block.
note that there is no need to allow for the case of an
expression variable which references an seblk since a
variable which is of the form *var is equivalent to var.
```

```
expression block (exblk)
an expression block is built for each expression
referenced in a program or created by eval or convert
during execution of a program.
     +------------------------------------+
     i                extyp               i
     +------------------------------------+
     i                exstm               i
```

if **.csln**

```
     +------------------------------------+
     i                exsln               i
```

fi

```
     +------------------------------------+
     i                exlen               i
     +------------------------------------+
     i                exflc               i
     +------------------------------------+
     /                                    /
     /                excod               /
     /                                    /
     +------------------------------------+
```

`extyp`	**equ**	`0`	ptr to routine b$exl to load expr
`exstm`	**equ**	`cdstm`	stores stmnt no. during evaluation

if **.csln**

`exsln`	**equ**	`exstm+1`	stores line no. during evaluation
`exlen`	**equ**	`exsln+1`	length of exblk in bytes

else

`exlen`	**equ**	`exstm+1`	length of exblk in bytes

fi

`exflc`	**equ**	`exlen+1`	failure code (=o$fex)
`excod`	**equ**	`exflc+1`	pseudo-code for expression
`exsi$`	**equ**	`excod`	number of standard fields in exblk

```
there are two cases for excod depending on whether the
expression can be evaluated by name (see description
of cdblk for details of code for expressions).
if the expression can be evaluated by name we have.
                      (code for expr by name)
                      =o$rnm
if the expression can only be evaluated by value.
                      (code for expr by value)
                      =o$rvl
```

```
field function block (ffblk)
a field function block is used to control the selection
of a field from a program defined datatype block.
a call to data creates an ffblk for each field.
     +------------------------------------+
     i                fcode               i
     +------------------------------------+
     i                fargs               i
     +------------------------------------+
     i                ffdfp               i
     +------------------------------------+
     i                ffnxt               i
     +------------------------------------+
     i                ffofs               i
     +------------------------------------+
```

ffdfp	**equ**	fargs+1	pointer to associated dfblk
ffnxt	**equ**	ffdfp+1	ptr to next ffblk on chain or zero
ffofs	**equ**	ffnxt+1	offset (bytes) to field in pdblk
ffsi$	**equ**	ffofs+1	size of ffblk in words

```
the fcode field points to the routine b$ffc.
fargs always contains one.
ffdfp is used to verify that the correct program defined
datatype is being accessed by this call.
ffdfp is non-reloc. because dfblk is in static
ffofs is used to select the appropriate field. note that
it is an actual offset (not a field number)
ffnxt is used to point to the next ffblk of the same name
in the case where there are several fields of the same
name for different datatypes. zero marks the end of chain
```

```
    integer constant block (icblk)
    an icblk is created for every integer referenced or
    created by a program. note however that certain internal
    integer values are stored as addresses (e.g. the length
    field in a string constant block)
         +------------------------------------+
         i                icget               i
         +------------------------------------+
         *                icval               *
         +------------------------------------+
icget   equ  0                                     ptr to routine b$icl to load int
icval   equ  icget+1                               integer value
icsi$   equ  icval+cfp$i                           size of icblk
    the length of the icval field is cfp$i.
```

```
    keyword variable block (kvblk)
    a kvblk is used to represent a keyword pseudo-variable.
    a kvblk is built for each keyword reference (kwnam).
         +------------------------------------+
         i                kvtyp               i
         +------------------------------------+
         i                kvvar               i
         +------------------------------------+
         i                kvnum               i
         +------------------------------------+
kvtyp   equ   0                                    pointer to dummy routine b$kvt
kvvar   equ   kvtyp+1                              pointer to dummy block trbkv
kvnum   equ   kvvar+1                              keyword number
kvsi$   equ   kvnum+1                              size of kvblk
    the name of a keyword variable is represented by a
    base pointer to the kvblk and an offset of kvvar. the
    value appears to be trapped by the pointer to trbkv.
```

```
name block (nmblk)
a name block is used wherever a name must be stored as
a value following use of the unary dot operator.
     +------------------------------------+
     i                nmtyp               i
     +------------------------------------+
     i                nmbas               i
     +------------------------------------+
     i                nmofs               i
     +------------------------------------+
```

nmtyp	**equ**	0	ptr to routine b$nml to load name
nmbas	**equ**	nmtyp+1	base pointer for variable
nmofs	**equ**	nmbas+1	offset for variable
nmsi$	**equ**	nmofs+1	size of nmblk

```
the actual field representing the contents of the name
is found nmofs bytes past the address in nmbas.
the name is split into base and offset form to avoid
creation of a pointer into the middle of a block which
could not be handled properly by the garbage collector.
a name may be built for any variable (see section on
representations of variables) this includes the
cases of pseudo-variables.
```

```
    pattern block, no parameters (p0blk)
    a p0blk is used to represent pattern nodes which do
    not require the use of any parameter values.
         +------------------------------------+
         i                pcode               i
         +------------------------------------+
         i                pthen               i
         +------------------------------------+
pcode   equ  0                                      ptr to match routine (p$xxx)
pthen   equ  pcode+1                                pointer to subsequent node
pasi$   equ  pthen+1                                size of p0blk
    pthen points to the pattern block for the subsequent
    node to be matched. this is a pointer to the pattern
    block ndnth if there is no subsequent (end of pattern)
    pcode is a pointer to the match routine for the node.
```

```
    pattern block (one parameter)
    a p1blk is used to represent pattern nodes which
    require one parameter value.
         +------------------------------------+
         i                pcode               i
         +------------------------------------+
         i                pthen               i
         +------------------------------------+
         i                parm1               i
         +------------------------------------+
```

parm1	**equ**	pthen+1	first parameter value
pbsi$	**equ**	parm1+1	size of p1blk in words

```
    see p0blk for definitions of pcode, pthen
    parm1 contains a parameter value used in matching the
    node. for example, in a len pattern, it is the integer
    argument to len. the details of the use of the parameter
    field are included in the description of the individual
    match routines. parm1 is always an address pointer which
    is processed by the garbage collector.
```

```
pattern block (two parameters)
a p2blk is used to represent pattern nodes which
require two parameter values.
     +------------------------------------+
     i                pcode               i
     +------------------------------------+
     i                pthen               i
     +------------------------------------+
     i                parm1               i
     +------------------------------------+
     i                parm2               i
     +------------------------------------+
```

parm2	**equ**	parm1+1	second parameter value
pcsi$	**equ**	parm2+1	size of p2blk in words

```
see p1blk for definitions of pcode, pthen, parm1
parm2 is a parameter which performs the same sort of
function as parm1 (see description of p1blk).
parm2 is a non-relocatable field and is not
processed by the garbage collector. accordingly, it may
not contain a pointer to a block in dynamic memory.
```

```
    program-defined datatype block
    a pdblk represents the data item formed by a call to a
    datatype function as defined by the system function data.
         +------------------------------------+
         i                pdtyp               i
         +------------------------------------+
         i                idval               i
         +------------------------------------+
         i                pddfp               i
         +------------------------------------+
         /                                    /
         /                pdfld               /
         /                                    /
         +------------------------------------+
pdtyp   equ   0                                     ptr to dummy routine b$pdt
pddfp   equ   idval+1                               ptr to associated dfblk
pdfld   equ   pddfp+1                               start of field value pointers
pdof    equ   dffld-pdfld                           difference in offset to field ptrs
pdsi$   equ   pdfld                                 size of standard fields in pdblk
pddfs   equ   dfsi$-pdsi$                           difference in dfblk, pdblk sizes
    the pddfp pointer may be used to determine the datatype
    and the names of the fields if required. the dfblk also
    contains the length of the pdblk in bytes (field dfpdl).
    pddfp is non-reloc. because dfblk is in static
    pdfld values are stored in order from left to right.
    they contain values or pointers to trblk chains.
```

```
program defined function block (pfblk)
a pfblk is created for each call to the define function
and a pointer to the pfblk placed in the proper vrblk.
     +------------------------------------+
     i                fcode               i
     +------------------------------------+
     i                fargs               i
     +------------------------------------+
     i                pflen               i
     +------------------------------------+
     i                pfvbl               i
     +------------------------------------+
     i                pfnlo               i
     +------------------------------------+
     i                pfcod               i
     +------------------------------------+
     i                pfctr               i
     +------------------------------------+
     i                pfrtr               i
     +------------------------------------+
     /                                    /
     /                pfarg               /
     /                                    /
     +------------------------------------+
```

pflen	**equ**	fargs+1	length of pfblk in bytes
pfvbl	**equ**	pflen+1	pointer to vrblk for function name
pfnlo	**equ**	pfvbl+1	number of locals
pfcod	**equ**	pfnlo+1	ptr to vrblk for entry label
pfctr	**equ**	pfcod+1	trblk ptr if call traced else 0
pfrtr	**equ**	pfctr+1	trblk ptr if return traced else 0
pfarg	**equ**	pfrtr+1	vrblk ptrs for arguments and locals
pfagb	**equ**	pfarg-1	offset behind pfarg for arg, local
pfsi$	**equ**	pfarg	number of standard fields in pfblk

```
the fcode field points to the routine b$pfc.
pfarg is stored in the following order.
     arguments (left to right)
     locals (left to right)
```

---

if **.cnra**

else

```
    real constant block (rcblk)
    an rcblk is created for every real referenced or
    created by a program.
         +------------------------------------+
         i                rcget               i
         +------------------------------------+
         *                rcval               *
         +------------------------------------+
rcget   equ   0                     ptr to routine b$rcl to load real
rcval   equ   rcget+1               real value
rcsi$   equ   rcval+cfp$r           size of rcblk
    the length of the rcval field is cfp$r.
fi
```

```
string constant block (scblk)
an scblk is built for every string referenced or created
by a program.
     +------------------------------------+
     i                scget               i
     +------------------------------------+
     i                sclen               i
     +------------------------------------+
     /                                    /
     /                schar               /
     /                                    /
     +------------------------------------+
```

scget	**equ**	0	ptr to routine b$scl to load string
sclen	**equ**	scget+1	length of string in characters
schar	**equ**	sclen+1	characters of string
scsi$	**equ**	schar	size of standard fields in scblk

```
the characters of the string are stored left justified.
the final word is padded on the right with zeros.
(i.e. the character whose internal code is zero).
the value of sclen may not exceed mxlen. this ensures
that character offsets (e.g. the pattern match cursor)
can be correctly processed by the garbage collector.
note that the offset to the characters of the string
is given in bytes by cfp$f and that this value is
automatically allowed for in plc, psc.
note that for a spitbol scblk, the value of cfp$f
is given by cfp$b*schar.
```

```
    simple expression block (seblk)
    an seblk is used to represent an expression of the form
    *(natural variable). all other expressions are exblks.
         +------------------------------------+
         i                setyp               i
         +------------------------------------+
         i                sevar               i
         +------------------------------------+
```

setyp	**equ**	0	ptr to routine b$sel to load expr
sevar	**equ**	setyp+1	ptr to vrblk for variable
sesi$	**equ**	sevar+1	length of seblk in words

```
standard variable block (svblk)
an svblk is assembled in the constant section for each
variable which satisfies one of the following conditions.
1)   it is the name of a system function
2)   it has an initial value
3)   it has a keyword association
4)   it has a standard i/o association
6)   it has a standard label association
if vrblks are constructed for any of these variables,
then the vrsvp field points to the svblk (see vrblk)
     +------------------------------------+
     i                svbit               i
     +------------------------------------+
     i                svlen               i
     +------------------------------------+
     /                svchs               /
     +------------------------------------+
     i                svknm               i
     +------------------------------------+
     i                svfnc               i
     +------------------------------------+
     i                svnar               i
     +------------------------------------+
     i                svlbl               i
     +------------------------------------+
     i                svval               i
     +------------------------------------+
```

```
    standard variable block (continued)
svbit   equ   0                        bit string indicating attributes
svlen   equ   1                        (=sclen) length of name in chars
svchs   equ   2                        (=schar) characters of name
svsi$   equ   2                        number of standard fields in svblk
svpre   equ   1                        set if preevaluation permitted
svffc   equ   svpre+svpre              set on if fast call permitted
svckw   equ   svffc+svffc              set on if keyword value constant
svprd   equ   svckw+svckw              set on if predicate function
svnbt   equ   4                        number of bits to right of svknm
svknm   equ   svprd+svprd              set on if keyword association
svfnc   equ   svknm+svknm              set on if system function
svnar   equ   svfnc+svfnc              set on if system function
svlbl   equ   svnar+svnar              set on if system label
svval   equ   svlbl+svlbl              set on if predefined value
    note that the last five bits correspond in order
    to the fields which are present (see procedure gtnvr).
    the following definitions are used in the svblk table
svfnf   equ   svfnc+svnar              function with no fast call
svfnn   equ   svfnf+svffc              function with fast call, no preeval
svfnp   equ   svfnn+svpre              function allowing preevaluation
svfpr   equ   svfnn+svprd              predicate function
svfnk   equ   svfnn+svknm              no preeval func + keyword
svkwv   equ   svknm+svval              keyword + value
svkwc   equ   svckw+svknm              keyword with constant value
svkvc   equ   svkwv+svckw              constant keyword + value
svkvl   equ   svkvc+svlbl              constant keyword + value + label
svfpk   equ   svfnp+svkvc              preeval fcn + const keywd + val
    the svpre bit allows the compiler to preevaluate a call
    to the associated system function if all the arguments
    are themselves constants. functions in this category
    must have no side effects and must never cause failure.
    the call may generate an error condition.
    the svffc bit allows the compiler to generate the special
    fast call after adjusting the number of arguments. only
    the item and apply functions fall outside this category.
    the svckw bit is set if the associated keyword value is
    a constant, thus allowing preevaluation for a value call.
    the svprd bit is set on for all predicate functions to
    enable the special concatenation code optimization.
```

```
svblk (continued)
svknm                 keyword number
     svknm is present only for a standard keyword assoc.
     it contains a keyword number as defined by the
     keyword number table given later on.
svfnc                 system function pointer
     svfnc is present only for a system function assoc.
     it is a pointer to the actual code for the system
     function. the generated code for a fast call is a
     pointer to the svfnc field of the svblk for the
     function. the vrfnc field of the vrblk points to
     this same field, in which case, it serves as the
     fcode field for the function call.
svnar                 number of function arguments
     svnar is present only for a system function assoc.
     it is the number of arguments required for a call
     to the system function. the compiler uses this
     value to adjust the number of arguments in a fast
     call and in the case of a function called through
     the vrfnc field of the vrblk, the svnar field
     serves as the fargs field for o$fnc. a special
     case occurs if this value is set to 999. this is
     used to indicate that the function has a variable
     number of arguments and causes o$fnc to pass control
     without adjusting the argument count. the only
     predefined functions using this are apply and item.
svlbl                 system label pointer
     svlbl is present only for a standard label assoc.
     it is a pointer to a system label routine (l$xxx).
     the vrlbl field of the corresponding vrblk points to
     the svlbl field of the svblk.
svval                 system value pointer
     svval is present only for a standard value.
     it is a pointer to the pattern node (ndxxx) which
     is the standard initial value of the variable.
     this value is copied to the vrval field of the vrblk
```

```
    svblk (continued)
    keyword number table
    the following table gives symbolic names for keyword
    numbers. these values are stored in the svknm field of
    svblks and in the kvnum field of kvblks. see also
    procedures asign, acess and kwnam.
    unprotected keywords with one word integer values
k$abe   equ  0                                  abend
k$anc   equ  k$abe+cfp$b                        anchor
```

---

if **.culc**

```
k$cas   equ  k$anc+cfp$b                        case
k$cod   equ  k$cas+cfp$b                        code
```

else

```
k$cod   equ  k$anc+cfp$b                        code
```

fi

---

if **.ccmk**

```
k$com   equ  k$cod+cfp$b                        compare
k$dmp   equ  k$com+cfp$b                        dump
```

else

```
k$dmp   equ  k$cod+cfp$b                        dump
```

fi

```
k$erl   equ  k$dmp+cfp$b                        errlimit
k$ert   equ  k$erl+cfp$b                        errtype
k$ftr   equ  k$ert+cfp$b                        ftrace
k$fls   equ  k$ftr+cfp$b                        fullscan
k$inp   equ  k$fls+cfp$b                        input
k$mxl   equ  k$inp+cfp$b                        maxlength
k$oup   equ  k$mxl+cfp$b                        output
```

---

if **.cnpf**

```
k$tra   equ  k$oup+cfp$b                        trace
```

else

```
k$pfl   equ  k$oup+cfp$b                        profile
k$tra   equ  k$pfl+cfp$b                        trace
```

fi

```
k$trm   equ  k$tra+cfp$b                        trim
    protected keywords with one word integer values
k$fnc   equ  k$trm+cfp$b                        fnclevel
k$lst   equ  k$fnc+cfp$b                        lastno
```

---

if **.csln**

```
k$lln   equ  k$lst+cfp$b                        lastline
k$lin   equ  k$lln+cfp$b                        line
k$stn   equ  k$lin+cfp$b                        stno
```

else

```
k$stn   equ  k$lst+cfp$b                        stno
```

fi

```
    keywords with constant pattern values
k$abo   equ  k$stn+cfp$b                        abort
k$arb   equ  k$abo+pasi$                        arb
k$bal   equ  k$arb+pasi$                        bal
k$fal   equ  k$bal+pasi$                        fail
k$fen   equ  k$fal+pasi$                        fence
```

```
k$rem   equ  k$fen+pasi$                    rem
k$suc   equ  k$rem+pasi$                    succeed
```

```
    keyword number table (continued)
    special keywords
k$alp   equ   k$suc+1                    alphabet
k$rtn   equ   k$alp+1                    rtntype
k$stc   equ   k$rtn+1                    stcount
k$etx   equ   k$stc+1                    errtext
```

---

if **.csfn**

```
k$fil   equ   k$etx+1                    file
k$lfl   equ   k$fil+1                    lastfile
k$stl   equ   k$lfl+1                    stlimit
```

else

```
k$stl   equ   k$etx+1                    stlimit
```

fi

---

if **.culk**

```
k$lcs   equ   k$stl+1                    lcase
k$ucs   equ   k$lcs+1                    ucase
```

fi

```
    relative offsets of special keywords
k$$al   equ   k$alp-k$alp                alphabet
k$$rt   equ   k$rtn-k$alp                rtntype
k$$sc   equ   k$stc-k$alp                stcount
k$$et   equ   k$etx-k$alp                errtext
```

---

if **.csfn**

```
k$$fl   equ   k$fil-k$alp                file
k$$lf   equ   k$lfl-k$alp                lastfile
```

fi

```
k$$sl   equ   k$stl-k$alp                stlimit
```

---

if **.culk**

```
k$$lc   equ   k$lcs-k$alp                lcase
k$$uc   equ   k$ucs-k$alp                ucase
k$$n$   equ   k$$uc+1                    number of special cases
```

else

```
k$$n$   equ   k$$sl+1                    number of special cases
```

fi

```
    symbols used in asign and acess procedures
k$p$$   equ   k$fnc                      first protected keyword
k$v$$   equ   k$abo                      first keyword with constant value
k$s$$   equ   k$alp                      first keyword with special acess
```

```
format of a table block (tbblk)
a table block is used to represent a table value.
it is built by a call to the table or convert functions.
     +------------------------------------+
     i                tbtyp               i
     +------------------------------------+
     i                idval               i
     +------------------------------------+
     i                tblen               i
     +------------------------------------+
     i                tbinv               i
     +------------------------------------+
     /                                    /
     /                tbbuk               /
     /                                    /
     +------------------------------------+
```

tbtyp	**equ**	0	pointer to dummy routine b$tbt
tblen	**equ**	offs2	length of tbblk in bytes
tbinv	**equ**	offs3	default initial lookup value
tbbuk	**equ**	tbinv+1	start of hash bucket pointers
tbsi$	**equ**	tbbuk	size of standard fields in tbblk
tbnbk	**equ**	11	default no. of buckets

```
the table block is a hash table which points to chains
of table element blocks representing the elements
in the table which hash into the same bucket.
tbbuk entries either point to the first teblk on the
chain or they point to the tbblk itself to indicate the
end of the chain.
```

```
    table element block (teblk)
    a table element is used to represent a single entry in
    a table (see description of tbblk format for hash table)
         +------------------------------------+
         i                tetyp               i
         +------------------------------------+
         i                tesub               i
         +------------------------------------+
         i                teval               i
         +------------------------------------+
         i                tenxt               i
         +------------------------------------+
tetyp   equ   0                                       pointer to dummy routine b$tet
tesub   equ   tetyp+1                                 subscript value
teval   equ   tesub+1                                 (=vrval) table element value
tenxt   equ   teval+1                                 link to next teblk
    see s$cnv where relation is assumed with tenxt and tbbuk
tesi$   equ   tenxt+1                                 size of teblk in words
    tenxt points to the next teblk on the hash chain from the
    tbbuk chain for this hash index. at the end of the chain,
    tenxt points back to the start of the tbblk.
    teval contains a data pointer or a trblk pointer.
    tesub contains a data pointer.
```

```
    trap block (trblk)
    a trap block is used to represent a trace or input or
    output association in response to a call to the trace
    input or output system functions. see below for details
         +------------------------------------+
         i                tridn               i
         +------------------------------------+
         i                trtyp               i
         +------------------------------------+
         i  trval or trlbl or trnxt or trkvr  i
         +------------------------------------+
         i       trtag or trter or trtrf      i
         +------------------------------------+
         i            trfnc or trfpt          i
         +------------------------------------+
```

tridn	**equ**	0	pointer to dummy routine b$trt
trtyp	**equ**	tridn+1	trap type code
trval	**equ**	trtyp+1	value of trapped variable (=vrval)
trnxt	**equ**	trval	ptr to next trblk on trblk chain
trlbl	**equ**	trval	ptr to actual label (traced label)
trkvr	**equ**	trval	vrblk pointer for keyword trace
trtag	**equ**	trval+1	trace tag
trter	**equ**	trtag	ptr to terminal vrblk or null
trtrf	**equ**	trtag	ptr to trblk holding fcblk ptr
trfnc	**equ**	trtag+1	trace function vrblk (zero if none)
trfpt	**equ**	trfnc	fcblk ptr for sysio
trsi$	**equ**	trfnc+1	number of words in trblk
trtin	**equ**	0	trace type for input association
trtac	**equ**	trtin+1	trace type for access trace
trtvl	**equ**	trtac+1	trace type for value trace
trtou	**equ**	trtvl+1	trace type for output association
trtfc	**equ**	trtou+1	trace type for fcblk identification

```
trap block (continued)
variable input association
     the value field of the variable points to a trblk
     instead of containing the data value. in the case
     of a natural variable, the vrget and vrsto fields
     contain =b$vra and =b$vrv to activate the check.
     trtyp is set to trtin
     trnxt points to next trblk or trval has variable val
     trter is a pointer to svblk if association is
     for input, terminal, else it is null.
     trtrf points to the trap block which in turn points
     to an fcblk used for i/o association.
     trfpt is the fcblk ptr returned by sysio.
variable access trace association
     the value field of the variable points to a trblk
     instead of containing the data value. in the case
     of a natural variable, the vrget and vrsto fields
     contain =b$vra and =b$vrv to activate the check.
     trtyp is set to trtac
     trnxt points to next trblk or trval has variable val
     trtag is the trace tag (0 if none)
     trfnc is the trace function vrblk ptr (0 if none)
variable value trace association
     the value field of the variable points to a trblk
     instead of containing the data value. in the case
     of a natural variable, the vrget and vrsto fields
     contain =b$vra and =b$vrv to activate the check.
     trtyp is set to trtvl
     trnxt points to next trblk or trval has variable val
     trtag is the trace tag (0 if none)
     trfnc is the trace function vrblk ptr (0 if none)
```

```
trap block (continued)
variable output association
     the value field of the variable points to a trblk
     instead of containing the data value. in the case
     of a natural variable, the vrget and vrsto fields
     contain =b$vra and =b$vrv to activate the check.
     trtyp is set to trtou
     trnxt points to next trblk or trval has variable val
     trter is a pointer to svblk if association is
     for output, terminal, else it is null.
     trtrf points to the trap block which in turn points
     to an fcblk used for i/o association.
     trfpt is the fcblk ptr returned by sysio.
function call trace
     the pfctr field of the corresponding pfblk is set
     to point to a trblk.
     trtyp is set to trtin
     trnxt is zero
     trtag is the trace tag (0 if none)
     trfnc is the trace function vrblk ptr (0 if none)
function return trace
     the pfrtr field of the corresponding pfblk is set
     to point to a trblk
     trtyp is set to trtin
     trnxt is zero
     trtag is the trace tag (0 if none)
     trfnc is the trace function vrblk ptr (0 if none)
label trace
     the vrlbl of the vrblk for the label is
     changed to point to a trblk and the vrtra field is
     set to b$vrt to activate the check.
     trtyp is set to trtin
     trlbl points to the actual label (cdblk) value
     trtag is the trace tag (0 if none)
     trfnc is the trace function vrblk ptr (0 if none)
```

```
trap block (continued)
keyword trace
     keywords which can be traced possess a unique
     location which is zero if there is no trace and
     points to a trblk if there is a trace. the locations
     are as follows.
     r$ert            errtype
     r$fnc            fnclevel
     r$stc            stcount
     the format of the trblk is as follows.
     trtyp is set to trtin
     trkvr is a pointer to the vrblk for the keyword
     trtag is the trace tag (0 if none)
     trfnc is the trace function vrblk ptr (0 if none)
input/output file arg1 trap block
     the value field of the variable points to a trblk
     instead of containing the data value. in the case of
     a natural variable, the vrget and vrsto fields
     contain =b$vra and =b$vrv. this trap block is used
     to hold a pointer to the fcblk which an
     implementation may request to hold information
     about a file.
     trtyp is set to trtfc
     trnext points to next trblk or trval is variable val
     trfnm is 0
     trfpt is the fcblk pointer.
note that when multiple traps are set on a variable
the order is in ascending value of trtyp field.
input association (if present)
access trace (if present)
value trace (if present)
output association (if present)
the actual value of the variable is stored in the trval
field of the last trblk on the chain.
this implementation does not permit trace or i/o
associations to any of the pseudo-variables.
```

```
vector block (vcblk)
a vcblk is used to represent an array value which has
one dimension whose lower bound is one. all other arrays
are represented by arblks. a vcblk is created by the
system function array (s$arr) when passed an integer arg.
     +------------------------------------+
     i                vctyp               i
     +------------------------------------+
     i                idval               i
     +------------------------------------+
     i                vclen               i
     +------------------------------------+
     i                vcvls               i
     +------------------------------------+
```

vctyp	**equ**	0	pointer to dummy routine b$vct
vclen	**equ**	offs2	length of vcblk in bytes
vcvls	**equ**	offs3	start of vector values
vcsi$	**equ**	vcvls	size of standard fields in vcblk
vcvlb	**equ**	vcvls-1	offset one word behind vcvls
vctbd	**equ**	tbsi$-vcsi$	difference in sizes - see prtvl

```
vcvls are either data pointers or trblk pointers
the dimension can be deduced from vclen.
```

```
variable block (vrblk)
a variable block is built in the static memory area
for every variable referenced or created by a program.
the order of fields is assumed in the model vrblk stnvr.
note that since these blocks only occur in the static
region, it is permissible to point to any word in
the block and this is used to provide three distinct
access points from the generated code as follows.
1)   point to vrget (first word of vrblk) to load the
     value of the variable onto the main stack.
2)   point to vrsto (second word of vrblk) to store the
     top stack element as the value of the variable.
3)   point to vrtra (fourth word of vrblk) to jump to
     the label associated with the variable name.
     +------------------------------------+
     i                vrget               i
     +------------------------------------+
     i                vrsto               i
     +------------------------------------+
     i                vrval               i
     +------------------------------------+
     i                vrtra               i
     +------------------------------------+
     i                vrlbl               i
     +------------------------------------+
     i                vrfnc               i
     +------------------------------------+
     i                vrnxt               i
     +------------------------------------+
     i                vrlen               i
     +------------------------------------+
     /                                    /
     /            vrchs = vrsvp           /
     /                                    /
     +------------------------------------+
```

```
    variable block (continued)
vrget  equ  0                  pointer to routine to load value
vrsto  equ  vrget+1            pointer to routine to store value
vrval  equ  vrsto+1            variable value
vrvlo  equ  vrval-vrsto        offset to value from store field
vrtra  equ  vrval+1            pointer to routine to jump to label
vrlbl  equ  vrtra+1            pointer to code for label
vrlbo  equ  vrlbl-vrtra        offset to label from transfer field
vrfnc  equ  vrlbl+1            pointer to function block
vrnxt  equ  vrfnc+1            pointer to next vrblk on hash chain
vrlen  equ  vrnxt+1            length of name (or zero)
vrchs  equ  vrlen+1            characters of name (vrlen gt 0)
vrsvp  equ  vrlen+1            ptr to svblk (vrlen eq 0)
vrsi$  equ  vrchs+1            number of standard fields in vrblk
vrsof  equ  vrlen-sclen        offset to dummy scblk for name
vrsvo  equ  vrsvp-vrsof        pseudo-offset to vrsvp field
    vrget = b$vrl if not input associated or access traced
    vrget = b$vra if input associated or access traced
    vrsto = b$vrs if not output associated or value traced
    vrsto = b$vrv if output associated or value traced
    vrsto = b$vre if value is protected pattern value
    vrval points to the appropriate value unless the
    variable is i/o/trace associated in which case, vrval
    points to an appropriate trblk (trap block) chain.
    vrtra = b$vrg if the label is not traced
    vrtra = b$vrt if the label is traced
    vrlbl points to a cdblk if there is a label
    vrlbl points to the svblk svlbl field for a system label
    vrlbl points to stndl for an undefined label
    vrlbl points to a trblk if the label is traced
    vrfnc points to a ffblk for a field function
    vrfnc points to a dfblk for a datatype function
    vrfnc points to a pfblk for a program defined function
    vrfnc points to a efblk for an external loaded function
    vrfnc points to svfnc (svblk) for a system function
    vrfnc points to stndf if the function is undefined
    vrnxt points to the next vrblk on this chain unless
    this is the end of the chain in which case it is zero.
    vrlen is the name length for a non-system variable.
    vrlen is zero for a system variable.
    vrchs is the name (ljrz) if vrlen is non-zero.
    vrsvp is a ptr to the svblk if vrlen is zero.
```

```
format of a non-relocatable external block (xnblk)
an xnblk is a block representing an unknown (external)
data value. the block contains no pointers to other
relocatable blocks. an xnblk is used by external function
processing or possibly for system i/o routines etc.
the macro-system itself does not use xnblks.
this type of block may be used as a file control block.
see sysfc,sysin,sysou,s$inp,s$oup for details.
     +------------------------------------+
     i                xntyp               i
     +------------------------------------+
     i                xnlen               i
     +------------------------------------+
     /                                    /
     /                xndta               /
     /                                    /
     +------------------------------------+
```

xntyp	**equ**	0	pointer to dummy routine b$xnt
xnlen	**equ**	xntyp+1	length of xnblk in bytes
xndta	**equ**	xnlen+1	data words
xnsi$	**equ**	xndta	size of standard fields in xnblk

```
note that the term non-relocatable refers to the contents
and not the block itself. an xnblk can be moved around if
it is built in the dynamic memory area.
```

```
relocatable external block (xrblk)
an xrblk is a block representing an unknown (external)
data value. the data area in this block consists only
of address values and any addresses pointing into the
dynamic memory area must point to the start of other
data blocks. see also description of xnblk.
this type of block may be used as a file control block.
see sysfc,sysin,sysou,s$inp,s$oup for details.
     +------------------------------------+
     i                xrtyp               i
     +------------------------------------+
     i                xrlen               i
     +------------------------------------+
     /                                    /
     /                xrptr               /
     /                                    /
     +------------------------------------+
```

xrtyp	**equ**	0	pointer to dummy routine b$xrt
xrlen	**equ**	xrtyp+1	length of xrblk in bytes
xrptr	**equ**	xrlen+1	start of address pointers
xrsi$	**equ**	xrptr	size of standard fields in xrblk

```
    s$cnv (convert) function switch constants.  the values
    are tied to the order of the entries in the svctb table
    and hence to the branch table in s$cnv.
cnvst   equ   8                          max standard type code for convert
```

---

if **.cnra**

```
cnvrt   equ   cnvst                      no reals - same as standard types
```
else
```
cnvrt   equ   cnvst+1                    convert code for reals
```
fi

---

if **.cnbf**

```
cnvbt   equ   cnvrt                      no buffers - same as real code
```
else
```
cnvbt   equ   cnvrt+1                    convert code for buffer
```
fi
```
cnvtt   equ   cnvbt+1                    bsw code for convert
    input image length
iniln   equ   1024                       default image length for compiler
inils   equ   1024                       image length if -sequ in effect
ionmb   equ   2                          name base used for iochn in sysio
ionmo   equ   4                          name offset used for iochn in sysio
    minimum value for keyword maxlngth
    should be larger than iniln
mnlen   equ   1024                       min value allowed keyword maxlngth
mxern   equ   329                        err num inadequate startup memory
    in general, meaningful mnemonics should be used for
    offsets. however for small integers used often in
    literals the following general definitions are provided.
num01   equ   329
num02   equ   329
num03   equ   329
num04   equ   329
num05   equ   329
num06   equ   329
num07   equ   329
num08   equ   329
num09   equ   329
num10   equ   329
nm320   equ   329
nm321   equ   329
nini8   equ   329
nini9   equ   329
thsnd   equ   329
```

```
    numbers of undefined spitbol operators
opbun   equ   5                    no. of binary undefined ops
opuun   equ   6                    no of unary undefined ops
    offsets used in prtsn, prtmi and acess
prsnf   equ   13                   offset used in prtsn
prtmf   equ   21                   offset to col 21 (prtmi)
rilen   equ   1024                 buffer length for sysri
    codes for stages of processing
stgic   equ   0                    initial compile
stgxc   equ   stgic+1              execution compile (code)
stgev   equ   stgxc+1              expression eval during execution
stgxt   equ   stgev+1              execution time
stgce   equ   stgxt+1              initial compile after end line
stgxe   equ   stgce+1              exec. compile after end line
stgnd   equ   stgce-stgic          difference in stage after end
stgee   equ   stgxe+1              eval evaluating expression
stgno   equ   stgee+1              number of codes
```

```
    statement number pad count for listr
```

---

if **.csn6**

```
stnpd   equ   6                      statement no. pad count
```

fi

---

if **.csn8**

```
stnpd   equ   8                      statement no. pad count
```

fi

---

if **.csn5**

```
stnpd   equ   5                      statement no. pad count
```

fi

```
    syntax type codes
    these codes are returned from the scane procedure.
    they are spaced 3 apart for the benefit of expan.
t$uop   equ   0                      unary operator
t$lpr   equ   t$uop+3                left paren
t$lbr   equ   t$lpr+3                left bracket
t$cma   equ   t$lbr+3                comma
t$fnc   equ   t$cma+3                function call
t$var   equ   t$fnc+3                variable
t$con   equ   t$var+3                constant
t$bop   equ   t$con+3                binary operator
t$rpr   equ   t$bop+3                right paren
t$rbr   equ   t$rpr+3                right bracket
t$col   equ   t$rbr+3                colon
t$smc   equ   t$col+3                semi-colon
    the following definitions are used only in the goto field
t$fgo   equ   t$smc+1                failure goto
t$sgo   equ   t$fgo+1                success goto
    the above codes are grouped so that codes for elements
    which can legitimately immediately precede a unary
    operator come first to facilitate operator syntax check.
t$uok   equ   t$fnc                  last code ok before unary operator
```

```
   definitions of values for expan jump table
t$uo0   equ   t$uop+0          unary operator, state zero
t$uo1   equ   t$uop+1          unary operator, state one
t$uo2   equ   t$uop+2          unary operator, state two
t$lp0   equ   t$lpr+0          left paren, state zero
t$lp1   equ   t$lpr+1          left paren, state one
t$lp2   equ   t$lpr+2          left paren, state two
t$lb0   equ   t$lbr+0          left bracket, state zero
t$lb1   equ   t$lbr+1          left bracket, state one
t$lb2   equ   t$lbr+2          left bracket, state two
t$cm0   equ   t$cma+0          comma, state zero
t$cm1   equ   t$cma+1          comma, state one
t$cm2   equ   t$cma+2          comma, state two
t$fn0   equ   t$fnc+0          function call, state zero
t$fn1   equ   t$fnc+1          function call, state one
t$fn2   equ   t$fnc+2          function call, state two
t$va0   equ   t$var+0          variable, state zero
t$va1   equ   t$var+1          variable, state one
t$va2   equ   t$var+2          variable, state two
t$co0   equ   t$con+0          constant, state zero
t$co1   equ   t$con+1          constant, state one
t$co2   equ   t$con+2          constant, state two
t$bo0   equ   t$bop+0          binary operator, state zero
t$bo1   equ   t$bop+1          binary operator, state one
t$bo2   equ   t$bop+2          binary operator, state two
t$rp0   equ   t$rpr+0          right paren, state zero
t$rp1   equ   t$rpr+1          right paren, state one
t$rp2   equ   t$rpr+2          right paren, state two
t$rb0   equ   t$rbr+0          right bracket, state zero
t$rb1   equ   t$rbr+1          right bracket, state one
t$rb2   equ   t$rbr+2          right bracket, state two
t$cl0   equ   t$col+0          colon, state zero
t$cl1   equ   t$col+1          colon, state one
t$cl2   equ   t$col+2          colon, state two
t$sm0   equ   t$smc+0          semicolon, state zero
t$sm1   equ   t$smc+1          semicolon, state one
t$sm2   equ   t$smc+2          semicolon, state two
t$nes   equ   t$sm2+1          number of entries in branch table
```

```
     definition of offsets used in control card processing
```

---

if **.culc**

```
cc$ca   equ   0                                                  -case
cc$do   equ   cc$ca+1                                            -double
```
else
```
cc$do   equ   0                                                  -double
```
fi

---

if **.ccmk**
```
cc$co   equ   cc$do+1                                            -compare
cc$du   equ   cc$co+1                                            -dump
```
else
```
cc$du   equ   cc$do+1                                            -dump
```
fi

---

if **.cinc**
```
cc$cp   equ   cc$du+1                                            -copy
cc$ej   equ   cc$cp+1                                            -eject
```
else
```
cc$ej   equ   cc$du+1                                            -eject
```
fi
```
cc$er   equ   cc$ej+1                                            -errors
cc$ex   equ   cc$er+1                                            -execute
cc$fa   equ   cc$ex+1                                            -fail
```

---

if **.cinc**
```
cc$in   equ   cc$fa+1                                            -include
```

---

if **.csln**
```
cc$ln   equ   cc$in+1                                            -line
cc$li   equ   cc$ln+1                                            -list
```
else
```
cc$li   equ   cc$in+1                                            -list
```
fi
else

---

if **.csln**
```
cc$ln   equ   cc$fa+1                                            -line
cc$li   equ   cc$ln+1                                            -list
```
else
```
cc$li   equ   cc$fa+1                                            -list
```
fi
fi
```
cc$nr   equ   cc$li+1                                            -noerrors
cc$nx   equ   cc$nr+1                                            -noexecute
cc$nf   equ   cc$nx+1                                            -nofail
cc$nl   equ   cc$nf+1                                            -nolist
cc$no   equ   cc$nl+1                                            -noopt
cc$np   equ   cc$no+1                                            -noprint
cc$op   equ   cc$np+1                                            -optimise
cc$pr   equ   cc$op+1                                            -print
cc$si   equ   cc$pr+1                                            -single
cc$sp   equ   cc$si+1                                            -space
cc$st   equ   cc$sp+1                                            -stitl
cc$ti   equ   cc$st+1                                            -title
cc$tr   equ   cc$ti+1                                            -trace
```

`cc$nc`	**equ**	`cc$tr+1`	number of control cards
`ccnoc`	**equ**	`4`	no. of chars included in match
`ccofs`	**equ**	`7`	offset to start of title/subtitle

---

if **.cinc**

`ccinm`	**equ**	`9`	max depth of include file nesting

fi

```
    definitions of stack offsets used in cmpil procedure
    see description at start of cmpil procedure for details
    of use of these locations on the stack.
cmstm   equ   0                      tree for statement body
cmsgo   equ   cmstm+1                tree for success goto
cmfgo   equ   cmsgo+1                tree for fail goto
cmcgo   equ   cmfgo+1                conditional goto flag
cmpcd   equ   cmcgo+1                previous cdblk pointer
cmffp   equ   cmpcd+1                failure fill in flag for previous
cmffc   equ   cmffp+1                failure fill in flag for current
cmsop   equ   cmffc+1                success fill in offset for previous
cmsoc   equ   cmsop+1                success fill in offset for current
cmlbl   equ   cmsoc+1                ptr to vrblk for current label
cmtra   equ   cmlbl+1                ptr to entry cdblk
cmnen   equ   cmtra+1                count of stack entries for cmpil
```

---

if **.cnpf**

else

```
    a few constants used by the profiler
pfpd1   equ   8                      pad positions ...
pfpd2   equ   20                     ... for profile ...
pfpd3   equ   32                     ... printout
pf$i2   equ   cfp$i+cfp$i            size of table entry (2 ints)
```

fi

---

if **.crel**

```
    definition of limits and adjustments that are built by
    relcr for use by the routines that relocate pointers
    after a save file is reloaded.  see reloc etc. for usage.
    a block of information is built that is used in
    relocating pointers.  there are rnsi$ instances
    of a rssi$ word structure.  each instance corresponds
    to one of the regions that a pointer might point into.
    each structure takes the form:
         +------------------------------------+
         i    address past end of section     i
         +------------------------------------+
         i  adjustment from old to new adrs   i
         +------------------------------------+
         i    address of start of section     i
         +------------------------------------+
    the instances are ordered thusly:
         +------------------------------------+
         i           dynamic storage          i
         +------------------------------------+
         i           static storage           i
         +------------------------------------+
         i       working section globals      i
         +------------------------------------+
         i          constant section          i
         +------------------------------------+
         i            code section            i
         +------------------------------------+
    symbolic names for these locations as offsets from
    the first entry are provided here.
    definitions within a section
rlend   equ  0                                    end
rladj   equ  rlend+1                              adjustment
rlstr   equ  rladj+1                              start
rssi$   equ  rlstr+1                              size of section
rnsi$   equ  5                                    number of structures
    overall definitions of all structures
rldye   equ  0                                    dynamic region end
rldya   equ  rldye+1                              dynamic region adjustment
rldys   equ  rldya+1                              dynamic region start
rlste   equ  rldys+1                              static region end
rlsta   equ  rlste+1                              static region adjustment
rlsts   equ  rlsta+1                              static region start
rlwke   equ  rlsts+1                              working section globals end
rlwka   equ  rlwke+1                              working section globals adjustment
rlwks   equ  rlwka+1                              working section globals start
rlcne   equ  rlwks+1                              constants section end
rlcna   equ  rlcne+1                              constants section adjustment
rlcns   equ  rlcna+1                              constants section start
rlcde   equ  rlcns+1                              code section end
rlcda   equ  rlcde+1                              code section adjustment
rlcds   equ  rlcda+1                              code section start
rlsi$   equ  rlcds+1                              number of fields in structure
fi
```

```
    this section consists entirely of assembled constants.
    all label names are five letters. the order is
    approximately alphabetical, but in some cases (always
    documented), constants must be placed in some special
    order which must not be disturbed.
    it must also be remembered that there is a requirement
    for no forward references which also disturbs the
    alphabetical order in some cases.
        sec                                   start of constant section
    start of constant section
c$aaa   dac   0                               first location of constant section
    free store percentage (used by alloc)
alfsp   dac   e$fsp                           free store percentage
    bit constants for general use
bits0   dbc   0                               all zero bits
bits1   dbc   1                               one bit in low order position
bits2   dbc   2                               bit in position 2
bits3   dbc   4                               bit in position 3
bits4   dbc   8                               bit in position 4
bits5   dbc   16                              bit in position 5
bits6   dbc   32                              bit in position 6
bits7   dbc   64                              bit in position 7
bits8   dbc   128                             bit in position 8
bits9   dbc   256                             bit in position 9
bit10   dbc   512                             bit in position 10
bit11   dbc   1024                            bit in position 11
bit12   dbc   2048                            bit in position 12
bitsm   dbc   cfp$m                           mask for max integer
    bit constants for svblk (svbit field) tests
btfnc   dbc   svfnc                           bit to test for function
btknm   dbc   svknm                           bit to test for keyword number
btlbl   dbc   svlbl                           bit to test for label
btffc   dbc   svffc                           bit to test for fast call
btckw   dbc   svckw                           bit to test for constant keyword
btkwv   dbc   svkwv                           bits to test for keword with value
btprd   dbc   svprd                           bit to test for predicate function
btpre   dbc   svpre                           bit to test for preevaluation
btval   dbc   svval                           bit to test for value
```

```
    list of names used for control card processing
```

---

if **.culc**

```
ccnms   dtc   svval
        dtc   svval
```
else
```
ccnms   dtc   svval
```
fi

---

if **.ccmk**
```
        dtc   svval
```
fi
```
        dtc   svval
```

---

if **.cinc**
```
        dtc   svval
```
fi
```
        dtc   svval
        dtc   svval
        dtc   svval
        dtc   svval
```

---

if **.cinc**
```
        dtc   svval
```
fi

---

if **.csln**
```
        dtc   svval
```
fi
```
        dtc   svval
        dtc   svval
        dtc   svval
        dtc   svval
        dtc   svval
        dtc   svval
        dtc   svval
        dtc   svval
        dtc   svval
        dtc   svval
        dtc   svval
        dtc   svval
        dtc   svval
        dtc   svval
    header messages for dumpr procedure (scblk format)
dmhdk   dac   b$scl                         dump of keyword values
        dac   b$scl                         dump of keyword values
        dtc   /dump of keyword              values/
dmhdv   dac   b$scl                         dump of natural variables
        dac   b$scl                         dump of natural variables
        dtc   /dump of natural              variables/
```

```
    message text for compilation statistics
encm1   dac   /dump of natural
```

---

if **.cbyt**

```
        dac   /dump of natural
        dtc   /dump of natural
encm2   dac   /dump of natural
        dac   /dump of natural
        dtc   /dump of natural
```

else

```
        dac   /dump of natural
        dtc   /dump of natural
encm2   dac   /dump of natural
        dac   /dump of natural
        dtc   /dump of natural
```

fi

```
encm3   dac   /dump of natural
        dac   /dump of natural
        dtc   /dump of natural
encm4   dac   /dump of natural
```

---

if **.ctmd**

```
        dac   /dump of natural
        dtc   /dump of natural
```

else

```
        dac   /dump of natural
        dtc   /dump of natural
```

fi

```
encm5   dac   b$scl                        execution suppressed
        dac   b$scl                        execution suppressed
        dtc   b$scl                        execution suppressed
    string constant for abnormal end
endab   dac   b$scl
        dac   b$scl
        dtc   b$scl
```

```
    memory overflow during initialisation
endmo   dac   b$scl
endml   dac   b$scl
        dtc   b$scl
    string constant for message issued by l$end
endms   dac   b$scl
        dac   b$scl
        dtc   b$scl
    fail message for stack fail section
endso   dac   b$scl                              stack overflow in garbage collector
        dac   b$scl                              stack overflow in garbage collector
        dtc   /stack overflow in                 garbage collection/
    string constant for time up
endtu   dac   /stack overflow inin
        dac   /stack overflow ininin
        dtc   /stack overflow inininin
```

```
    string constant for error message (error section)
ermms   dac   b$scl                                  error
        dac   b$scl                                  error
        dtc   b$scl                                  error
ermns   dac   b$scl                                  string / – /
        dac   b$scl                                  string / – /
        dtc   b$scl                                  string / – /
    string constant for page numbering
lstms   dac   b$scl                                  page
        dac   b$scl                                  page
        dtc   b$scl                                  page
    listing header message
headr   dac   b$scl
        dac   b$scl
        dtc   /macro spitbol version                 3.7/
headv   dac   b$scl                                  for exit() version no. check
        dac   b$scl                                  for exit() version no. check
        dtc   b$scl                                  for exit() version no. check
```

---

if **.csed**

```
    free store percentage (used by gbcol)
gbsdp   dac   e$sed                                  sediment percentage
```

fi

```
    integer constants for general use
    icbld optimisation uses the first three.
int$r   dac   e$sed
intv0   dic   +0                                     0
inton   dac   +0                                     0
intv1   dic   +1                                     1
inttw   dac   +1                                     1
intv2   dic   +2                                     2
intvt   dic   +10                                    10
intvh   dic   +100                                   100
intth   dic   +1000                                  1000
    table used in icbld optimisation
intab   dac   int$r                                  pointer to 0
        dac   inton                                  pointer to 1
        dac   inttw                                  pointer to 2
```

```
    special pattern nodes. the following pattern nodes
    consist simply of a pcode pointer, see match routines
    (p$xxx) for full details of their use and format).
ndabb   dac   p$abb                              arbno
ndabd   dac   p$abd                              arbno
ndarc   dac   p$arc                              arb
ndexb   dac   p$exb                              expression
ndfnb   dac   p$fnb                              fence()
ndfnd   dac   p$fnd                              fence()
ndexc   dac   p$exc                              expression
ndimb   dac   p$imb                              immediate assignment
ndimd   dac   p$imd                              immediate assignment
ndnth   dac   p$nth                              pattern end (null pattern)
ndpab   dac   p$pab                              pattern assignment
ndpad   dac   p$pad                              pattern assignment
nduna   dac   p$una                              anchor point movement
    keyword constant pattern nodes. the following nodes are
    used as the values of pattern keywords and the initial
    values of the corresponding natural variables. all
    nodes are in p0blk format and the order is tied to the
    definitions of corresponding k$xxx symbols.
ndabo   dac   p$abo                              abort
        dac   p$abo                              abort
ndarb   dac   p$arb                              arb
        dac   p$arb                              arb
ndbal   dac   p$bal                              bal
        dac   p$bal                              bal
ndfal   dac   p$fal                              fail
        dac   p$fal                              fail
ndfen   dac   p$fen                              fence
        dac   p$fen                              fence
ndrem   dac   p$rem                              rem
        dac   p$rem                              rem
ndsuc   dac   p$suc                              succeed
        dac   p$suc                              succeed
    null string. all null values point to this string. the
    svchs field contains a blank to provide for easy default
    processing in trace, stoptr, lpad and rpad.
    nullw contains 10 blanks which ensures an all blank word
    but for very exceptional machines.
nulls   dac   b$scl                              null string value
        dac   0                                  sclen = 0
nullw   dtc   0                                  sclen = 0
```

---

if **.culk**

```
    constant strings for lcase and ucase keywords
lcase   dac   0
        dac   0
        dtc   0
ucase   dac   0
        dac   0
        dtc   0
```

fi

```
    operator dope vectors (see dvblk format)
opdvc   dac  o$cnc                        concatenation
        dac  o$cnc                        concatenation
        dac  o$cnc                        concatenation
        dac  o$cnc                        concatenation
    opdvs is used when scanning below the top level to
    insure that the concatenation will not be later
    mistaken for pattern matching
opdvp   dac  o$cnc                        concatenation - not pattern match
        dac  o$cnc                        concatenation - not pattern match
        dac  o$cnc                        concatenation - not pattern match
        dac  o$cnc                        concatenation - not pattern match
    note that the order of the remaining entries is tied to
    the order of the coding in the scane procedure.
opdvs   dac  o$ass                        assignment
        dac  o$ass                        assignment
        dac  o$ass                        assignment
        dac  o$ass                        assignment
        dac  6                            unary equal
        dac  6                            unary equal
        dac  6                            unary equal
        dac  o$pmv                        pattern match
        dac  o$pmv                        pattern match
        dac  o$pmv                        pattern match
        dac  o$pmv                        pattern match
        dac  o$int                        interrogation
        dac  o$int                        interrogation
        dac  o$int                        interrogation
        dac  1                            binary ampersand
        dac  1                            binary ampersand
        dac  1                            binary ampersand
        dac  1                            binary ampersand
        dac  o$kwv                        keyword reference
        dac  o$kwv                        keyword reference
        dac  o$kwv                        keyword reference
        dac  o$alt                        alternation
        dac  o$alt                        alternation
        dac  o$alt                        alternation
        dac  o$alt                        alternation
```

```
operator dope vectors (continued)
    dac   5                       unary vertical bar
    dac   5                       unary vertical bar
    dac   5                       unary vertical bar
    dac   0                       binary at
    dac   0                       binary at
    dac   0                       binary at
    dac   0                       binary at
    dac   o$cas                   cursor assignment
    dac   o$cas                   cursor assignment
    dac   o$cas                   cursor assignment
    dac   2                       binary number sign
    dac   2                       binary number sign
    dac   2                       binary number sign
    dac   2                       binary number sign
    dac   7                       unary number sign
    dac   7                       unary number sign
    dac   7                       unary number sign
    dac   o$dvd                   division
    dac   o$dvd                   division
    dac   o$dvd                   division
    dac   o$dvd                   division
    dac   9                       unary slash
    dac   9                       unary slash
    dac   9                       unary slash
    dac   o$mlt                   multiplication
    dac   o$mlt                   multiplication
    dac   o$mlt                   multiplication
    dac   o$mlt                   multiplication
```

```
operator dope vectors (continued)
    dac  0                          deferred expression
    dac  0                          deferred expression
    dac  0                          deferred expression
    dac  3                          binary percent
    dac  3                          binary percent
    dac  3                          binary percent
    dac  3                          binary percent
    dac  8                          unary percent
    dac  8                          unary percent
    dac  8                          unary percent
    dac  o$exp                      exponentiation
    dac  o$exp                      exponentiation
    dac  o$exp                      exponentiation
    dac  o$exp                      exponentiation
    dac  10                         unary exclamation
    dac  10                         unary exclamation
    dac  10                         unary exclamation
    dac  o$ima                      immediate assignment
    dac  o$ima                      immediate assignment
    dac  o$ima                      immediate assignment
    dac  o$ima                      immediate assignment
    dac  o$inv                      indirection
    dac  o$inv                      indirection
    dac  o$inv                      indirection
    dac  4                          binary not
    dac  4                          binary not
    dac  4                          binary not
    dac  4                          binary not
    dac  0                          negation
    dac  0                          negation
    dac  0                          negation
```

```
    operator dope vectors (continued)
        dac   o$sub                         subtraction
        dac   o$sub                         subtraction
        dac   o$sub                         subtraction
        dac   o$sub                         subtraction
        dac   o$com                         complementation
        dac   o$com                         complementation
        dac   o$com                         complementation
        dac   o$add                         addition
        dac   o$add                         addition
        dac   o$add                         addition
        dac   o$add                         addition
        dac   o$aff                         affirmation
        dac   o$aff                         affirmation
        dac   o$aff                         affirmation
        dac   o$pas                         pattern assignment
        dac   o$pas                         pattern assignment
        dac   o$pas                         pattern assignment
        dac   o$pas                         pattern assignment
        dac   o$nam                         name reference
        dac   o$nam                         name reference
        dac   o$nam                         name reference
    special dvs for goto operators (see procedure scngf)
opdvd   dac   o$god                         direct goto
        dac   o$god                         direct goto
        dac   o$god                         direct goto
opdvn   dac   o$goc                         complex normal goto
        dac   o$goc                         complex normal goto
        dac   o$goc                         complex normal goto
```

```
    operator entry address pointers, used in code
oamn$   dac   o$amn          array ref (multi-subs by value)
oamv$   dac   o$amv          array ref (multi-subs by value)
oaon$   dac   o$aon          array ref (one sub by name)
oaov$   dac   o$aov          array ref (one sub by value)
ocer$   dac   o$cer          compilation error
ofex$   dac   o$fex          failure in expression evaluation
ofif$   dac   o$fif          failure during goto evaluation
ofnc$   dac   o$fnc          function call (more than one arg)
ofne$   dac   o$fne          function name error
ofns$   dac   o$fns          function call (single argument)
ogof$   dac   o$gof          set goto failure trap
oinn$   dac   o$inn          indirection by name
okwn$   dac   o$kwn          keyword reference by name
olex$   dac   o$lex          load expression by name
olpt$   dac   o$lpt          load pattern
olvn$   dac   o$lvn          load variable name
onta$   dac   o$nta          negation, first entry
ontb$   dac   o$ntb          negation, second entry
ontc$   dac   o$ntc          negation, third entry
opmn$   dac   o$pmn          pattern match by name
opms$   dac   o$pms          pattern match (statement)
opop$   dac   o$pop          pop top stack item
ornm$   dac   o$rnm          return name from expression
orpl$   dac   o$rpl          pattern replacement
orvl$   dac   o$rvl          return value from expression
osla$   dac   o$sla          selection, first entry
oslb$   dac   o$slb          selection, second entry
oslc$   dac   o$slc          selection, third entry
osld$   dac   o$sld          selection, fourth entry
ostp$   dac   o$stp          stop execution
ounf$   dac   o$unf          unexpected failure
```

```
    table of names of undefined binary operators for opsyn
opsnb   dac   ch$at                                    at
        dac   ch$am                                    ampersand
        dac   ch$nm                                    number
        dac   ch$pc                                    percent
        dac   ch$nt                                    not
    table of names of undefined unary operators for opsyn
opnsu   dac   ch$br                                    vertical bar
        dac   ch$eq                                    equal
        dac   ch$nm                                    number
        dac   ch$pc                                    percent
        dac   ch$sl                                    slash
        dac   ch$ex                                    exclamation
```

---

if **.cnpf**
else

```
    address const containing profile table entry size
pfi2a   dac   ch$ex
    profiler message strings
pfms1   dac   ch$ex
        dac   ch$ex
        dtc   ch$ex
pfms2   dac   ch$ex
        dac   ch$ex
        dtc   /stmt number of                          – execution time –/
pfms3   dac   /stmt number ofof                        – execution time –/
        dac   /stmt number ofofof                      – execution time –/
        dtc   /number executions                       total(msec) per excn(mcsec)/
```

fi

---

if **.cnra**
else

```
    real constants for general use. note that the constants
    starting at reav1 form a powers of ten table (used in
    gtnum and gtstg)
reav0   drc   +0.0                                     0.0
```

---

if **.cncr**
else

```
reap1   drc   +0.1                                     0.1
reap5   drc   +0.5                                     0.5
```

fi

```
reav1   drc   +1.0                                     10**0
reavt   drc   +1.0e+1                                  10**1
        drc   +1.0e+2                                  10**2
        drc   +1.0e+3                                  10**3
        drc   +1.0e+4                                  10**4
        drc   +1.0e+5                                  10**5
        drc   +1.0e+6                                  10**6
        drc   +1.0e+7                                  10**7
        drc   +1.0e+8                                  10**8
        drc   +1.0e+9                                  10**9
reatt   drc   +1.0e+10                                 10**10
```

fi

```
    string constants (scblk format) for dtype procedure
scarr   dac   b$scl                                   array
        dac   b$scl                                   array
        dtc   b$scl                                   array
```

---

if **.cnbf**
else

```
scbuf   dac   b$scl                                   buffer
        dac   b$scl                                   buffer
        dtc   b$scl                                   buffer
```

fi

```
sccod   dac   b$scl                                   code
        dac   b$scl                                   code
        dtc   b$scl                                   code
scexp   dac   b$scl                                   expression
        dac   b$scl                                   expression
        dtc   b$scl                                   expression
scext   dac   b$scl                                   external
        dac   b$scl                                   external
        dtc   b$scl                                   external
scint   dac   b$scl                                   integer
        dac   b$scl                                   integer
        dtc   b$scl                                   integer
scnam   dac   b$scl                                   name
        dac   b$scl                                   name
        dtc   b$scl                                   name
scnum   dac   b$scl                                   numeric
        dac   b$scl                                   numeric
        dtc   b$scl                                   numeric
scpat   dac   b$scl                                   pattern
        dac   b$scl                                   pattern
        dtc   b$scl                                   pattern
```

---

if **.cnra**
else

```
screa   dac   b$scl                                   real
        dac   b$scl                                   real
        dtc   b$scl                                   real
```

fi

```
scstr   dac   b$scl                                   string
        dac   b$scl                                   string
        dtc   b$scl                                   string
sctab   dac   b$scl                                   table
        dac   b$scl                                   table
        dtc   b$scl                                   table
```

---

if **.cnlf**

```
scfil   dac   b$scl                                   file (for extended load arguments)
        dac   b$scl                                   file (for extended load arguments)
        dtc   b$scl                                   file (for extended load arguments)
```

fi

```
    string constants (scblk format) for kvrtn (see retrn)
scfrt   dac   b$scl                                   freturn
        dac   b$scl                                   freturn
        dtc   b$scl                                   freturn
scnrt   dac   b$scl                                   nreturn
        dac   b$scl                                   nreturn
        dtc   b$scl                                   nreturn
scrtn   dac   b$scl                                   return
        dac   b$scl                                   return
        dtc   b$scl                                   return
    datatype name table for dtype procedure. the order of
    these entries is tied to the b$xxx definitions for blocks
    note that slots for buffer and real data types are filled
    even if these data types are conditionalized out of the
    implementation.  this is done so that the block numbering
    at bl$ar etc. remains constant in all versions.
scnmt   dac   scarr                                   arblk array
        dac   sccod                                   cdblk code
        dac   scexp                                   exblk expression
        dac   scint                                   icblk integer
        dac   scnam                                   nmblk name
        dac   scpat                                   p0blk pattern
        dac   scpat                                   p1blk pattern
        dac   scpat                                   p2blk pattern
```

---

if **.cnra**

```
        dac   nulls                                   rcblk no real in this version
```

else

```
        dac   screa                                   rcblk real
```

fi

```
        dac   scstr                                   scblk string
        dac   scexp                                   seblk expression
        dac   sctab                                   tbblk table
        dac   scarr                                   vcblk array
        dac   scext                                   xnblk external
        dac   scext                                   xrblk external
```

---

if **.cnbf**

```
        dac   nulls                                   bfblk no buffer in this version
```

else

```
        dac   scbuf                                   bfblk buffer
```

fi

---

if **.cnra**

else

```
    string constant for real zero
scre0   dac   scbuf
        dac   scbuf
        dtc   scbuf
```

fi

```
    used to re-initialise kvstl
```

if **.cs16**

```
stlim   dic   +32767                              default statement limit
```

else

if **.cs32**

```
stlim   dic   +2147483647                         default statement limit
```

else

```
stlim   dic   +50000                              default statement limit
```

fi

fi

```
    dummy function block used for undefined functions
stndf   dac   o$fun                               ptr to undefined function err call
        dac   0                                   dummy fargs count for call circuit
    dummy code block used for undefined labels
stndl   dac   l$und                               code ptr points to undefined lbl
    dummy operator block used for undefined operators
stndo   dac   o$oun                               ptr to undefined operator err call
        dac   0                                   dummy fargs count for call circuit
    standard variable block. this block is used to initialize
    the first seven fields of a newly constructed vrblk.
    its format is tied to the vrblk definitions (see gtnvr).
stnvr   dac   b$vrl                               vrget
        dac   b$vrs                               vrsto
        dac   nulls                               vrval
        dac   b$vrg                               vrtra
        dac   stndl                               vrlbl
        dac   stndf                               vrfnc
        dac   0                                   vrnxt
```

```
    messages used in end of run processing (stopr)
stpm1   dac   b$scl                              in statement
        dac   b$scl                              in statement
        dtc   b$scl                              in statement
stpm2   dac   b$scl
        dac   b$scl
        dtc   b$scl
stpm3   dac   b$scl
```

---

if **.ctmd**

```
        dac   b$scl
        dtc   b$scl
```

else

```
        dac   b$scl
        dtc   b$scl
```

fi

```
stpm4   dac   b$scl
        dac   b$scl
        dtc   b$scl
stpm5   dac   b$scl
        dac   b$scl
        dtc   b$scl
```

---

if **.csln**

```
stpm6   dac   b$scl                              in line
        dac   b$scl                              in line
        dtc   b$scl                              in line
```

fi

---

if **.csfn**

```
stpm7   dac   b$scl                              in file
        dac   b$scl                              in file
        dtc   b$scl                              in file
```

fi

```
    chars for /tu/ ending code
strtu   dtc   b$scl
    table used by convert function to check datatype name
    the entries are ordered to correspond to branch table
    in s$cnv
svctb   dac   scstr                              string
        dac   scint                              integer
        dac   scnam                              name
        dac   scpat                              pattern
        dac   scarr                              array
        dac   sctab                              table
        dac   scexp                              expression
        dac   sccod                              code
        dac   scnum                              numeric
```

---

if **.cnra**

else

```
        dac   screa                              real
```

fi

---

if **.cnbf**

else

```
        dac  scbuf                     buffer
fi
        dac  0                         zero marks end of list
```

```
    messages (scblk format) used by trace procedures
tmasb   dac   b$scl                               asterisks for trace statement no
        dac   b$scl                               asterisks for trace statement no
        dtc   b$scl                               asterisks for trace statement no
tmbeb   dac   b$scl                               blank-equal-blank
        dac   b$scl                               blank-equal-blank
        dtc   b$scl                               blank-equal-blank
    dummy trblk for expression variable
trbev   dac   b$trt                               dummy trblk
    dummy trblk for keyword variable
trbkv   dac   b$trt                               dummy trblk
    dummy code block to return control to trxeq procedure
trxdr   dac   o$txr                               block points to return routine
trxdc   dac   trxdr                               pointer to block
```

```
    standard variable blocks
    see svblk format for full details of the format. the
    vrblks are ordered by length and within each length the
    order is alphabetical by name of the variable.
v$eqf   dbc   svfpr                                  eq
        dac   svfpr                                  eq
        dtc   svfpr                                  eq
        dac   svfpr                                  eq
        dac   svfpr                                  eq
v$gef   dbc   svfpr                                  ge
        dac   svfpr                                  ge
        dtc   svfpr                                  ge
        dac   svfpr                                  ge
        dac   svfpr                                  ge
v$gtf   dbc   svfpr                                  gt
        dac   svfpr                                  gt
        dtc   svfpr                                  gt
        dac   svfpr                                  gt
        dac   svfpr                                  gt
v$lef   dbc   svfpr                                  le
        dac   svfpr                                  le
        dtc   svfpr                                  le
        dac   svfpr                                  le
        dac   svfpr                                  le
```

---

if **.cmth**

```
v$lnf   dbc   svfnp                                  ln
        dac   svfnp                                  ln
        dtc   svfnp                                  ln
        dac   svfnp                                  ln
        dac   svfnp                                  ln
```

fi

```
v$ltf   dbc   svfpr                                  lt
        dac   svfpr                                  lt
        dtc   svfpr                                  lt
        dac   svfpr                                  lt
        dac   svfpr                                  lt
v$nef   dbc   svfpr                                  ne
        dac   svfpr                                  ne
        dtc   svfpr                                  ne
        dac   svfpr                                  ne
        dac   svfpr                                  ne
```

---

if **.c370**

```
v$orf   dbc   svfnp                                  or
        dac   svfnp                                  or
        dtc   svfnp                                  or
        dac   svfnp                                  or
        dac   svfnp                                  or
```

fi

---

if **.c370**

```
v$abs   dbc   svfnp                                  abs
        dac   svfnp                                  abs
        dtc   svfnp                                  abs
```

	dac	`svfnp`	abs
	dac	`svfnp`	abs
fi			

---

if **.c370**

`v$and`	**dbc**	`svfnp`	and
	dac	`svfnp`	and
	dtc	`svfnp`	and
	dac	`svfnp`	and
	dac	`svfnp`	and
fi			
`v$any`	**dbc**	`svfnp`	any
	dac	`svfnp`	any
	dtc	`svfnp`	any
	dac	`svfnp`	any
	dac	`svfnp`	any
`v$arb`	**dbc**	`svkvc`	arb
	dac	`svkvc`	arb
	dtc	`svkvc`	arb
	dac	`svkvc`	arb
	dac	`svkvc`	arb

```
    standard variable blocks (continued)
v$arg   dbc   svfnn                                   arg
        dac   svfnn                                   arg
        dtc   svfnn                                   arg
        dac   svfnn                                   arg
        dac   svfnn                                   arg
v$bal   dbc   svkvc                                   bal
        dac   svkvc                                   bal
        dtc   svkvc                                   bal
        dac   svkvc                                   bal
        dac   svkvc                                   bal
```

---

if **.cmth**

```
v$cos   dbc   svfnp                                   cos
        dac   svfnp                                   cos
        dtc   svfnp                                   cos
        dac   svfnp                                   cos
        dac   svfnp                                   cos
```

fi

```
v$end   dbc   svlbl                                   end
        dac   svlbl                                   end
        dtc   svlbl                                   end
        dac   svlbl                                   end
```

---

if **.cmth**

```
v$exp   dbc   svfnp                                   exp
        dac   svfnp                                   exp
        dtc   svfnp                                   exp
        dac   svfnp                                   exp
        dac   svfnp                                   exp
```

fi

```
v$len   dbc   svfnp                                   len
        dac   svfnp                                   len
        dtc   svfnp                                   len
        dac   svfnp                                   len
        dac   svfnp                                   len
v$leq   dbc   svfpr                                   leq
        dac   svfpr                                   leq
        dtc   svfpr                                   leq
        dac   svfpr                                   leq
        dac   svfpr                                   leq
v$lge   dbc   svfpr                                   lge
        dac   svfpr                                   lge
        dtc   svfpr                                   lge
        dac   svfpr                                   lge
        dac   svfpr                                   lge
v$lgt   dbc   svfpr                                   lgt
        dac   svfpr                                   lgt
        dtc   svfpr                                   lgt
        dac   svfpr                                   lgt
        dac   svfpr                                   lgt
v$lle   dbc   svfpr                                   lle
        dac   svfpr                                   lle
        dtc   svfpr                                   lle
        dac   svfpr                                   lle
```

dac `svfpr` lle

```
    standard variable blocks (continued)
v$llt   dbc   svfpr                        llt
        dac   svfpr                        llt
        dtc   svfpr                        llt
        dac   svfpr                        llt
        dac   svfpr                        llt
v$lne   dbc   svfpr                        lne
        dac   svfpr                        lne
        dtc   svfpr                        lne
        dac   svfpr                        lne
        dac   svfpr                        lne
v$pos   dbc   svfnp                        pos
        dac   svfnp                        pos
        dtc   svfnp                        pos
        dac   svfnp                        pos
        dac   svfnp                        pos
v$rem   dbc   svkvc                        rem
        dac   svkvc                        rem
        dtc   svkvc                        rem
        dac   svkvc                        rem
        dac   svkvc                        rem
```

---

if **.cust**

```
v$set   dbc   svfnn                        set
        dac   svfnn                        set
        dtc   svfnn                        set
        dac   svfnn                        set
        dac   svfnn                        set
```

fi

---

if **.cmth**

```
v$sin   dbc   svfnp                        sin
        dac   svfnp                        sin
        dtc   svfnp                        sin
        dac   svfnp                        sin
        dac   svfnp                        sin
```

fi

```
v$tab   dbc   svfnp                        tab
        dac   svfnp                        tab
        dtc   svfnp                        tab
        dac   svfnp                        tab
        dac   svfnp                        tab
```

---

if **.cmth**

```
v$tan   dbc   svfnp                        tan
        dac   svfnp                        tan
        dtc   svfnp                        tan
        dac   svfnp                        tan
        dac   svfnp                        tan
```

fi

---

if **.c370**

```
v$xor   dbc   svfnp                        xor
        dac   svfnp                        xor
        dtc   svfnp                        xor
        dac   svfnp                        xor
```

```
        dac   svfnp                          xor
fi
```

```
if .cmth
v$atn   dbc   svfnp                          atan
        dac   svfnp                          atan
        dtc   svfnp                          atan
        dac   svfnp                          atan
        dac   svfnp                          atan
fi
```

```
if .culc
v$cas   dbc   svknm                          case
        dac   svknm                          case
        dtc   svknm                          case
        dac   svknm                          case
fi
v$chr   dbc   svfnp                          char
        dac   svfnp                          char
        dtc   svfnp                          char
        dac   svfnp                          char
        dac   svfnp                          char
```

```
if .cmth
v$chp   dbc   svfnp                          chop
        dac   svfnp                          chop
        dtc   svfnp                          chop
        dac   svfnp                          chop
        dac   svfnp                          chop
fi
v$cod   dbc   svfnk                          code
        dac   svfnk                          code
        dtc   svfnk                          code
        dac   svfnk                          code
        dac   svfnk                          code
        dac   svfnk                          code
v$cop   dbc   svfnn                          copy
        dac   svfnn                          copy
        dtc   svfnn                          copy
        dac   svfnn                          copy
        dac   svfnn                          copy
```

`standard variable blocks (continued)`

`v$dat`	**dbc**	`svfnn`	data
	dac	`svfnn`	data
	dtc	`svfnn`	data
	dac	`svfnn`	data
	dac	`svfnn`	data
`v$dte`	**dbc**	`svfnn`	date
	dac	`svfnn`	date
	dtc	`svfnn`	date
	dac	`svfnn`	date
	dac	`svfnn`	date
`v$dmp`	**dbc**	`svfnk`	dump
	dac	`svfnk`	dump
	dtc	`svfnk`	dump
	dac	`svfnk`	dump
	dac	`svfnk`	dump
	dac	`svfnk`	dump
`v$dup`	**dbc**	`svfnn`	dupl
	dac	`svfnn`	dupl
	dtc	`svfnn`	dupl
	dac	`svfnn`	dupl
	dac	`svfnn`	dupl
`v$evl`	**dbc**	`svfnn`	eval
	dac	`svfnn`	eval
	dtc	`svfnn`	eval
	dac	`svfnn`	eval
	dac	`svfnn`	eval

---

if **.cnex**

else

`v$ext`	**dbc**	`svfnn`	exit
	dac	`svfnn`	exit
	dtc	`svfnn`	exit
	dac	`svfnn`	exit
	dac	`svfnn`	exit

fi

`v$fal`	**dbc**	`svkvc`	fail
	dac	`svkvc`	fail
	dtc	`svkvc`	fail
	dac	`svkvc`	fail
	dac	`svkvc`	fail

---

if **.csfn**

`v$fil`	**dbc**	`svknm`	file
	dac	`svknm`	file
	dtc	`svknm`	file
	dac	`svknm`	file

fi

`v$hst`	**dbc**	`svfnn`	host
	dac	`svfnn`	host
	dtc	`svfnn`	host
	dac	`svfnn`	host
	dac	`svfnn`	host

```
    standard variable blocks (continued)
v$itm   dbc   svfnf                                   item
        dac   svfnf                                   item
        dtc   svfnf                                   item
        dac   svfnf                                   item
        dac   svfnf                                   item
```

```
if .csln
v$lin   dbc   svknm                                   line
        dac   svknm                                   line
        dtc   svknm                                   line
        dac   svknm                                   line
fi
```

```
if .cnld
else
v$lod   dbc   svfnn                                   load
        dac   svfnn                                   load
        dtc   svfnn                                   load
        dac   svfnn                                   load
        dac   svfnn                                   load
fi
v$lpd   dbc   svfnp                                   lpad
        dac   svfnp                                   lpad
        dtc   svfnp                                   lpad
        dac   svfnp                                   lpad
        dac   svfnp                                   lpad
v$rpd   dbc   svfnp                                   rpad
        dac   svfnp                                   rpad
        dtc   svfnp                                   rpad
        dac   svfnp                                   rpad
        dac   svfnp                                   rpad
v$rps   dbc   svfnp                                   rpos
        dac   svfnp                                   rpos
        dtc   svfnp                                   rpos
        dac   svfnp                                   rpos
        dac   svfnp                                   rpos
v$rtb   dbc   svfnp                                   rtab
        dac   svfnp                                   rtab
        dtc   svfnp                                   rtab
        dac   svfnp                                   rtab
        dac   svfnp                                   rtab
v$si$   dbc   svfnp                                   size
        dac   svfnp                                   size
        dtc   svfnp                                   size
        dac   svfnp                                   size
        dac   svfnp                                   size
```

```
if .cnsr
else
v$srt   dbc   svfnn                                   sort
        dac   svfnn                                   sort
        dtc   svfnn                                   sort
        dac   svfnn                                   sort
        dac   svfnn                                   sort
```

fi

`v$spn`	**dbc**	`svfnp`	span
	dac	`svfnp`	span
	dtc	`svfnp`	span
	dac	`svfnp`	span
	dac	`svfnp`	span

if **.cmth**

```
v$sqr   dbc  svfnp                    sqrt
        dac  svfnp                    sqrt
        dtc  svfnp                    sqrt
        dac  svfnp                    sqrt
        dac  svfnp                    sqrt
```

fi

```
v$stn   dbc  svknm                    stno
        dac  svknm                    stno
        dtc  svknm                    stno
        dac  svknm                    stno
v$tim   dbc  svfnn                    time
        dac  svfnn                    time
        dtc  svfnn                    time
        dac  svfnn                    time
        dac  svfnn                    time
v$trm   dbc  svfnk                    trim
        dac  svfnk                    trim
        dtc  svfnk                    trim
        dac  svfnk                    trim
        dac  svfnk                    trim
        dac  svfnk                    trim
v$abe   dbc  svknm                    abend
        dac  svknm                    abend
        dtc  svknm                    abend
        dac  svknm                    abend
v$abo   dbc  svkvl                    abort
        dac  svkvl                    abort
        dtc  svkvl                    abort
        dac  svkvl                    abort
        dac  svkvl                    abort
        dac  svkvl                    abort
v$app   dbc  svfnf                    apply
        dac  svfnf                    apply
        dtc  svfnf                    apply
        dac  svfnf                    apply
        dac  svfnf                    apply
v$abn   dbc  svfnp                    arbno
        dac  svfnp                    arbno
        dtc  svfnp                    arbno
        dac  svfnp                    arbno
        dac  svfnp                    arbno
v$arr   dbc  svfnn                    array
        dac  svfnn                    array
        dtc  svfnn                    array
        dac  svfnn                    array
        dac  svfnn                    array
```

```
    standard variable blocks (continued)
v$brk   dbc  svfnp                                   break
        dac  svfnp                                   break
        dtc  svfnp                                   break
        dac  svfnp                                   break
        dac  svfnp                                   break
v$clr   dbc  svfnn                                   clear
        dac  svfnn                                   clear
        dtc  svfnn                                   clear
        dac  svfnn                                   clear
        dac  svfnn                                   clear
```

---

if **.c370**

```
v$cmp   dbc  svfnp                                   compl
        dac  svfnp                                   compl
        dtc  svfnp                                   compl
        dac  svfnp                                   compl
        dac  svfnp                                   compl
```

fi

```
v$ejc   dbc  svfnn                                   eject
        dac  svfnn                                   eject
        dtc  svfnn                                   eject
        dac  svfnn                                   eject
        dac  svfnn                                   eject
v$fen   dbc  svfpk                                   fence
        dac  svfpk                                   fence
        dtc  svfpk                                   fence
        dac  svfpk                                   fence
        dac  svfpk                                   fence
        dac  svfpk                                   fence
        dac  svfpk                                   fence
v$fld   dbc  svfnn                                   field
        dac  svfnn                                   field
        dtc  svfnn                                   field
        dac  svfnn                                   field
        dac  svfnn                                   field
v$idn   dbc  svfpr                                   ident
        dac  svfpr                                   ident
        dtc  svfpr                                   ident
        dac  svfpr                                   ident
        dac  svfpr                                   ident
v$inp   dbc  svfnk                                   input
        dac  svfnk                                   input
        dtc  svfnk                                   input
        dac  svfnk                                   input
        dac  svfnk                                   input
        dac  svfnk                                   input
```

---

if **.culk**

```
v$lcs   dbc  svkwc                                   lcase
        dac  svkwc                                   lcase
        dtc  svkwc                                   lcase
        dac  svkwc                                   lcase
```

fi

```
v$loc   dbc  svfnn                                   local
```

dac	`svfnn`	local
dtc	`svfnn`	local
dac	`svfnn`	local
dac	`svfnn`	local

```
    standard variable blocks (continued)
v$ops   dbc   svfnn                                  opsyn
        dac   svfnn                                  opsyn
        dtc   svfnn                                  opsyn
        dac   svfnn                                  opsyn
        dac   svfnn                                  opsyn
v$rmd   dbc   svfnp                                  remdr
        dac   svfnp                                  remdr
        dtc   svfnp                                  remdr
        dac   svfnp                                  remdr
        dac   svfnp                                  remdr
```

---

if **.cnsr**
else

```
v$rsr   dbc   svfnn                                  rsort
        dac   svfnn                                  rsort
        dtc   svfnn                                  rsort
        dac   svfnn                                  rsort
        dac   svfnn                                  rsort
```

fi

```
v$tbl   dbc   svfnn                                  table
        dac   svfnn                                  table
        dtc   svfnn                                  table
        dac   svfnn                                  table
        dac   svfnn                                  table
v$tra   dbc   svfnk                                  trace
        dac   svfnk                                  trace
        dtc   svfnk                                  trace
        dac   svfnk                                  trace
        dac   svfnk                                  trace
        dac   svfnk                                  trace
```

---

if **.culk**

```
v$ucs   dbc   svkwc                                  ucase
        dac   svkwc                                  ucase
        dtc   svkwc                                  ucase
        dac   svkwc                                  ucase
```

fi

```
v$anc   dbc   svknm                                  anchor
        dac   svknm                                  anchor
        dtc   svknm                                  anchor
        dac   svknm                                  anchor
```

---

if **.cnbf**
else

```
v$apn   dbc   svfnn                                  append
        dac   svfnn                                  append
        dtc   svfnn                                  append
        dac   svfnn                                  append
        dac   svfnn                                  append
```

fi

```
v$bkx   dbc   svfnp                                  breakx
        dac   svfnp                                  breakx
        dtc   svfnp                                  breakx
        dac   svfnp                                  breakx
```

	dac	`svfnp`	breakx
if **.cnbf**			
else			
`v$buf`	**dbc**	`svfnn`	buffer
	dac	`svfnn`	buffer
	dtc	`svfnn`	buffer
	dac	`svfnn`	buffer
	dac	`svfnn`	buffer
fi			
`v$def`	**dbc**	`svfnn`	define
	dac	`svfnn`	define
	dtc	`svfnn`	define
	dac	`svfnn`	define
	dac	`svfnn`	define
`v$det`	**dbc**	`svfnn`	detach
	dac	`svfnn`	detach
	dtc	`svfnn`	detach
	dac	`svfnn`	detach
	dac	`svfnn`	detach

```
    standard variable blocks (continued)
v$dif   dbc   svfpr                                   differ
        dac   svfpr                                   differ
        dtc   svfpr                                   differ
        dac   svfpr                                   differ
        dac   svfpr                                   differ
v$ftr   dbc   svknm                                   ftrace
        dac   svknm                                   ftrace
        dtc   svknm                                   ftrace
        dac   svknm                                   ftrace
```

```
if .cnbf
else
v$ins   dbc   svfnn                                   insert
        dac   svfnn                                   insert
        dtc   svfnn                                   insert
        dac   svfnn                                   insert
        dac   svfnn                                   insert
fi
v$lst   dbc   svknm                                   lastno
        dac   svknm                                   lastno
        dtc   svknm                                   lastno
        dac   svknm                                   lastno
v$nay   dbc   svfnp                                   notany
        dac   svfnp                                   notany
        dtc   svfnp                                   notany
        dac   svfnp                                   notany
        dac   svfnp                                   notany
v$oup   dbc   svfnk                                   output
        dac   svfnk                                   output
        dtc   svfnk                                   output
        dac   svfnk                                   output
        dac   svfnk                                   output
        dac   svfnk                                   output
v$ret   dbc   svlbl                                   return
        dac   svlbl                                   return
        dtc   svlbl                                   return
        dac   svlbl                                   return
v$rew   dbc   svfnn                                   rewind
        dac   svfnn                                   rewind
        dtc   svfnn                                   rewind
        dac   svfnn                                   rewind
        dac   svfnn                                   rewind
v$stt   dbc   svfnn                                   stoptr
        dac   svfnn                                   stoptr
        dtc   svfnn                                   stoptr
        dac   svfnn                                   stoptr
        dac   svfnn                                   stoptr
```

```
    standard variable blocks (continued)
v$sub   dbc   svfnn                         substr
        dac   svfnn                         substr
        dtc   svfnn                         substr
        dac   svfnn                         substr
        dac   svfnn                         substr
v$unl   dbc   svfnn                         unload
        dac   svfnn                         unload
        dtc   svfnn                         unload
        dac   svfnn                         unload
        dac   svfnn                         unload
v$col   dbc   svfnn                         collect
        dac   svfnn                         collect
        dtc   svfnn                         collect
        dac   svfnn                         collect
        dac   svfnn                         collect
```

---

if **.ccmk**

```
v$com   dbc   svknm                         compare
        dac   svknm                         compare
        dtc   svknm                         compare
        dac   svknm                         compare
```

fi

```
v$cnv   dbc   svfnn                         convert
        dac   svfnn                         convert
        dtc   svfnn                         convert
        dac   svfnn                         convert
        dac   svfnn                         convert
v$enf   dbc   svfnn                         endfile
        dac   svfnn                         endfile
        dtc   svfnn                         endfile
        dac   svfnn                         endfile
        dac   svfnn                         endfile
v$etx   dbc   svknm                         errtext
        dac   svknm                         errtext
        dtc   svknm                         errtext
        dac   svknm                         errtext
v$ert   dbc   svknm                         errtype
        dac   svknm                         errtype
        dtc   svknm                         errtype
        dac   svknm                         errtype
v$frt   dbc   svlbl                         freturn
        dac   svlbl                         freturn
        dtc   svlbl                         freturn
        dac   svlbl                         freturn
v$int   dbc   svfpr                         integer
        dac   svfpr                         integer
        dtc   svfpr                         integer
        dac   svfpr                         integer
        dac   svfpr                         integer
v$nrt   dbc   svlbl                         nreturn
        dac   svlbl                         nreturn
        dtc   svlbl                         nreturn
        dac   svlbl                         nreturn
```

if **.cnpf**

else

v$pfl	**dbc**	svknm	profile
	dac	svknm	profile
	dtc	svknm	profile
	dac	svknm	profile

fi

v$rpl	**dbc**	svfnp	replace
	dac	svfnp	replace
	dtc	svfnp	replace
	dac	svfnp	replace
	dac	svfnp	replace
v$rvs	**dbc**	svfnp	reverse
	dac	svfnp	reverse
	dtc	svfnp	reverse
	dac	svfnp	reverse
	dac	svfnp	reverse
v$rtn	**dbc**	svknm	rtntype
	dac	svknm	rtntype
	dtc	svknm	rtntype
	dac	svknm	rtntype
v$stx	**dbc**	svfnn	setexit
	dac	svfnn	setexit
	dtc	svfnn	setexit
	dac	svfnn	setexit
	dac	svfnn	setexit
v$stc	**dbc**	svknm	stcount
	dac	svknm	stcount
	dtc	svknm	stcount
	dac	svknm	stcount
v$stl	**dbc**	svknm	stlimit
	dac	svknm	stlimit
	dtc	svknm	stlimit
	dac	svknm	stlimit
v$suc	**dbc**	svkvc	succeed
	dac	svkvc	succeed
	dtc	svkvc	succeed
	dac	svkvc	succeed
	dac	svkvc	succeed
v$alp	**dbc**	svkwc	alphabet
	dac	svkwc	alphabet
	dtc	svkwc	alphabet
	dac	svkwc	alphabet
v$cnt	**dbc**	svlbl	continue
	dac	svlbl	continue
	dtc	svlbl	continue
	dac	svlbl	continue

```
    standard variable blocks (continued)
v$dtp   dbc   svfnp                                   datatype
        dac   svfnp                                   datatype
        dtc   svfnp                                   datatype
        dac   svfnp                                   datatype
        dac   svfnp                                   datatype
v$erl   dbc   svknm                                   errlimit
        dac   svknm                                   errlimit
        dtc   svknm                                   errlimit
        dac   svknm                                   errlimit
v$fnc   dbc   svknm                                   fnclevel
        dac   svknm                                   fnclevel
        dtc   svknm                                   fnclevel
        dac   svknm                                   fnclevel
v$fls   dbc   svknm                                   fullscan
        dac   svknm                                   fullscan
        dtc   svknm                                   fullscan
        dac   svknm                                   fullscan
```

---

if **.csfn**

```
v$lfl   dbc   svknm                                   lastfile
        dac   svknm                                   lastfile
        dtc   svknm                                   lastfile
        dac   svknm                                   lastfile
```

fi

---

if **.csln**

```
v$lln   dbc   svknm                                   lastline
        dac   svknm                                   lastline
        dtc   svknm                                   lastline
        dac   svknm                                   lastline
```

fi

```
v$mxl   dbc   svknm                                   maxlngth
        dac   svknm                                   maxlngth
        dtc   svknm                                   maxlngth
        dac   svknm                                   maxlngth
v$ter   dbc   0                                       terminal
        dac   0                                       terminal
        dtc   0                                       terminal
        dac   0                                       terminal
```

---

if **.cbsp**

```
v$bsp   dbc   svfnn                                   backspace
        dac   svfnn                                   backspace
        dtc   svfnn                                   backspace
        dac   svfnn                                   backspace
        dac   svfnn                                   backspace
```

fi

```
v$pro   dbc   svfnn                                   prototype
        dac   svfnn                                   prototype
        dtc   svfnn                                   prototype
        dac   svfnn                                   prototype
        dac   svfnn                                   prototype
v$scn   dbc   svlbl                                   scontinue
        dac   svlbl                                   scontinue
```

dtc	`svlbl`	scontinue
dac	`svlbl`	scontinue
dbc	`0`	dummy entry to end list
dac	`10`	length gt 9 (scontinue)

```
    list of svblk pointers for keywords to be dumped. the
    list is in the order which appears on the dump output.
vdmkw   dac   v$anc                          anchor
```
---
if **.culc**
```
        dac   v$cas                          ccase
```
fi
```
        dac   v$cod                          code
```
---
if **.ccmk**

---
if **.ccmc**
```
        dac   v$com                          compare
```
else
```
        dac   1                              compare not printed
```
fi
fi
```
        dac   v$dmp                          dump
        dac   v$erl                          errlimit
        dac   v$etx                          errtext
        dac   v$ert                          errtype
```
---
if **.csfn**
```
        dac   v$fil                          file
```
fi
```
        dac   v$fnc                          fnclevel
        dac   v$ftr                          ftrace
        dac   v$fls                          fullscan
        dac   v$inp                          input
```
---
if **.csfn**
```
        dac   v$lfl                          lastfile
```
fi

---
if **.csln**
```
        dac   v$lln                          lastline
```
fi
```
        dac   v$lst                          lastno
```
---
if **.csln**
```
        dac   v$lin                          line
```
fi
```
        dac   v$mxl                          maxlength
        dac   v$oup                          output
```
---
if **.cnpf**
else
```
        dac   v$pfl                          profile
```
fi
```
        dac   v$rtn                          rtntype
        dac   v$stc                          stcount
        dac   v$stl                          stlimit
        dac   v$stn                          stno
        dac   v$tra                          trace
        dac   v$trm                          trim
        dac   0                              end of list
    table used by gtnvr to search svblk lists
vsrch   dac   0                              dummy entry to get proper indexing
```

	dac	`v$eqf`	start of 1 char variables (none)
	dac	`v$eqf`	start of 2 char variables
	dac	`v$any`	start of 3 char variables

if **.cmth**

	dac	`v$atn`	start of 4 char variables

else

if **.culc**

	dac	`v$cas`	start of 4 char variables

else

	dac	`v$chr`	start of 4 char variables

fi

fi

	dac	`v$abe`	start of 5 char variables
	dac	`v$anc`	start of 6 char variables
	dac	`v$col`	start of 7 char variables
	dac	`v$alp`	start of 8 char variables

if **.cbsp**

	dac	`v$bsp`	start of 9 char variables

else

	dac	`v$pro`	start of 9 char variables

fi

`last location in constant section`

`c$yyy`	**dac**	`0`	last location in constant section

```
the working storage section contains areas which are
changed during execution of the program. the value
assembled is the initial value before execution starts.
all these areas are fixed length areas. variable length
data is stored in the static or dynamic regions of the
allocated data areas.
the values in this area are described either as work
areas or as global values. a work area is used in an
ephemeral manner and the value is not saved from one
entry into a routine to another. a global value is a
less temporary location whose value is saved from one
call to another.
w$aaa marks the start of the working section whilst
w$yyy marks its end.  g$aaa marks the division between
temporary and global values.
global values are further subdivided to facilitate
processing by the garbage collector. r$aaa through
r$yyy are global values that may point into dynamic
storage and hence must be relocated after each garbage
collection.  they also serve as root pointers to all
allocated data that must be preserved.  pointers between
a$aaa and r$aaa may point into code, static storage,
or mark the limits of dynamic memory.  these pointers
must be adjusted when the working section is saved to a
file and subsequently reloaded at a different address.
a general part of the approach in this program is not
to overlap work areas between procedures even though a
small amount of space could be saved. such overlap is
considered a source of program errors and decreases the
information left behind after a system crash of any kind.
the names of these locations are labels with five letter
(a-y,$) names. as far as possible the order is kept
alphabetical by these names but in some cases there
are slight departures caused by other order requirements.
unless otherwise documented, the order of work areas
does not affect the execution of the spitbol program.
```

sec start of working storage section

```
    this area is not cleared by initial code
cmlab   dac   b$scl                     string used to check label legality
        dac   b$scl                     string used to check label legality
        dtc   b$scl                     string used to check label legality
    label to mark start of work area
w$aaa   dac   b$scl
    work areas for acess procedure
actrm   dac   0                         trim indicator
    work areas for alloc procedure
aldyn   dac   0                         amount of dynamic store
allia   dic   +0                        dump ia
allsv   dac   0                         save wb in alloc
    work areas for alost procedure
alsta   dac   0                         save wa in alost
    work areas for array function (s$arr)
arcdm   dac   0                         count dimensions
arnel   dic   +0                        count elements
arptr   dac   0                         offset ptr into arblk
arsvl   dic   +0                        save integer low bound
```

```
    work areas for arref routine
arfsi   dic   +0                 save current evolving subscript
arfxs   dac   0                  save base stack pointer
    work areas for b$efc block routine
befof   dac   0                  save offset ptr into efblk
    work areas for b$pfc block routine
bpfpf   dac   0                  save pfblk pointer
bpfsv   dac   0                  save old function value
bpfxt   dac   0                  pointer to stacked arguments
    work area for collect function (s$col)
clsvi   dic   +0                 save integer argument
    work areas value for cncrd
cnscc   dac   0                  pointer to control card string
cnswc   dac   0                  word count
cnr$t   dac   0                  pointer to r$ttl or r$stl
    work areas for convert function (s$cnv)
cnvtp   dac   0                  save ptr into scvtb
    work areas for data function (s$dat)
datdv   dac   0                  save vrblk ptr for datatype name
datxs   dac   0                  save initial stack pointer
    work areas for define function (s$def)
deflb   dac   0                  save vrblk ptr for label
defna   dac   0                  count function arguments
defvr   dac   0                  save vrblk ptr for function name
defxs   dac   0                  save initial stack pointer
    work areas for dumpr procedure
dmarg   dac   0                  dump argument
dmpsa   dac   0                  preserve wa over prtvl call
```

---

if **.ccmk**

```
dmpsb   dac   0                  preserve wb over syscm call
```

fi

```
dmpsv   dac   0                  general scratch save
dmvch   dac   0                  chain pointer for variable blocks
dmpch   dac   0                  save sorted vrblk chain pointer
dmpkb   dac   0                  dummy kvblk for use in dumpr
dmpkt   dac   0                  kvvar trblk ptr (must follow dmpkb)
dmpkn   dac   0                  keyword number (must follow dmpkt)
    work area for dtach
dtcnb   dac   0                  name base
dtcnm   dac   0                  name ptr
    work areas for dupl function (s$dup)
dupsi   dic   +0                 store integer string length
    work area for endfile (s$enf)
enfch   dac   0                  for iochn chain head
```

```
    work areas for ertex
ertwa   dac   0                          save wa
ertwb   dac   0                          save wb
    work areas for evali
evlin   dac   0                          dummy pattern block pcode
evlis   dac   0                          then node (must follow evlin)
evliv   dac   0                          value of parm1 (must follow evlis)
evlio   dac   0                          ptr to original node
evlif   dac   0                          flag for simple/complex argument
    work area for expan
expsv   dac   0                          save op dope vector pointer
    work areas for gbcol procedure
gbcfl   dac   0                          garbage collector active flag
gbclm   dac   0                          pointer to last move block (pass 3)
gbcnm   dac   0                          dummy first move block
gbcns   dac   0                          rest of dummy block (follows gbcnm)
```

---

if **.csed**

---

if **.cepp**

else

```
gbcmk   dac   0                          bias when marking entry point
```

fi

```
gbcia   dic   +0                         dump ia
gbcsd   dac   0                          first address beyond sediment
gbcsf   dac   0                          free space within sediment
```

fi

```
gbsva   dac   0                          save wa
gbsvb   dac   0                          save wb
gbsvc   dac   0                          save wc
    work areas for gtnvr procedure
gnvhe   dac   0                          ptr to end of hash chain
gnvnw   dac   0                          number of words in string name
gnvsa   dac   0                          save wa
gnvsb   dac   0                          save wb
gnvsp   dac   0                          pointer into vsrch table
gnvst   dac   0                          pointer to chars of string
    work areas for gtarr
gtawa   dac   0                          save wa
    work areas for gtint
gtina   dac   0                          save wa
gtinb   dac   0                          save wb
```

```
    work areas for gtnum procedure
gtnnf   dac   0                        zero/nonzero for result +/-
gtnsi   dic   +0                       general integer save
```

---

if **.cnra**
else

```
gtndf   dac   0                        0/1 for dec point so far no/yes
gtnes   dac   0                        zero/nonzero exponent +/-
gtnex   dic   +0                       real exponent
gtnsc   dac   0                        scale (places after point)
gtnsr   drc   +0.0                     general real save
gtnrd   dac   0                        flag for ok real number
```

fi

```
    work areas for gtpat procedure
gtpsb   dac   0                        save wb
    work areas for gtstg procedure
gtssf   dac   0                        0/1 for result +/-
gtsvc   dac   0                        save wc
gtsvb   dac   0                        save wb
```

---

if **.cnra**
else

---

if **.cncr**
else

```
gtses   dac   0                        char + or - for exponent +/-
gtsrs   drc   +0.0                     general real save
```

fi
fi

```
    work areas for gtvar procedure
gtvrc   dac   0                        save wc
```

---

if **.cnbf**
else

```
    work areas for insbf
insab   dac   0                        entry wa + entry wb
insln   dac   0                        length of insertion string
inssa   dac   0                        save entry wa
inssb   dac   0                        save entry wb
inssc   dac   0                        save entry wc
```

fi

```
    work areas for ioput
ioptt   dac   0                        type of association
```

---

if **.cnld**
else

```
    work areas for load function
lodfn   dac   0                        pointer to vrblk for func name
lodna   dac   0                        count number of arguments
```

fi

---

if **.cnpf**
else

```
    work area for profiler
pfsvw   dac   0                        to save a w-reg
```

fi

```
    work areas for prtnm procedure
```

```
prnsi   dic    +0                          scratch integer loc
    work areas for prtsn procedure
prsna   dac    0                           save wa
    work areas for prtst procedure
prsva   dac    0                           save wa
prsvb   dac    0                           save wb
prsvc   dac    0                           save char counter
    work area for prtnl
prtsa   dac    0                           save wa
prtsb   dac    0                           save wb
    work area for prtvl
prvsi   dac    0                           save idval
    work areas for pattern match routines
psave   dac    0                           temporary save for current node ptr
psavc   dac    0                           save cursor in p$spn, p$str
```

---

if **.crel**

```
    work area for relaj routine
rlals   dac    0                           ptr to list of bounds and adjusts
    work area for reldn routine
rldcd   dac    0                           save code adjustment
rldst   dac    0                           save static adjustment
rldls   dac    0                           save list pointer
```

fi

```
    work areas for retrn routine
rtnbp   dac    0                           to save a block pointer
rtnfv   dac    0                           new function value (result)
rtnsv   dac    0                           old function value (saved value)
    work areas for substr function (s$sub)
sbssv   dac    0                           save third argument
    work areas for scan procedure
scnsa   dac    0                           save wa
scnsb   dac    0                           save wb
scnsc   dac    0                           save wc
scnof   dac    0                           save offset
```

---

if **.cnsr**

else

```
    work area used by sorta, sortc, sortf, sorth
srtdf   dac   0                          datatype field name
srtfd   dac   0                          found dfblk address
srtff   dac   0                          found field name
srtfo   dac   0                          offset to field name
srtnr   dac   0                          number of rows
srtof   dac   0                          offset within row to sort key
srtrt   dac   0                          root offset
srts1   dac   0                          save offset 1
srts2   dac   0                          save offset 2
srtsc   dac   0                          save wc
srtsf   dac   0                          sort array first row offset
srtsn   dac   0                          save n
srtso   dac   0                          offset to a(0)
srtsr   dac   0                          0, non-zero for sort, rsort
srtst   dac   0                          stride from one row to next
srtwc   dac   0                          dump wc
fi
    work areas for stopr routine
stpsi   dic   +0                         save value of stcount
stpti   dic   +0                         save time elapsed
    work areas for tfind procedure
tfnsi   dic   +0                         number of headers
    work areas for xscan procedure
xscrt   dac   0                          save return code
xscwb   dac   0                          save register wb
    start of global values in working section
g$aaa   dac   0
    global value for alloc procedure
alfsf   dic   +0                         factor in free store pcntage check
    global values for cmpil procedure
cmerc   dac   0                          count of initial compile errors
cmpln   dac   0                          line number of first line of stmt
cmpxs   dac   0                          save stack ptr in case of errors
cmpsn   dac   1                          number of next statement to compile
    global values for cncrd
```

```
if .cinc
cnsil   dac   0                          save scnil during include process.
cnind   dac   0                          current include file nest level
cnspt   dac   0                          save scnpt during include process.
fi
cnttl   dac   0                          flag for -title, -stitl
    global flag for suppression of compilation statistics.
cpsts   dac   0                          suppress comp. stats if non zero
    global values for control card switches
cswdb   dac   0                          0/1 for -single/-double
cswer   dac   0                          0/1 for -errors/-noerrors
cswex   dac   0                          0/1 for -execute/-noexecute
cswfl   dac   1                          0/1 for -nofail/-fail
cswin   dac   iniln                      xxx for -inxxx
cswls   dac   1                          0/1 for -nolist/-list
cswno   dac   0                          0/1 for -optimise/-noopt
cswpr   dac   0                          0/1 for -noprint/-print
```

```
    global location used by patst procedure
ctmsk   dbc   0                              last bit position used in r$ctp
curid   dac   0                              current id value
```

```
    global value for cdwrd procedure
cwcof   dac   0                                  next word offset in current ccblk
```

---

if **.csed**

```
    global locations for dynamic storage pointers
dnams   dac   0                                  size of sediment in baus
```

fi

```
    global area for error processing.
erich   dac   0                                  copy error reports to int.chan if 1
erlst   dac   0                                  for listr when errors go to int.ch.
errft   dac   0                                  fatal error flag
errsp   dac   0                                  error suppression flag
    global flag for suppression of execution stats
exsts   dac   0                                  suppress exec stats if set
    global values for exfal and return
flprt   dac   0                                  location of fail offset for return
flptr   dac   0                                  location of failure offset on stack
    global location to count garbage collections (gbcol)
```

---

if **.csed**

```
gbsed   dic   +0                                 factor in sediment pcntage check
```

fi

```
gbcnt   dac   0                                  count of garbage collections
    global value for gtcod and gtexp
gtcef   dac   0                                  save fail ptr in case of error
    global locations for gtstg procedure
```

---

if **.cnra**

else

---

if **.cncr**

else

```
gtsrn   drc   +0.0                               rounding factor 0.5*10**-cfp$s
gtssc   drc   +0.0                               scaling value 10**cfp$s
```

fi

fi

```
gtswk   dac   0                                  ptr to work area for gtstg
    global flag for header printing
headp   dac   0                                  header printed flag
    global values for variable hash table
hshnb   dic   +0                                 number of hash buckets
    global areas for init
initr   dac   0                                  save terminal flag
```

```
    global values for keyword values which are stored as one
    word integers. these values must be assembled in the
    following order (as dictated by k$xxx definition values).
kvabe   dac   0                        abend
kvanc   dac   0                        anchor
```

---

if **.culc**

```
kvcas   dac   0                        case
```

fi

```
kvcod   dac   0                        code
```

---

if **.ccmk**

```
kvcom   dac   0                        compare
```

fi

```
kvdmp   dac   0                        dump
kverl   dac   0                        errlimit
kvert   dac   0                        errtype
kvftr   dac   0                        ftrace
kvfls   dac   1                        fullscan
kvinp   dac   1                        input
kvmxl   dac   5000                     maxlength
kvoup   dac   1                        output
```

---

if **.cnpf**

else

```
kvpfl   dac   0                        profile
```

fi

```
kvtra   dac   0                        trace
kvtrm   dac   0                        trim
kvfnc   dac   0                        fnclevel
kvlst   dac   0                        lastno
```

---

if **.csln**

```
kvlln   dac   0                        lastline
kvlin   dac   0                        line
```

fi

```
kvstn   dac   0                        stno
    global values for other keywords
kvalp   dac   0                        alphabet
kvrtn   dac   nulls                    rtntype (scblk pointer)
```

---

if **.cs16**

```
kvstl   dic   +32767                   stlimit
kvstc   dic   +32767                   stcount (counts down from stlimit)
```

else

---

if **.cs32**

```
kvstl   dic   +2147483647              stlimit
kvstc   dic   +2147483647              stcount (counts down from stlimit)
```

else

```
kvstl   dic   +50000                   stlimit
kvstc   dic   +50000                   stcount (counts down from stlimit)
```

fi

fi

```
    global values for listr procedure
```

---

if **.cinc**

```
lstid   dac   0                        include depth of current image
```

fi

```
lstlc   dac   0        count lines on source list page
lstnp   dac   0        max number of lines on page
lstpf   dac   1        set nonzero if current image listed
lstpg   dac   0        current source list page number
lstpo   dac   0        offset to page nnn message
lstsn   dac   0        remember last stmnum listed
    global maximum size of spitbol objects
mxlen   dac   0        initialised by sysmx call
    global execution control variable
noxeq   dac   0        set non-zero to inhibit execution
```

---

if **.cnpf**

else

```
    global profiler values locations
pfdmp   dac   0        set non-0 if &profile set non-0
pffnc   dac   0        set non-0 if funct just entered
pfstm   dic   +0       to store starting time of stmt
pfetm   dic   +0       to store ending time of stmt
pfnte   dac   0        nr of table entries
pfste   dic   +0       gets int rep of table entry size
```

fi

```
    global values used in pattern match routines
pmdfl   dac  0                         pattern assignment flag
pmhbs   dac  0                         history stack base pointer
pmssl   dac  0                         length of subject string in chars
```

---

if **.cpol**

```
    global values for interface polling (syspl)
polcs   dac  1                         poll interval start value
polct   dac  1                         poll interval counter
```

fi

```
    global flags used for standard file listing options
prich   dac  0                         printer on interactive channel
prstd   dac  0                         tested by prtpg
prsto   dac  0                         standard listing option flag
    global values for print procedures
prbuf   dac  0                         ptr to print bfr in static
precl   dac  0                         extended/compact listing flag
prlen   dac  0                         length of print buffer in chars
prlnw   dac  0                         length of print buffer in words
profs   dac  0                         offset to next location in prbuf
prtef   dac  0                         endfile flag
```

```
    global area for readr
rdcln   dac   0                                  current statement line number
rdnln   dac   0                                  next statement line number
    global amount of memory reserved for end of execution
rsmem   dac   0                                  reserve memory
    global area for stmgo counters
stmcs   dac   1                                  counter startup value
stmct   dac   1                                  counter active value
    adjustable global values
    all the pointers in this section can point to the
    dynamic or the static region.
    when a save file is reloaded, these pointers must
    be adjusted if static or dynamic memory is now
    at a different address.  see routine reloc for
    additional information.
    some values cannot be move here because of adjacency
    constraints.  they are handled specially by reloc et al.
    these values are kvrtn,
    values gtswk, kvalp, and prbuf are reinitialized by
    procedure insta, and do not need to appear here.
    values flprt, flptr, gtcef, and stbas point into the
    stack and are explicitly adjusted by osint's restart
    procedure.
a$aaa   dac   0                                  start of adjustable values
cmpss   dac   0                                  save subroutine stack ptr
dnamb   dac   0                                  start of dynamic area
dnamp   dac   0                                  next available loc in dynamic area
dname   dac   0                                  end of available dynamic area
hshtb   dac   0                                  pointer to start of vrblk hash tabl
hshte   dac   0                                  pointer past end of vrblk hash tabl
iniss   dac   0                                  save subroutine stack ptr
pftbl   dac   0                                  gets adrs of (imag) table base
prnmv   dac   0                                  vrblk ptr from last name search
statb   dac   0                                  start of static area
state   dac   0                                  end of static area
stxvr   dac   nulls                              vrblk pointer or null
    relocatable global values
    all the pointers in this section can point to blocks in
    the dynamic storage area and must be relocated by the
    garbage collector. they are identified by r$xxx names.
r$aaa   dac   0                                  start of relocatable values
r$arf   dac   0                                  array block pointer for arref
r$ccb   dac   0                                  ptr to ccblk being built (cdwrd)
r$cim   dac   0                                  ptr to current compiler input str
r$cmp   dac   0                                  copy of r$cim used in cmpil
r$cni   dac   0                                  ptr to next compiler input string
r$cnt   dac   0                                  cdblk pointer for setexit continue
r$cod   dac   0                                  pointer to current cdblk or exblk
r$ctp   dac   0                                  ptr to current ctblk for patst
r$cts   dac   0                                  ptr to last string scanned by patst
r$ert   dac   0                                  trblk pointer for errtype trace
r$etx   dac   nulls                              pointer to errtext string
r$exs   dac   0                                  = save xl in expdm
```

`r$fcb`	**dac**	`0`	fcblk chain head
`r$fnc`	**dac**	`0`	trblk pointer for fnclevel trace
`r$gtc`	**dac**	`0`	keep code ptr for gtcod,gtexp

---

if **.cinc**

`r$ici`	**dac**	`0`	saved r$cim during include process.

---

if **.csfn**

`r$ifa`	**dac**	`0`	array of file names by incl. depth
`r$ifl`	**dac**	`0`	array of line nums by include depth

fi

`r$ifn`	**dac**	`0`	last include file name
`r$inc`	**dac**	`0`	table of include file names seen

fi

`r$io1`	**dac**	`0`	file arg1 for ioput
`r$io2`	**dac**	`0`	file arg2 for ioput
`r$iof`	**dac**	`0`	fcblk ptr or 0
`r$ion`	**dac**	`0`	name base ptr
`r$iop`	**dac**	`0`	predecessor block ptr for ioput
`r$iot`	**dac**	`0`	trblk ptr for ioput

---

if **.cnbf**

else

`r$pmb`	**dac**	`0`	buffer ptr in pattern match

fi

`r$pms`	**dac**	`0`	subject string ptr in pattern match
`r$ra2`	**dac**	`0`	replace second argument last time
`r$ra3`	**dac**	`0`	replace third argument last time
`r$rpt`	**dac**	`0`	ptr to ctblk replace table last usd
`r$scp`	**dac**	`0`	save pointer from last scane call

---

if **.csfn**

`r$sfc`	**dac**	`nulls`	current source file name
`r$sfn`	**dac**	`0`	ptr to source file name table

fi

`r$sxl`	**dac**	`0`	preserve xl in sortc
`r$sxr`	**dac**	`0`	preserve xr in sorta/sortc
`r$stc`	**dac**	`0`	trblk pointer for stcount trace
`r$stl`	**dac**	`0`	source listing sub-title
`r$sxc`	**dac**	`0`	code (cdblk) ptr for setexit trap
`r$ttl`	**dac**	`nulls`	source listing title
`r$xsc`	**dac**	`0`	string pointer for xscan

```
    the remaining pointers in this list are used to point
    to function blocks for normally undefined operators.
r$uba   dac  stndo                            binary at
r$ubm   dac  stndo                            binary ampersand
r$ubn   dac  stndo                            binary number sign
r$ubp   dac  stndo                            binary percent
r$ubt   dac  stndo                            binary not
r$uub   dac  stndo                            unary vertical bar
r$uue   dac  stndo                            unary equal
r$uun   dac  stndo                            unary number sign
r$uup   dac  stndo                            unary percent
r$uus   dac  stndo                            unary slash
r$uux   dac  stndo                            unary exclamation
r$yyy   dac  0                                last relocatable location
    global locations used in scan procedure
scnbl   dac  0                                set non-zero if scanned past blanks
scncc   dac  0                                non-zero to scan control card name
scngo   dac  0                                set non-zero to scan goto field
scnil   dac  0                                length of current input image
scnpt   dac  0                                pointer to next location in r$cim
scnrs   dac  0                                set non-zero to signal rescan
scnse   dac  0                                start of current element
scntp   dac  0                                save syntax type from last call
    global value for indicating stage (see error section)
stage   dac  0                                initial value = initial compile
```

```
    global stack pointer
stbas   dac   0                     pointer past stack base
    global values for setexit function (s$stx)
stxoc   dac   0                     code pointer offset
stxof   dac   0                     failure offset
    global value for time keeping
timsx   dic   +0                    time at start of execution
timup   dac   0                     set when time up occurs
    global values for xscan and xscni procedures
xsofs   dac   0                     offset to current location in r$xsc
    label to mark end of working section
w$yyy   dac   0
```

	sec		start of program section
`s$aaa`	**ent**	`bl$$i`	mark start of code

if **.crel**

```
    relocation
    the following section provides services to osint to
    relocate portions of the workspace.  it is used when
    a saved memory image must be restarted at a different
    location.
    relaj -- relocate a list of pointers
    (wa)                 ptr past last pointer of list
    (wb)                 ptr to first pointer of list
    (xl)                 list of boundaries and adjustments
    jsr  relaj           call to process list of pointers
    (wb)                 destroyed
relaj   prc  e,0                                    entry point
        mov  xr,-(xs)                               save xr
        mov  wa,-(xs)                               save wa
        mov  xl,rlals                               save ptr to list of bounds
        mov  wb,xr                                  ptr to first pointer to process
    merge here to check if done
rlaj0   mov  rlals,xl                               restore xl
        bne  xr,(xs),rlaj1                          proceed if more to do
        mov  (xs)+,wa                               restore wa
        mov  (xs)+,xr                               restore xr
        exi                                         return to caller
    merge here to process next pointer on list
rlaj1   mov  (xr),wa                                load next pointer on list
        lct  wb,=rnsi$                              number of sections of adjusters
    merge here to process next section of stack list
rlaj2   bgt  wa,rlend(xl),rlaj3                     ok if past end of section
        blt  wa,rlstr(xl),rlaj3                     or if before start of section
        add  rladj(xl),wa                           within section, add adjustment
        mov  wa,(xr)                                return updated ptr to memory
        brn  rlaj4                                  done with this pointer
    here if not within section
rlaj3   add  *rssi$,xl                              advance to next section
        bct  wb,rlaj2                               jump if more to go
    here when finished processing one pointer
rlaj4   ica  xr                                     increment to next ptr on list
        brn  rlaj0                                  jump to check for completion
        enp                                         end procedure relaj
```

```
    relcr -- create relocation info after save file reload
    (wa)                original s$aaa code section adr
    (wb)                original c$aaa constant section adr
    (wc)                original g$aaa working section adr
    (xr)                ptr to start of static region
    (cp)                ptr to start of dynamic region
    (xl)                ptr to area to receive information
    jsr  relcr          create relocation information
    (wa,wb,wc,xr)       destroyed
    a block of information is built at (xl) that is used
    in relocating pointers.  there are rnsi$ instances
    of a rssi$ word structure.  each instance corresponds
    to one of the regions that a pointer might point into.
    the layout of this structure is shown in the definitions
    section, together with symbolic definitions of the
    entries as offsets from xl.
relcr prc  e,0                  entry point
      add  *rlsi$,xl            point past build area
      mov  wa,-(xl)             save original code address
      mov  =s$aaa,wa            compute adjustment
      sub  (xl),wa              as new s$aaa minus original s$aaa
      mov  wa,-(xl)             save code adjustment
      mov  =s$yyy,wa            end of target code section
      sub  =s$aaa,wa            length of code section
      add  num01(xl),wa         plus original start address
      mov  wa,-(xl)             end of original code section
      mov  wb,-(xl)             save constant section address
      mov  =c$aaa,wb            start of constants section
      mov  =c$yyy,wa            end of constants section
      sub  wb,wa                length of constants section
      sub  (xl),wb              new c$aaa minus original c$aaa
      mov  wb,-(xl)             save constant adjustment
      add  num01(xl),wa         length plus original start adr
      mov  wa,-(xl)             save as end of original constants
      mov  wc,-(xl)             save working globals address
      mov  =g$aaa,wc            start of working globals section
      mov  =w$yyy,wa            end of working section
      sub  wc,wa                length of working globals
      sub  (xl),wc              new g$aaa minus original g$aaa
      mov  wc,-(xl)             save working globals adjustment
      add  num01(xl),wa         length plus original start adr
      mov  wa,-(xl)             save as end of working globals
      mov  statb,wb             old start of static region
      mov  wb,-(xl)             save
      sub  wb,xr                compute adjustment
      mov  xr,-(xl)             save new statb minus old statb
      mov  state,-(xl)          old end of static region
      mov  dnamb,wb             old start of dynamic region
      mov  wb,-(xl)             save
      scp  wa                   new start of dynamic
      sub  wb,wa                compute adjustment
      mov  wa,-(xl)             save new dnamb minus old dnamb
      mov  dnamp,wc             old end of dynamic region in use
```

```
mov  wc,-(xl)        save as end of old dynamic region
exi  wc,-(xl)        save as end of old dynamic region
enp  wc,-(xl)        save as end of old dynamic region
```

```
    reldn -- relocate pointers in the dynamic region
    (xl)                 list of boundaries and adjustments
    (xr)                 ptr to first location to process
    (wc)                 ptr past last location to process
    jsr  reldn           call to process blocks in dynamic
    (wa,wb,wc,xr)        destroyed
    processes all blocks in the dynamic region.  within a
    block, pointers to the code section, constant section,
    working globals section, static region, and dynamic
    region are relocated as needed.
reldn   prc  e,0                                    entry point
        mov  rlcda(xl),rldcd                        save code adjustment
        mov  rlsta(xl),rldst                        save static adjustment
        mov  xl,rldls                               save list pointer
    merge here to process the next block in dynamic
rld01   add  rldcd,(xr)                             adjust block type word
        mov  (xr),xl                                load block type word
        lei  xl                                     load entry point id (bl$xx)
    block type switch. note that blocks with no relocatable
    fields just return to rld05 to continue to next block.
    note that dfblks do not appear in dynamic, only in static.
    ccblks and cmblks are not live when a save file is
    created, and can be skipped.
    further note:  static blocks other than vrblks discovered
    while scanning dynamic must be adjusted at this time.
    see processing of ffblk for example.
```

```
    reldn (continued)
        bsw  xl,bl$$$                                switch on block type
        iff  bl$ar,rld03                             arblk
if .cnbf
        iff  bl$bc,rld05                             bcblk - dummy to fill out iffs
else
        iff  bl$bc,rld06                             bcblk
fi
        iff  bl$bf,rld05                             bfblk
        iff  bl$cc,rld05                             ccblk
        iff  bl$cd,rld07                             cdblk
        iff  bl$cm,rld05                             cmblk
        iff  bl$ct,rld05                             ctblk
        iff  bl$df,rld05                             dfblk
        iff  bl$ef,rld08                             efblk
        iff  bl$ev,rld09                             evblk
        iff  bl$ex,rld10                             exblk
        iff  bl$ff,rld11                             ffblk
        iff  bl$ic,rld05                             icblk
        iff  bl$kv,rld13                             kvblk
        iff  bl$nm,rld13                             nmblk
        iff  bl$p0,rld13                             p0blk
        iff  bl$p1,rld14                             p1blk
        iff  bl$p2,rld14                             p2blk
        iff  bl$pd,rld15                             pdblk
        iff  bl$pf,rld16                             pfblk
if .cnra
else
        iff  bl$rc,rld05                             rcblk
fi
        iff  bl$sc,rld05                             scblk
        iff  bl$se,rld13                             seblk
        iff  bl$tb,rld17                             tbblk
        iff  bl$te,rld18                             teblk
        iff  bl$tr,rld19                             trblk
        iff  bl$vc,rld17                             vcblk
        iff  bl$xn,rld05                             xnblk
        iff  bl$xr,rld20                             xrblk
        esw                                          end of jump table
    arblk
rld03   mov  arlen(xr),wa                            load length
        mov  arofs(xr),wb                            set offset to 1st reloc fld (arpro)
    merge here to process pointers in a block
    (xr)                 ptr to current block
    (wc)                 ptr past last location to process
    (wa)                 length (reloc flds + flds at start)
    (wb)                 offset to first reloc field
rld04   add  xr,wa                                   point past last reloc field
        add  xr,wb                                   point to first reloc field
        mov  rldls,xl                                point to list of bounds
        jsr  relaj                                   adjust pointers
```

```
    reldn (continued)
    merge here to advance to next block
    (xr)                 ptr to current block
    (wc)                 ptr past last location to process
rld05   mov  (xr),wa                          block type word
        jsr  blkln                            get length of block
        add  wa,xr                            point to next block
        blt  xr,wc,rld01                      continue if more to process
        mov  rldls,xl                         restore xl
        exi                                   return to caller if done
```

---

if **.cnbf**

else

```
    bcblk
rld06   mov  *bcsi$,wa                        set length
        mov  *bcbuf,wb                        and offset
        brn  rld04                            all set
```

fi

```
    cdblk
rld07   mov  cdlen(xr),wa                     load length
        mov  *cdfal,wb                        set offset
        bne  (xr),=b$cdc,rld04                jump back if not complex goto
        mov  *cdcod,wb                        do not process cdfal word
        brn  rld04                            jump back
    efblk
    if the efcod word points to an xnblk, the xnblk type
    word will not be adjusted.  since this is implementation
    dependent, we will not worry about it.
rld08   mov  *efrsl,wa                        set length
        mov  *efcod,wb                        and offset
        brn  rld04                            all set
    evblk
rld09   mov  *offs3,wa                        point past third field
        mov  *evexp,wb                        set offset
        brn  rld04                            all set
    exblk
rld10   mov  exlen(xr),wa                     load length
        mov  *exflc,wb                        set offset
        brn  rld04                            jump back
```

```
    reldn (continued)
    ffblk
    this block contains a ptr to a dfblk in the static rgn.
    because there are multiple ffblks pointing to the same
    dfblk (one for each field name), we only process the
    dfblk when we encounter the ffblk for the first field.
    the dfblk in turn contains a pointer to an scblk within
    static.
rld11   bne  ffofs(xr),*pdfld,rld12          skip dfblk if not first field
        mov  xr,-(xs)                        save xr
        mov  ffdfp(xr),xr                    load old ptr to dfblk
        add  rldst,xr                        current location of dfblk
        add  rldcd,(xr)                      adjust dfblk type word
        mov  dflen(xr),wa                    length of dfblk
        mov  *dfnam,wb                       offset to dfnam field
        add  xr,wa                           point past last reloc field
        add  xr,wb                           point to first reloc field
        mov  rldls,xl                        point to list of bounds
        jsr  relaj                           adjust pointers
        mov  dfnam(xr),xr                    pointer to static scblk
        add  rldcd,(xr)                      adjust scblk type word
        mov  (xs)+,xr                        restore ffblk pointer
    ffblk (continued)
    merge here to set up for adjustment of ptrs in ffblk
rld12   mov  *ffofs,wa                       set length
        mov  *ffdfp,wb                       set offset
        brn  rld04                           all set
    kvblk, nmblk, p0blk, seblk
rld13   mov  *offs2,wa                       point past second field
        mov  *offs1,wb                       offset is one (only reloc fld is 2)
        brn  rld04                           all set
    p1blk, p2blk
    in p2blks, parm2 contains either a bit mask or the
    name offset of a variable.  it never requires relocation.
rld14   mov  *parm2,wa                       length (parm2 is non-relocatable)
        mov  *pthen,wb                       set offset
        brn  rld04                           all set
    pdblk
    note that the dfblk pointed to by this pdblk was
    processed when the ffblk was encountered.  because
    the data function will be called before any records are
    defined, the ffblk is encountered before any
    corresponding pdblk.
rld15   mov  pddfp(xr),xl                    load ptr to dfblk
        add  rldst,xl                        adjust for static relocation
        mov  dfpdl(xl),wa                    get pdblk length
        mov  *pddfp,wb                       set offset
        brn  rld04                           all set
```

```
    reldn (continued)
    pfblk
rld16   add   rldst,pfvbl(xr)           adjust non-contiguous field
        mov   pflen(xr),wa              get pfblk length
        mov   *pfcod,wb                 offset to first reloc
        brn   rld04                     all set
    tbblk, vcblk
rld17   mov   offs2(xr),wa              load length
        mov   *offs3,wb                 set offset
        brn   rld04                     jump back
    teblk
rld18   mov   *tesi$,wa                 set length
        mov   *tesub,wb                 and offset
        brn   rld04                     all set
    trblk
rld19   mov   *trsi$,wa                 set length
        mov   *trval,wb                 and offset
        brn   rld04                     all set
    xrblk
rld20   mov   xrlen(xr),wa              load length
        mov   *xrptr,wb                 set offset
        brn   rld04                     jump back
        enp                             end procedure reldn
```

```
    reloc -- relocate storage after save file reload
    (xl)                  list of boundaries and adjustments
    jsr  reloc            relocate all pointers
    (wa,wb,wc,xr)         destroyed
    the list of boundaries and adjustments pointed to by
    register xl is created by a call to relcr, which should
    be consulted for information on its structure.
reloc   prc   e,0                      entry point
        mov   rldys(xl),xr             old start of dynamic
        mov   rldye(xl),wc             old end of dynamic
        add   rldya(xl),xr             create new start of dynamic
        add   rldya(xl),wc             create new end of dynamic
        jsr   reldn                    relocate pointers in dynamic
        jsr   relws                    relocate pointers in working sect
        jsr   relst                    relocate pointers in static
        exi                            return to caller
        enp                            end procedure reloc
```

```
    relst -- relocate pointers in the static region
    (xl)                 list of boundaries and adjustments
    jsr  relst           call to process blocks in static
    (wa,wb,wc,xr)        destroyed
    only vrblks on the hash chain and any profile block are
    processed.  other static blocks (dfblks) are processed
    during processing of dynamic blocks.
    global work locations will be processed at this point,
    so pointers there can be relied upon.
relst   prc  e,0                                   entry point
        mov  pftbl,xr                              profile table
        bze  xr,rls01                              branch if no table allocated
        add  rlcda(xl),(xr)                        adjust block type word
    here after dealing with profiler
rls01   mov  hshtb,wc                              point to start of hash table
        mov  wc,wb                                 point to first hash bucket
        mov  hshte,wa                              point beyond hash table
        jsr  relaj                                 adjust bucket pointers
    loop through slots in hash table
rls02   beq  wc,hshte,rls05                        done if none left
        mov  wc,xr                                 else copy slot pointer
        ica  wc                                    bump slot pointer
        sub  *vrnxt,xr                             set offset to merge into loop
    loop through vrblks on one hash chain
rls03   mov  vrnxt(xr),xr                          point to next vrblk on chain
        bze  xr,rls02                              jump for next bucket if chain end
        mov  *vrlen,wa                             offset of first loc past ptr fields
        mov  *vrget,wb                             offset of first location in vrblk
        bnz  vrlen(xr),rls04                       jump if not system variable
        mov  *vrsi$,wa                             offset to include vrsvp field
    merge here to process fields of vrblk
rls04   add  xr,wa                                 create end ptr
        add  xr,wb                                 create start ptr
        jsr  relaj                                 adjust pointers in vrblk
        brn  rls03                                 check for another vrblk on chain
    here when all vrblks processed
rls05   exi                                        return to caller
        enp                                        end procedure relst
```

```
    relws -- relocate pointers in the working section
    (xl)                  list of boundaries and adjustments
    jsr  relws            call to process working section
    (wa,wb,wc,xr)         destroyed
    pointers between a$aaa and r$yyy are examined and
    adjusted if necessary.  the pointer kvrtn is also
    adjusted although it lies outside this range.
    dname is explicitly adjusted because the limits
    on dynamic region in stack are to the area actively
    in use (between dnamb and dnamp), and dname is outside
    this range.
relws   prc   e,0                 entry point
        mov   =a$aaa,wb           point to start of adjustables
        mov   =r$yyy,wa           point to end of adjustables
        jsr   relaj               relocate adjustable pointers
        add   rldya(xl),dname     adjust ptr missed by relaj
        mov   =kvrtn,wb           case of kvrtn
        mov   wb,wa               handled specially
        ica   wa                  one value to adjust
        jsr   relaj               adjust kvrtn
        exi                       return to caller
        enp                       end procedure relws
```

fi

```
    initialisation
    the following section receives control from the system
    at the start of a run with the registers set as follows.
    (wa)                  initial stack pointer
    (xr)                  points to first word of data area
    (xl)                  points to last word of data area
start  prc  e,0                                    entry point
       mov  wa,xs                                  discard return
       jsr  systm                                  initialise timer
```

---

if **.cnbt**

```
       sti  timsx                                  store time
       mov  xr,statb                               start address of static
```

else

```
    initialise work area (essential for batched runs)
       mov  xr,wb                                  preserve xr
       mov  =w$yyy,wa                              point to end of work area
       sub  =w$aaa,wa                              get length of work area
       btw  wa                                     convert to words
       lct  wa,wa                                  count for loop
       mov  =w$aaa,xr                              set up index register
    clear work space
ini01  zer  (xr)+                                  clear a word
       bct  wa,ini01                               loop till done
       mov  =stndo,wa                              undefined operators pointer
       mov  =r$yyy,wc                              point to table end
       sub  =r$uba,wc                              length of undef. operators table
       btw  wc                                     convert to words
       lct  wc,wc                                  loop counter
       mov  =r$uba,xr                              set up xr
    set correct value into undefined operators table
ini02  mov  wa,(xr)+                               store value
       bct  wc,ini02                               loop till all done
       mov  =num01,wa                              get a 1
```

---

if **.cpol**

```
       mov  wa,polcs                               interface polling interval
       mov  wa,polct                               interface polling interval
```

fi

```
       mov  wa,cmpsn                               statement no
       mov  wa,cswfl                               nofail
       mov  wa,cswls                               list
       mov  wa,kvinp                               input
       mov  wa,kvoup                               output
       mov  wa,lstpf                               nothing for listr yet
       mov  =iniln,wa                              input image length
       mov  wa,cswin                               -in72
```

```
        mov  =nulls,wa                              get null string pointer
        mov  wa,kvrtn                               return
        mov  wa,r$etx                               errtext
        mov  wa,r$ttl                               title for listing
        mov  wa,stxvr                               setexit
        sti  timsx                                  store time in correct place
        ldi  stlim                                  get default stlimit
        sti  kvstl                                  statement limit
        sti  kvstc                                  statement count
        mov  wb,statb                               store start adrs of static
fi
        mov  *e$srs,rsmem                           reserve memory
        mov  xs,stbas                               store stack base
        sss  iniss                                  save s-r stack ptr
    now convert free store percentage to a suitable factor
    for easy testing in alloc routine.
        ldi  intvh                                  get 100
        dvi  alfsp                                  form 100 / alfsp
        sti  alfsf                                  store the factor
```

```
if .csed
    now convert free sediment percentage to a suitable factor
    for easy testing in gbcol routine.
        ldi  intvh                                  get 100
        dvi  gbsdp                                  form 100 / gbsdp
        sti  gbsed                                  store the factor
fi
```

```
if .cnra
else
```

```
if .cncr
else
    initialize values for real conversion routine
        lct  wb,=cfp$s                              load counter for significant digits
        ldr  reav1                                  load 1.0
    loop to compute 10**(max number significant digits)
ini03   mlr  reavt                                  * 10.0
        bct  wb,ini03                               loop till done
        str  gtssc                                  store 10**(max sig digits)
        ldr  reap5                                  load 0.5
        dvr  gtssc                                  compute 0.5*10**(max sig digits)
        str  gtsrn                                  store as rounding bias
fi
fi
        zer  wc                                     set to read parameters
        jsr  prpar                                  read them
```

```
    now compute starting address for dynamic store and if
    necessary request more memory.
        sub   *e$srs,xl              allow for reserve memory
        mov   prlen,wa               get print buffer length
        add   =cfp$a,wa              add no. of chars in alphabet
        add   =nstmx,wa              add chars for gtstg bfr
        ctb   wa,8                   convert to bytes, allowing a margin
        mov   statb,xr               point to static base
        add   wa,xr                  increment for above buffers
        add   *e$hnb,xr              increment for hash table
        add   *e$sts,xr              bump for initial static block
        jsr   sysmx                  get mxlen
        mov   wa,kvmxl               provisionally store as maxlngth
        mov   wa,mxlen               and as mxlen
        bgt   xr,wa,ini06            skip if static hi exceeds mxlen
        ctb   wa,1                   round up and make bigger than mxlen
        mov   wa,xr                  use it instead
    here to store values which mark initial division
    of data area into static and dynamic
ini06   mov   xr,dnamb               dynamic base adrs
        mov   xr,dnamp               dynamic ptr
        bnz   wa,ini07               skip if non-zero mxlen
        dca   xr                     point a word in front
        mov   xr,kvmxl               use as maxlngth
        mov   xr,mxlen               and as mxlen
```

```
    loop here if necessary till enough memory obtained
    so that dname is above dnamb
ini07  mov  xl,dname                     store dynamic end address
       blt  dnamb,xl,ini09               skip if high enough
       jsr  sysmm                        request more memory
       wtb  xr                           get as baus (sgd05)
       add  xr,xl                        bump by amount obtained
       bnz  xr,ini07                     try again
```

---

if **.cera**

```
       mov  =mxern,wa                    insufficient memory for maxlength
       zer  wb                           no column number info
       zer  wc                           no line number info
       mov  =stgic,xr                    initial compile stage
```

---

if **.csfn**

```
       mov  =nulls,xl                    no file name
```

fi

```
       jsr  sysea                        advise of error
       ppm  ini08                        cant use error logic yet
       brn  ini08                        force termination
    insert text for error 329 in error message table
       erb  329,requested maxlngth       too large
```

fi

```
ini08  mov  =endmo,xr                    point to failure message
       mov  endml,wa                     message length
       jsr  syspr                        print it (prtst not yet usable)
       ppm                               should not fail
       zer  xl                           no fcb chain yet
       mov  =num10,wb                    set special code value
       jsr  sysej                        pack up (stopr not yet usable)
    initialise structures at start of static region
ini09  mov  statb,xr                     point to static again
       jsr  insta                        initialize static
    initialize number of hash headers
       mov  =e$hnb,wa                    get number of hash headers
       mti  wa                           convert to integer
       sti  hshnb                        store for use by gtnvr procedure
       lct  wa,wa                        counter for clearing hash table
       mov  xr,hshtb                     pointer to hash table
    loop to clear hash table
ini11  zer  (xr)+                        blank a word
       bct  wa,ini11                     loop
       mov  xr,hshte                     end of hash table adrs is kept
       mov  xr,state                     store static end address
```

---

if **.csfn**

```
    init table to map statement numbers to source file names
       mov  =num01,wc                    table will have only one bucket
       mov  =nulls,xl                    default table value
       mov  xl,r$sfc                     current source file name
       jsr  tmake                        create table
       mov  xr,r$sfn                     save ptr to table
```

fi

---

if **.cinc**

```
initialize table to detect duplicate include file names
    mov  =num01,wc          table will have only one bucket
    mov  =nulls,xl          default table value
    jsr  tmake              create table
    mov  xr,r$inc           save ptr to table
```

---

if **.csfn**

```
initialize array to hold names of nested include files
    mov  =ccinm,wa          maximum nesting level
    mov  =nulls,xl          null string default value
    jsr  vmake              create array
    ppm  vmake              create array
    mov  xr,r$ifa           save ptr to array
init array to hold line numbers of nested include files
    mov  =ccinm,wa          maximum nesting level
    mov  =inton,xl          integer one default value
    jsr  vmake              create array
    ppm  vmake              create array
    mov  xr,r$ifl           save ptr to array
```

fi

fi

```
initialize variable blocks for input and output
    mov  =v$inp,xl          point to string /input/
    mov  =trtin,wb          trblk type for input
    jsr  inout              perform input association
    mov  =v$oup,xl          point to string /output/
    mov  =trtou,wb          trblk type for output
    jsr  inout              perform output association
    mov  initr,wc           terminal flag
    bze  wc,ini13           skip if no terminal
    jsr  prpar              associate terminal
```

```
    check for expiry date
ini13   jsr   sysdc                 call date check
        mov   xs,flptr              in case stack overflows in compiler
    now compile source input code
        jsr   cmpil                 call compiler
        mov   xr,r$cod              set ptr to first code block
        mov   =nulls,r$ttl          forget title
        mov   =nulls,r$stl          forget sub-title
        zer   r$cim                 forget compiler input image
        zer   r$ccb                 forget interim code block
```

---

if **.cinc**

```
        zer   cnind                 in case end occurred with include
        zer   lstid                 listing include depth
```

fi

```
        zer   xl                    clear dud value
        zer   wb                    dont shift dynamic store up
```

---

if **.csed**

```
        zer   dnams                 collect sediment too
        jsr   gbcol                 clear garbage left from compile
        mov   xr,dnams              record new sediment size
```

else

```
        jsr   gbcol                 clear garbage left from compile
```

fi

```
        bnz   cpsts,inix0           skip if no listing of comp stats
        jsr   prtpg                 eject page
    print compile statistics
        jsr   prtmm                 print memory usage
        mti   cmerc                 get count of errors as integer
        mov   =encm3,xr             point to /compile errors/
        jsr   prtmi                 print it
        mti   gbcnt                 garbage collection count
        sbi   intv1                 adjust for unavoidable collect
        mov   =stpm5,xr             point to /storage regenerations/
        jsr   prtmi                 print gbcol count
        jsr   systm                 get time
        sbi   timsx                 get compilation time
        mov   =encm4,xr             point to compilation time (msec)/
        jsr   prtmi                 print message
        add   =num05,lstlc          bump line count
```

---

if **.cuej**

```
        bze   headp,inix0           no eject if nothing printed
        jsr   prtpg                 eject printer
```

fi

```
    prepare now to start execution
    set default input record length
inix0  bgt   cswin,=iniln,inix1          skip if not default -in72 used
       mov   =inils,cswin                else use default record length
    reset timer
inix1  jsr   systm                       get time again
       sti   timsx                       store for end run processing
       zer   gbcnt                       initialise collect count
       jsr   sysbx                       call before starting execution
       add   cswex,noxeq                 add -noexecute flag
       bnz   noxeq,inix2                 jump if execution suppressed
```

---

if **.cuej**
else

```
       bze   headp,iniy0                 no eject if nothing printed (sgd11)
       jsr   prtpg                       eject printer
```

fi

```
    merge when listing file set for execution.  also
    merge here when restarting a save file or load module.
iniy0  mnz   headp                       mark headers out regardless
       zer   -(xs)                       set failure location on stack
       mov   xs,flptr                    save ptr to failure offset word
       mov   r$cod,xr                    load ptr to entry code block
       mov   =stgxt,stage                set stage for execute time
```

---

if **.cpol**

```
       mov   =num01,polcs                reset interface polling interval
       mov   =num01,polct                reset interface polling interval
```

fi

---

if **.cnpf**
else

```
       mov   cmpsn,pfnte                 copy stmts compiled count in case
       mov   kvpfl,pfdmp                 start profiling if &profile set
       jsr   systm                       time yet again
       sti   systm                       time yet again
```

fi

```
       jsr   stgcc                       compute stmgo countdown counters
       bri   (xr)                        start xeq with first statement
    here if execution is suppressed
```

---

if **.cera**

```
inix2  zer   wa                          set abend value to zero
```

else

```
inix2  jsr   prtnl                       print a blank line
       mov   =encm5,xr                   point to /execution suppressed/
       jsr   prtst                       print string
       jsr   prtnl                       output line
       zer   wa                          set abend value to zero
```

fi

```
       mov   =nini9,wb                   set special code value
       zer   xl                          no fcb chain
       jsr   sysej                       end of job, exit to system
       enp                               end procedure start
    here from osint to restart a save file or load module.
```

```
rstrt   prc   e,0              entry point
        mov   stbas,xs         discard return
        zer   xl               clear xl
        brn   iniy0            resume execution
        enp                    end procedure rstrt
```

```
this section includes all routines which can be accessed
directly from the generated code except system functions.
all routines in this section start with a label of the
form o$xxx where xxx is three letters. the generated code
contains a pointer to the appropriate entry label.
since the general form of the generated code consists of
pointers to blocks whose first word is the address of the
actual entry point label (o$xxx).
these routines are in alphabetical order by their
entry label names (i.e. by the xxx of the o$xxx name)
these routines receive control as follows
(cp)                 pointer to next code word
(xs)                 current stack pointer
```

```
    binary plus (addition)
o$add   ent                                 entry point
        jsr   arith                         fetch arithmetic operands
        err   001,addition left             operand is not numeric
        err   002,addition right            operand is not numeric
```

---

if **.cnra**

else

```
        ppm   oadd1                         jump if real operands
```

fi

```
    here to add two integers
        adi   icval(xl)                     add right operand to left
        ino   exint                         return integer if no overflow
        erb   003,addition caused           integer overflow
```

---

if **.cnra**

else

```
    here to add two reals
oadd1   adr   rcval(xl)                     add right operand to left
        rno   exrea                         return real if no overflow
        erb   261,addition caused           real overflow
```

fi

```
   unary plus (affirmation)
o$aff   ent                             entry point
        mov  (xs)+,xr                   load operand
        jsr  gtnum                      convert to numeric
        err  004,affirmation operand    is not numeric
        mov  xr,-(xs)                   result if converted to numeric
        lcw  xr                         get next code word
        bri  (xr)                       execute it
```

```
    binary bar (alternation)
o$alt   ent                                  entry point
        mov  (xs)+,xr                        load right operand
        jsr  gtpat                           convert to pattern
        err  005,alternation right           operand is not pattern
    merge here from special (left alternation) case
oalt1   mov  =p$alt,wb                       set pcode for alternative node
        jsr  pbild                           build alternative node
        mov  xr,xl                           save address of alternative node
        mov  (xs)+,xr                        load left operand
        jsr  gtpat                           convert to pattern
        err  006,alternation left            operand is not pattern
        beq  xr,=p$alt,oalt2                 jump if left arg is alternation
        mov  xr,pthen(xl)                    set left operand as successor
        mov  xl,-(xs)                        stack result
        lcw  xr                              get next code word
        bri  (xr)                            execute it
    come here if left argument is itself an alternation
    the result is more efficient if we make the replacement
    (a / b) / c = a / (b / c)
oalt2   mov  parm1(xr),pthen(xl)             build the (b / c) node
        mov  pthen(xr),-(xs)                 set a as new left arg
        mov  xl,xr                           set (b / c) as new right arg
        brn  oalt1                           merge back to build a / (b / c)
```

```
    array reference (multiple subscripts, by name)
o$amn   ent                     entry point
        lcw   xr                load number of subscripts
        mov   xr,wb             set flag for by name
        brn   arref             jump to array reference routine
```

```
    array reference (multiple subscripts, by value)
o$amv   ent                  entry point
        lcw   xr             load number of subscripts
        zer   wb             set flag for by value
        brn   arref          jump to array reference routine
```

```
    array reference (one subscript, by name)
o$aon   ent                                      entry point
        mov  (xs),xr                             load subscript value
        mov  num01(xs),xl                        load array value
        mov  (xl),wa                             load first word of array operand
        beq  wa,=b$vct,oaon2                     jump if vector reference
        beq  wa,=b$tbt,oaon3                     jump if table reference
    here to use central array reference routine
oaon1   mov  =num01,xr                           set number of subscripts to one
        mov  xr,wb                               set flag for by name
        brn  arref                               jump to array reference routine
    here if we have a vector reference
oaon2   bne  (xr),=b$icl,oaon1                   use long routine if not integer
        ldi  icval(xr)                           load integer subscript value
        mfi  wa,exfal                            copy as address int, fail if ovflo
        bze  wa,exfal                            fail if zero
        add  =vcvlb,wa                           compute offset in words
        wtb  wa                                  convert to bytes
        mov  wa,(xs)                             complete name on stack
        blt  wa,vclen(xl),oaon4                  exit if subscript not too large
        brn  exfal                               else fail
    here for table reference
oaon3   mnz  wb                                  set flag for name reference
        jsr  tfind                               locate/create table element
        ppm  exfal                               fail if access fails
        mov  xl,num01(xs)                        store name base on stack
        mov  wa,(xs)                             store name offset on stack
    here to exit with result on stack
oaon4   lcw  xr                                  result on stack, get code word
        bri  (xr)                                execute next code word
```

```
    array reference (one subscript, by value)
o$aov   ent                                   entry point
        mov  (xs)+,xr                         load subscript value
        mov  (xs)+,xl                         load array value
        mov  (xl),wa                          load first word of array operand
        beq  wa,=b$vct,oaov2                  jump if vector reference
        beq  wa,=b$tbt,oaov3                  jump if table reference
    here to use central array reference routine
oaov1   mov  xl,-(xs)                         restack array value
        mov  xr,-(xs)                         restack subscript
        mov  =num01,xr                        set number of subscripts to one
        zer  wb                               set flag for value call
        brn  arref                            jump to array reference routine
    here if we have a vector reference
oaov2   bne  (xr),=b$icl,oaov1                use long routine if not integer
        ldi  icval(xr)                        load integer subscript value
        mfi  wa,exfal                         move as one word int, fail if ovflo
        bze  wa,exfal                         fail if zero
        add  =vcvlb,wa                        compute offset in words
        wtb  wa                               convert to bytes
        bge  wa,vclen(xl),exfal               fail if subscript too large
        jsr  acess                            access value
        ppm  exfal                            fail if access fails
        mov  xr,-(xs)                         stack result
        lcw  xr                               get next code word
        bri  (xr)                             execute it
    here for table reference by value
oaov3   zer  wb                               set flag for value reference
        jsr  tfind                            call table search routine
        ppm  exfal                            fail if access fails
        mov  xr,-(xs)                         stack result
        lcw  xr                               get next code word
        bri  (xr)                             execute it
```

```
    assignment
o$ass   ent                                   entry point
    o$rpl (pattern replacement) merges here
oass0   mov  (xs)+,wb                          load value to be assigned
        mov  (xs)+,wa                          load name offset
        mov  (xs),xl                           load name base
        mov  wb,(xs)                           store assigned value as result
        jsr  asign                             perform assignment
        ppm  exfal                             fail if assignment fails
        lcw  xr                                result on stack, get code word
        bri  (xr)                              execute next code word
```

```
    compilation error
o$cer   ent                                  entry point
        erb   007,compilation error          encountered during execution
```

```
    unary at (cursor assignment)
o$cas   ent                  entry point
        mov  (xs)+,wc        load name offset (parm2)
        mov  (xs)+,xr        load name base (parm1)
        mov  =p$cas,wb       set pcode for cursor assignment
        jsr  pbild           build node
        mov  xr,-(xs)        stack result
        lcw  xr              get next code word
        bri  (xr)            execute it
```

```
    concatenation
o$cnc   ent                         entry point
        mov  (xs),xr                load right argument
        beq  xr,=nulls,ocnc3        jump if right arg is null
        mov  1(xs),xl               load left argument
        beq  xl,=nulls,ocnc4        jump if left argument is null
        mov  =b$scl,wa              get constant to test for string
        bne  wa,(xl),ocnc2          jump if left arg not a string
        bne  wa,(xr),ocnc2          jump if right arg not a string
    merge here to concatenate two strings
ocnc1   mov  sclen(xl),wa           load left argument length
        add  sclen(xr),wa           compute result length
        jsr  alocs                  allocate scblk for result
        mov  xr,1(xs)               store result ptr over left argument
        psc  xr                     prepare to store chars of result
        mov  sclen(xl),wa           get number of chars in left arg
        plc  xl                     prepare to load left arg chars
        mvc                         move characters of left argument
        mov  (xs)+,xl               load right arg pointer, pop stack
        mov  sclen(xl),wa           load number of chars in right arg
        plc  xl                     prepare to load right arg chars
        mvc                         move characters of right argument
        zer  xl                     clear garbage value in xl
        lcw  xr                     result on stack, get code word
        bri  (xr)                   execute next code word
    come here if arguments are not both strings
ocnc2   jsr  gtstg                  convert right arg to string
        ppm  ocnc5                  jump if right arg is not string
        mov  xr,xl                  save right arg ptr
        jsr  gtstg                  convert left arg to string
        ppm  ocnc6                  jump if left arg is not a string
        mov  xr,-(xs)               stack left argument
        mov  xl,-(xs)               stack right argument
        mov  xr,xl                  move left arg to proper reg
        mov  (xs),xr                move right arg to proper reg
        brn  ocnc1                  merge back to concatenate strings
```

```
    concatenation (continued)
    come here for null right argument
ocnc3   ica   xs                        remove right arg from stack
        lcw   xr                        left argument on stack
        bri   (xr)                      execute next code word
    here for null left argument
ocnc4   ica   xs                        unstack one argument
        mov   xr,(xs)                   store right argument
        lcw   xr                        result on stack, get code word
        bri   (xr)                      execute next code word
    here if right argument is not a string
ocnc5   mov   xr,xl                     move right argument ptr
        mov   (xs)+,xr                  load left arg pointer
    merge here when left argument is not a string
ocnc6   jsr   gtpat                     convert left arg to pattern
        err   008,concatenation         left operand is not a string or pattern
        mov   xr,-(xs)                  save result on stack
        mov   xl,xr                     point to right operand
        jsr   gtpat                     convert to pattern
        err   009,concatenation         right operand is not a string or pattern
        mov   xr,xl                     move for pconc
        mov   (xs)+,xr                  reload left operand ptr
        jsr   pconc                     concatenate patterns
        mov   xr,-(xs)                  stack result
        lcw   xr                        get next code word
        bri   (xr)                      execute it
```

```
    complementation
o$com   ent                                  entry point
        mov  (xs)+,xr                        load operand
        mov  (xr),wa                         load type word
    merge back here after conversion
ocom1   beq  wa,=b$icl,ocom2                 jump if integer
```

---

if **.cnra**
else

```
        beq  wa,=b$rcl,ocom3                 jump if real
```

fi

```
        jsr  gtnum                           else convert to numeric
        err  010,negation operand            is not numeric
        brn  ocom1                           back to check cases
    here to complement integer
ocom2   ldi  icval(xr)                       load integer value
        ngi                                  negate
        ino  exint                           return integer if no overflow
        erb  011,negation caused             integer overflow
```

---

if **.cnra**
else

```
    here to complement real
ocom3   ldr  rcval(xr)                       load real value
        ngr                                  negate
        brn  exrea                           return real result
```

fi

```
    binary slash (division)
o$dvd   ent                                 entry point
        jsr   arith                         fetch arithmetic operands
        err   012,division left             operand is not numeric
        err   013,division right            operand is not numeric
```

---

if **.cnra**

else

```
        ppm   odvd2                         jump if real operands
```

fi

```
    here to divide two integers
        dvi   icval(xl)                     divide left operand by right
        ino   exint                         result ok if no overflow
        erb   014,division caused           integer overflow
```

---

if **.cnra**

else

```
    here to divide two reals
odvd2   dvr   rcval(xl)                     divide left operand by right
        rno   exrea                         return real if no overflow
        erb   262,division caused           real overflow
```

fi

```
    exponentiation
o$exp   ent                                   entry point
        mov  (xs)+,xr                         load exponent
        jsr  gtnum                            convert to number
        err  015,exponentiation               right operand is not numeric
        mov  xr,xl                            move exponent to xl
        mov  (xs)+,xr                         load base
        jsr  gtnum                            convert to numeric
        err  016,exponentiation               left operand is not numeric
```

---

if **.cnra**
else

```
        beq  (xl),=b$rcl,oexp7                jump if real exponent
```

fi

```
        ldi  icval(xl)                        load exponent
        ilt  oex12                            jump if negative exponent
```

---

if **.cnra**
else

```
        beq  wa,=b$rcl,oexp3                  jump if base is real
```

fi

```
    here to exponentiate an integer base and integer exponent
        mfi  wa,oexp2                         convert exponent to 1 word integer
        lct  wa,wa                            set loop counter
        ldi  icval(xr)                        load base as initial value
        bnz  wa,oexp1                         jump into loop if non-zero exponent
        ieq  oexp4                            error if 0**0
        ldi  intv1                            nonzero**0
        brn  exint                            give one as result for nonzero**0
    loop to perform exponentiation
oex13   mli  icval(xr)                        multiply by base
        iov  oexp2                            jump if overflow
oexp1   bct  wa,oex13                         loop if more to go
        brn  exint                            else return integer result
    here if integer overflow
oexp2   erb  017,exponentiation               caused integer overflow
```

```
    exponentiation (continued)
```
---
if **.cnra**

else

```
    here to exponentiate a real to an integer power
oexp3  mfi  wa,oexp6               convert exponent to one word
       lct  wa,wa                  set loop counter
       ldr  rcval(xr)              load base as initial value
       bnz  wa,oexp5               jump into loop if non-zero exponent
       req  oexp4                  error if 0.0**0
       ldr  reav1                  nonzero**0
       brn  exrea                  return 1.0 if nonzero**zero
```
fi

```
    here for error of 0**0 or 0.0**0
oexp4  erb  018,exponentiation     result is undefined
```
---
if **.cnra**

else

```
    loop to perform exponentiation
oex14  mlr  rcval(xr)              multiply by base
       rov  oexp6                  jump if overflow
oexp5  bct  wa,oex14               loop till computation complete
       brn  exrea                  then return real result
    here if real overflow
oexp6  erb  266,exponentiation     caused real overflow
    here with real exponent in (xl), numeric base in (xr)
```
---
if **.cmth**

```
oexp7  beq  (xr),=b$rcl,oexp8      jump if base real
       ldi  icval(xr)              load integer base
       itr                         convert to real
       jsr  rcbld                  create real in (xr)
    here with real exponent in (xl)
    numeric base in (xr) and ra
oexp8  zer  wb                     set positive result flag
       ldr  rcval(xr)              load base to ra
       rne  oexp9                  jump if base non-zero
       ldr  rcval(xl)              base is zero. check exponent
       req  oexp4                  jump if 0.0 ** 0.0
       ldr  reav0                  0.0 to non-zero exponent yields 0.0
       brn  exrea                  return zero result
    here with non-zero base in (xr) and ra, exponent in (xl)
    a negative base is allowed if the exponent is integral.
oexp9  rgt  oex10                  jump if base gt 0.0
       ngr                         make base positive
       jsr  rcbld                  create positive base in (xr)
       ldr  rcval(xl)              examine exponent
       chp                         chop to integral value
       rti  oexp6                  convert to integer, br if too large
       sbr  rcval(xl)              chop(exponent) - exponent
       rne  oex11                  non-integral power with neg base
       mfi  wb                     record even/odd exponent
       anb  bits1,wb               odd exponent yields negative result
       ldr  rcval(xr)              restore base to ra
    here with positive base in ra and (xr), exponent in (xl)
```

```
oex10  lnf                                       log of base
       rov   oexp6                               too large
       mlr   rcval(xl)                           times exponent
       rov   oexp6                               too large
       etx                                       e ** (exponent * ln(base))
       rov   oexp6                               too large
       bze   wb,exrea                            if no sign fixup required
       ngr                                       negative result needed
       brn                                       negative result needed
   here for non-integral exponent with negative base
oex11  erb   311,exponentiation                  of negative base to non-integral power
else
oexp7  erb   267,exponentiation                  right operand is real not integer
fi
fi
   here with negative integer exponent in ia
```

```
if .cmth
oex12  mov   xr,-(xs)                            stack base
       itr                                       convert to real exponent
       jsr   rcbld                               real negative exponent in (xr)
       mov   xr,xl                               put exponent in xl
       mov   (xs)+,xr                            restore base value
       brn   oexp7                               process real exponent
else
oex12  erb   019,exponentiation                  right operand is negative
fi
```

```
    failure in expression evaluation
    this entry point is used if the evaluation of an
    expression, initiated by the evalx procedure, fails.
    control is returned to an appropriate point in evalx.
o$fex   ent                                   entry point
        brn   evlx6                           jump to failure loc in evalx
```

```
    failure during evaluation of a complex or direct goto
o$fif   ent                                         entry point
        erb   020,goto evaluation                   failure
```

```
    function call (more than one argument)
o$fnc   ent                         entry point
        lcw   wa                    load number of arguments
        lcw   xr                    load function vrblk pointer
        mov   vrfnc(xr),xl          load function pointer
        bne   wa,fargs(xl),cfunc    use central routine if wrong num
        bri   (xl)                  jump to function if arg count ok
```

```
    function name error
o$fne   ent                                  entry point
        lcw   wa                             get next code word
        bne   wa,=ornm$,ofne1                fail if not evaluating expression
        bze   num02(xs),evlx3                ok if expr. was wanted by value
    here for error
ofne1   erb   021,function called            by name returned a value
```

```
    function call (single argument)
o$fns   ent                                   entry point
        lcw   xr                              load function vrblk pointer
        mov   =num01,wa                       set number of arguments to one
        mov   vrfnc(xr),xl                    load function pointer
        bne   wa,fargs(xl),cfunc              use central routine if wrong num
        bri   (xl)                            jump to function if arg count ok
```

```
    call to undefined function
o$fun   ent                                  entry point
        erb   022,undefined function         called
```

```
    execute complex goto
o$goc   ent                                     entry point
        mov  num01(xs),xr                       load name base pointer
        bhi  xr,state,ogoc1                     jump if not natural variable
        add  *vrtra,xr                          else point to vrtra field
        bri  (xr)                               and jump through it
    here if goto operand is not natural variable
ogoc1   erb  023,goto operand                   is not a natural variable
```

```
    execute direct goto
o$god   ent                                   entry point
        mov  (xs),xr                          load operand
        mov  (xr),wa                          load first word
        beq  wa,=b$cds,bcds0                  jump if code block to code routine
        beq  wa,=b$cdc,bcdc0                  jump if code block to code routine
        erb  024,goto operand                 in direct goto is not code
```

```
    set goto failure trap
    this routine is executed at the start of a complex or
    direct failure goto to trap a subsequent fail (see exfal)
o$gof   ent                         entry point
        mov  flptr,xr               point to fail offset on stack
        ica  (xr)                   point failure to o$fif word
        icp                         point to next code word
        lcw  xr                     fetch next code word
        bri  (xr)                   execute it
```

```
    binary dollar (immediate assignment)
    the pattern built by binary dollar is a compound pattern.
    see description at start of pattern match section for
    details of the structure which is constructed.
o$ima   ent                                 entry point
        mov  =p$imc,wb                      set pcode for last node
        mov  (xs)+,wc                       pop name offset (parm2)
        mov  (xs)+,xr                       pop name base (parm1)
        jsr  pbild                          build p$imc node
        mov  xr,xl                          save ptr to node
        mov  (xs),xr                        load left argument
        jsr  gtpat                          convert to pattern
        err  025,immediate assignment       left operand is not pattern
        mov  xr,(xs)                        save ptr to left operand pattern
        mov  =p$ima,wb                      set pcode for first node
        jsr  pbild                          build p$ima node
        mov  (xs)+,pthen(xr)                set left operand as p$ima successor
        jsr  pconc                          concatenate to form final pattern
        mov  xr,-(xs)                       stack result
        lcw  xr                             get next code word
        bri  (xr)                           execute it
```

```
    indirection (by name)
o$inn   ent                         entry point
        mnz  wb                     set flag for result by name
        brn  indir                  jump to common routine
```

```
    interrogation
o$int   ent                    entry point
        mov  =nulls,(xs)       replace operand with null
        lcw  xr                get next code word
        bri  (xr)              execute next code word
```

```
    indirection (by value)
o$inv   ent                 entry point
        zer   wb            set flag for by value
        brn   indir         jump to common routine
```

```
    keyword reference (by name)
o$kwn   ent                          entry point
        jsr   kwnam                  get keyword name
        brn   exnam                  exit with result name
```

```
    keyword reference (by value)
o$kwv   ent                          entry point
        jsr  kwnam                   get keyword name
        mov  xr,dnamp                delete kvblk
        jsr  acess                   access value
        ppm  exnul                   dummy (unused) failure return
        mov  xr,-(xs)                stack result
        lcw  xr                      get next code word
        bri  (xr)                    execute it
```

```
    load expression by name
o$lex   ent                          entry point
        mov  *evsi$,wa               set size of evblk
        jsr  alloc                   allocate space for evblk
        mov  =b$evt,(xr)             set type word
        mov  =trbev,evvar(xr)        set dummy trblk pointer
        lcw  wa                      load exblk pointer
        mov  wa,evexp(xr)            set exblk pointer
        mov  xr,xl                   move name base to proper reg
        mov  *evvar,wa               set name offset = zero
        brn  exnam                   exit with name in (xl,wa)
```

```
    load pattern value
o$lpt   ent                     entry point
        lcw   xr                load pattern pointer
        mov   xr,-(xs)          stack result
        lcw   xr                get next code word
        bri   (xr)              execute it
```

```
    load variable name
o$lvn  ent                    entry point
       lcw  wa                load vrblk pointer
       mov  wa,-(xs)          stack vrblk ptr (name base)
       mov  *vrval,-(xs)      stack name offset
       lcw  xr                get next code word
       bri  (xr)              execute next code word
```

```
    binary asterisk (multiplication)
o$mlt   ent                                  entry point
        jsr   arith                          fetch arithmetic operands
        err   026,multiplication             left operand is not numeric
        err   027,multiplication             right operand is not numeric
```

---

if **.cnra**
else

```
        ppm   omlt1                          jump if real operands
```

fi

```
    here to multiply two integers
        mli   icval(xl)                      multiply left operand by right
        ino   exint                          return integer if no overflow
        erb   028,multiplication             caused integer overflow
```

---

if **.cnra**
else

```
    here to multiply two reals
omlt1   mlr   rcval(xl)                      multiply left operand by right
        rno   exrea                          return real if no overflow
        erb   263,multiplication             caused real overflow
```

fi

```
   name reference
o$nam   ent                         entry point
        mov  *nmsi$,wa              set length of nmblk
        jsr  alloc                  allocate nmblk
        mov  =b$nml,(xr)            set name block code
        mov  (xs)+,nmofs(xr)        set name offset from operand
        mov  (xs)+,nmbas(xr)        set name base from operand
        mov  xr,-(xs)               stack result
        lcw  xr                     get next code word
        bri  (xr)                   execute it
```

```
    negation
    initial entry
o$nta   ent                                     entry point
        lcw   wa                                load new failure offset
        mov   flptr,-(xs)                       stack old failure pointer
        mov   wa,-(xs)                          stack new failure offset
        mov   xs,flptr                          set new failure pointer
        lcw   xr                                get next code word
        bri   (xr)                              execute next code word
    entry after successful evaluation of operand
o$ntb   ent                                     entry point
        mov   num02(xs),flptr                   restore old failure pointer
        brn   exfal                             and fail
    entry for failure during operand evaluation
o$ntc   ent                                     entry point
        ica   xs                                pop failure offset
        mov   (xs)+,flptr                       restore old failure pointer
        brn   exnul                             exit giving null result
```

```
    use of undefined operator
o$oun   ent                                     entry point
        erb   029,undefined operator            referenced
```

```
    binary dot (pattern assignment)
    the pattern built by binary dot is a compound pattern.
    see description at start of pattern match section for
    details of the structure which is constructed.
o$pas   ent                           entry point
        mov  =p$pac,wb                load pcode for p$pac node
        mov  (xs)+,wc                 load name offset (parm2)
        mov  (xs)+,xr                 load name base (parm1)
        jsr  pbild                    build p$pac node
        mov  xr,xl                    save ptr to node
        mov  (xs),xr                  load left operand
        jsr  gtpat                    convert to pattern
        err  030,pattern assignment   left operand is not pattern
        mov  xr,(xs)                  save ptr to left operand pattern
        mov  =p$paa,wb                set pcode for p$paa node
        jsr  pbild                    build p$paa node
        mov  (xs)+,pthen(xr)          set left operand as p$paa successor
        jsr  pconc                    concatenate to form final pattern
        mov  xr,-(xs)                 stack result
        lcw  xr                       get next code word
        bri  (xr)                     execute it
```

```
    pattern match (by name, for replacement)
o$pmn   ent                                        entry point
        zer   wb                                   set type code for match by name
        brn   match                                jump to routine to start match
```

```
    pattern match (statement)
    o$pms is used in place of o$pmv when the pattern match
    occurs at the outer (statement) level since in this
    case the substring value need not be constructed.
o$pms   ent                                 entry point
        mov  =num02,wb                      set flag for statement to match
        brn  match                          jump to routine to start match
```

```
    pattern match (by value)
o$pmv   ent                      entry point
        mov  =num01,wb           set type code for value match
        brn  match               jump to routine to start match
```

```
    pop top item on stack
o$pop   ent                        entry point
        ica    xs                  pop top stack entry
        lcw    xr                  get next code word
        bri    (xr)                execute next code word
```

```
    terminate execution (code compiled for end statement)
o$stp   ent                                     entry point
        brn   lend0                             jump to end circuit
```

```
    return name from expression
    this entry points is used if the evaluation of an
    expression, initiated by the evalx procedure, returns
    a name. control is returned to the proper point in evalx.
o$rnm   ent                                      entry point
        brn   evlx4                              return to evalx procedure
```

```
    pattern replacement
    when this routine gets control, the following stack
    entries have been made (see end of match routine p$nth)
                          subject name base
                          subject name offset
                          initial cursor value
                          final cursor value
                          subject string pointer
    (xs) ---------------- replacement value
o$rpl   ent                                      entry point
        jsr   gtstg                              convert replacement val to string
        err   031,pattern replacement            right operand is not a string
    get result length and allocate result scblk
        mov   (xs),xl                            load subject string pointer
```

---

if **.cnbf**
else

```
        beq   (xl),=b$bct,orpl4                  branch if buffer assignment
```

fi

```
        add   sclen(xl),wa                       add subject string length
        add   num02(xs),wa                       add starting cursor
        sub   num01(xs),wa                       minus final cursor = total length
        bze   wa,orpl3                           jump if result is null
        mov   xr,-(xs)                           restack replacement string
        jsr   alocs                              allocate scblk for result
        mov   num03(xs),wa                       get initial cursor (part 1 len)
        mov   xr,num03(xs)                       stack result pointer
        psc   xr                                 point to characters of result
    move part 1 (start of subject) to result
        bze   wa,orpl1                           jump if first part is null
        mov   num01(xs),xl                       else point to subject string
        plc   xl                                 point to subject string chars
        mvc                                      move first part to result
```

```
    pattern replacement (continued)
    now move in replacement value
orpl1   mov  (xs)+,xl              load replacement string, pop
        mov  sclen(xl),wa          load length
        bze  wa,orpl2              jump if null replacement
        plc  xl                    else point to chars of replacement
        mvc                        move in chars (part 2)
    now move in remainder of string (part 3)
orpl2   mov  (xs)+,xl              load subject string pointer, pop
        mov  (xs)+,wc              load final cursor, pop
        mov  sclen(xl),wa          load subject string length
        sub  wc,wa                 minus final cursor = part 3 length
        bze  wa,oass0              jump to assign if part 3 is null
        plc  xl,wc                 else point to last part of string
        mvc                        move part 3 to result
        brn  oass0                 jump to perform assignment
    here if result is null
orpl3   add  *num02,xs             pop subject str ptr, final cursor
        mov  =nulls,(xs)           set null result
        brn  oass0                 jump to assign null value
```

---

if **.cnbf**
else

```
    here for buffer substring assignment
orpl4   mov  xr,xl                 copy scblk replacement ptr
        mov  (xs)+,xr              unstack bcblk ptr
        mov  (xs)+,wb              get final cursor value
        mov  (xs)+,wa              get initial cursor
        sub  wa,wb                 get length in wb
        add  *num01,xs             get rid of name offset
        mov  xr,(xs)               store buffer result over name base
        jsr  insbf                 insert substring
        ppm                        convert fail impossible
        ppm  exfal                 fail if insert fails
        lcw  xr                    result on stack, get code word
        bri  (xr)                  execute next code word
```

fi

```
    return value from expression
    this entry points is used if the evaluation of an
    expression, initiated by the evalx procedure, returns
    a value. control is returned to the proper point in evalx
o$rvl   ent                                         entry point
        brn   evlx3                                 return to evalx procedure
```

```
    selection
    initial entry
o$sla   ent                 entry point
        lcw  wa             load new failure offset
        mov  flptr,-(xs)    stack old failure pointer
        mov  wa,-(xs)       stack new failure offset
        mov  xs,flptr       set new failure pointer
        lcw  xr             get next code word
        bri  (xr)           execute next code word
    entry after successful evaluation of alternative
o$slb   ent                 entry point
        mov  (xs)+,xr       load result
        ica  xs             pop fail offset
        mov  (xs),flptr     restore old failure pointer
        mov  xr,(xs)        restack result
        lcw  wa             load new code offset
        add  r$cod,wa       point to absolute code location
        lcp  wa             set new code pointer
        lcw  xr             get next code word
        bri  (xr)           execute next code word
    entry at start of subsequent alternatives
o$slc   ent                 entry point
        lcw  wa             load new fail offset
        mov  wa,(xs)        store new fail offset
        lcw  xr             get next code word
        bri  (xr)           execute next code word
    entry at start of last alternative
o$sld   ent                 entry point
        ica  xs             pop failure offset
        mov  (xs)+,flptr    restore old failure pointer
        lcw  xr             get next code word
        bri  (xr)           execute next code word
```

```
    binary minus (subtraction)
o$sub   ent                                  entry point
        jsr  arith                           fetch arithmetic operands
        err  032,subtraction left            operand is not numeric
        err  033,subtraction right           operand is not numeric
```

---

if **.cnra**
else

```
        ppm  osub1                           jump if real operands
```

fi

```
    here to subtract two integers
        sbi  icval(xl)                       subtract right operand from left
        ino  exint                           return integer if no overflow
        erb  034,subtraction caused          integer overflow
```

---

if **.cnra**
else

```
    here to subtract two reals
osub1   sbr  rcval(xl)                       subtract right operand from left
        rno  exrea                           return real if no overflow
        erb  264,subtraction caused          real overflow
```

fi

```
    dummy operator to return control to trxeq procedure
o$txr   ent                                     entry point
        brn   trxq1                             jump into trxeq procedure
```

```
    unexpected failure
    note that if a setexit trap is operating then
    transfer to system label continue
    will result in looping here.  difficult to avoid except
    with a considerable overhead which is not worthwhile or
    else by a technique such as setting kverl to zero.
o$unf   ent                                   entry point
        erb   035,unexpected failure          in -nofail mode
```

```
the first word of every block in dynamic storage and the
vrget, vrsto and vrtra fields of a vrblk contain a
pointer to an entry point in the program. all such entry
points are in the following section except those for
pattern blocks which are in the pattern matching segment
later on (labels of the form p$xxx), and dope vectors
(d$xxx) which are in the dope vector section following
the pattern routines (dope vectors are used for cmblks).
the entry points in this section have labels of the
form b$xxy where xx is the two character block type for
the corresponding block and y is any letter.
in some cases, the pointers serve no other purpose than
to identify the block type. in this case the routine
is never executed and thus no code is assembled.
for each of these entry points corresponding to a block
an entry point identification is assembled (bl$xx).
the exact entry conditions depend on the manner in
which the routine is accessed and are documented with
the individual routines as required.
the order of these routines is alphabetical with the
following exceptions.
the routines for seblk and exblk entries occur first so
that expressions can be quickly identified from the fact
that their routines lie before the symbol b$e$$.
these are immediately followed by the routine for a trblk
so that the test against the symbol b$t$$ checks for
trapped values or expression values (see procedure evalp)
the pattern routines lie after this section so that
patterns are identified with routines starting at or
after the initial instruction in these routines (p$aaa).
the symbol b$aaa defines the first location for block
routines and the symbol p$yyy (at the end of the pattern
match routines section) defines the last such entry point
```

b$aaa	**ent**	bl$$i	entry point of first block routine

```
    exblk
    the routine for an exblk loads the expression onto
    the stack as a value.
    (xr)                  pointer to exblk
b$exl   ent   bl$ex                       entry point (exblk)
        mov   xr,-(xs)                    stack result
        lcw   xr                          get next code word
        bri   (xr)                        execute it
```

```
    seblk
    the routine for seblk is accessed from the generated
    code to load the expression value onto the stack.
b$sel   ent   bl$se                                   entry point (seblk)
        mov   xr,-(xs)                                stack result
        lcw   xr                                      get next code word
        bri   (xr)                                    execute it
    define symbol which marks end of entries for expressions
b$e$$   ent   bl$$i                                   entry point
```

```
    trblk
    the routine for a trblk is never executed
b$trt   ent   bl$tr                                      entry point (trblk)
    define symbol marking end of trap and expression blocks
b$t$$   ent   bl$$i                                      end of trblk,seblk,exblk entries
```

```
    arblk
    the routine for arblk is never executed
b$art   ent   bl$ar                                  entry point (arblk)
```

```
    bcblk
    the routine for a bcblk is never executed
    (xr)                    pointer to bcblk
b$bct   ent   bl$bc                                  entry point (bcblk)
```

```
    bfblk
    the routine for a bfblk is never executed
    (xr)                  pointer to bfblk
b$bft   ent   bl$bf                                  entry point (bfblk)
```

```
    ccblk
    the routine for ccblk is never entered
b$cct   ent   bl$cc                          entry point (ccblk)
```

```
    cdblk
    the cdblk routines are executed from the generated code.
    there are two cases depending on the form of cdfal.
    entry for complex failure code at cdfal
    (xr)                  pointer to cdblk
b$cdc  ent  bl$cd                        entry point (cdblk)
bcdc0  mov  flptr,xs                     pop garbage off stack
       mov  cdfal(xr),(xs)               set failure offset
       brn  stmgo                        enter stmt
```

```
    cdblk (continued)
    entry for simple failure code at cdfal
    (xr)                  pointer to cdblk
b$cds   ent   bl$cd                               entry point (cdblk)
bcds0   mov   flptr,xs                            pop garbage off stack
        mov   *cdfal,(xs)                         set failure offset
        brn   stmgo                               enter stmt
```

```
    cmblk
    the routine for a cmblk is never executed
b$cmt   ent   bl$cm                                  entry point (cmblk)
```

```
    ctblk
    the routine for a ctblk is never executed
b$ctt   ent   bl$ct                                  entry point (ctblk)
```

```
    dfblk
    the routine for a dfblk is accessed from the o$fnc entry
    to call a datatype function and build a pdblk.
    (xl)                  pointer to dfblk
b$dfc   ent   bl$df                                    entry point
        mov   dfpdl(xl),wa                             load length of pdblk
        jsr   alloc                                    allocate pdblk
        mov   =b$pdt,(xr)                              store type word
        mov   xl,pddfp(xr)                             store dfblk pointer
        mov   xr,wc                                    save pointer to pdblk
        add   wa,xr                                    point past pdblk
        lct   wa,fargs(xl)                             set to count fields
    loop to acquire field values from stack
bdfc1   mov   (xs)+,-(xr)                              move a field value
        bct   wa,bdfc1                                 loop till all moved
        mov   wc,xr                                    recall pointer to pdblk
        brn   exsid                                    exit setting id field
```

```
    efblk
    the routine for an efblk is passed control form the o$fnc
    entry to call an external function.
    (xl)                  pointer to efblk
b$efc   ent   bl$ef                                  entry point (efblk)
```
---
if **.cnld**
else
```
        mov   fargs(xl),wc                           load number of arguments
        wtb   wc                                     convert to offset
        mov   xl,-(xs)                               save pointer to efblk
        mov   xs,xt                                  copy pointer to arguments
    loop to convert arguments
befc1   ica   xt                                     point to next entry
        mov   (xs),xr                                load pointer to efblk
        dca   wc                                     decrement eftar offset
        add   wc,xr                                  point to next eftar entry
        mov   eftar(xr),xr                           load eftar entry
```
---
if **.cnra**

---
if **.cnlf**
```
        bsw   xr,4                                   switch on type
```
else
```
        bsw   xr,3                                   switch on type
```
fi
else

---
if **.cnlf**
```
        bsw   xr,5                                   switch on type
```
else
```
        bsw   xr,4                                   switch on type
```
fi
fi
```
        iff   0,befc7                                no conversion needed
        iff   1,befc2                                string
        iff   2,befc3                                integer
```
---
if **.cnra**

---
if **.cnlf**
```
        iff   3,beff1                                file
```
fi
else
```
        iff   3,befc4                                real
```
---
if **.cnlf**
```
        iff   4,beff1                                file
```
fi
fi
```
        esw                                          end of switch on type
```
---
if **.cnlf**
```
    here to convert to file
beff1   mov   xt,-(xs)                               save entry pointer
        mov   wc,befof                               save offset
        mov   (xt),-(xs)                             stack arg pointer
        jsr   iofcb                                  convert to fcb
```

```
        err    298,external function          argument is not file
        err    298,external function          argument is not file
        err    298,external function          argument is not file
        mov    wa,xr                          point to fcb
        mov    (xs)+,xt                       reload entry pointer
        brn    befc5                          jump to merge
fi
    here to convert to string
befc2   mov    (xt),-(xs)                     stack arg ptr
        jsr    gtstg                          convert argument to string
        err    039,external function          argument is not a string
        brn    befc6                          jump to merge
```

```
    efblk (continued)
    here to convert an integer
befc3   mov  (xt),xr                       load next argument
        mov  wc,befof                      save offset
        jsr  gtint                         convert to integer
        err  040,external function         argument is not integer
```
---
if **.cnra**

else
```
        brn  befc5                         merge with real case
    here to convert a real
befc4   mov  (xt),xr                       load next argument
        mov  wc,befof                      save offset
        jsr  gtrea                         convert to real
        err  265,external function         argument is not real
```
fi
```
    integer case merges here
befc5   mov  befof,wc                      restore offset
    string merges here
befc6   mov  xr,(xt)                       store converted result
    no conversion merges here
befc7   bnz  wc,befc1                      loop back if more to go
    here after converting all the arguments
        mov  (xs)+,xl                      restore efblk pointer
        mov  fargs(xl),wa                  get number of args
        jsr  sysex                         call routine to call external fnc
        ppm  exfal                         fail if failure
        err  327,calling external          function - not found
        err  326,calling external          function - bad argument type
```
---
if **.cexp**
```
        wtb  wa                            convert number of args to bytes
        add  wa,xs                         remove arguments from stack
```
fi

```
    efblk (continued)
    return here with result in xr
    first defend against non-standard null string returned
        mov  efrsl(xl),wb                            get result type id
        bnz  wb,befa8                                branch if not unconverted
        bne  (xr),=b$scl,befc8                       jump if not a string
        bze  sclen(xr),exnul                         return null if null
    here if converted result to check for null string
befa8   bne  wb,=num01,befc8                         jump if not a string
        bze  sclen(xr),exnul                         return null if null
    return if result is in dynamic storage
befc8   blt  xr,dnamb,befc9                          jump if not in dynamic storage
        ble  xr,dnamp,exixr                          return result if already dynamic
    here we copy a result into the dynamic region
befc9   mov  (xr),wa                                 get possible type word
        bze  wb,bef11                                jump if unconverted result
        mov  =b$scl,wa                               string
        beq  wb,=num01,bef10                         yes jump
        mov  =b$icl,wa                               integer
        beq  wb,=num02,bef10                         yes jump
```

---

if **.cnra**
else

```
        mov  =b$rcl,wa                               real
```

fi

```
    store type word in result
bef10   mov  wa,(xr)                                 stored before copying to dynamic
    merge for unconverted result
bef11   beq  (xr),=b$scl,bef12                       branch if string result
        jsr  blkln                                   get length of block
        mov  xr,xl                                   copy address of old block
        jsr  alloc                                   allocate dynamic block same size
        mov  xr,-(xs)                                set pointer to new block as result
        mvw                                          copy old block to dynamic block
        zer  xl                                      clear garbage value
        lcw  xr                                      get next code word
        bri  (xr)                                    execute next code word
    here to return a string result that was not in dynamic.
    cannot use the simple word copy above because it will not
    guarantee zero padding in the last word.
bef12   mov  xr,xl                                   save source string pointer
        mov  sclen(xr),wa                            fetch string length
        bze  wa,exnul                                return null string if length zero
        jsr  alocs                                   allocate space for string
        mov  xr,-(xs)                                save as result pointer
        psc  xr                                      prepare to store chars of result
        plc  xl                                      point to chars in source string
        mov  wc,wa                                   number of characters to copy
        mvc                                          move characters to result string
        zer  xl                                      clear garbage value
        lcw  xr                                      get next code word
        bri  (xr)                                    execute next code word
```

fi

```
    evblk
    the routine for an evblk is never executed
b$evt   ent   bl$ev                              entry point (evblk)
```

```
    ffblk
    the routine for an ffblk is executed from the o$fnc entry
    to call a field function and extract a field value/name.
    (xl)                  pointer to ffblk
b$ffc   ent   bl$ff                          entry point (ffblk)
        mov   xl,xr                          copy ffblk pointer
        lcw   wc                             load next code word
        mov   (xs),xl                        load pdblk pointer
        bne   (xl),=b$pdt,bffc2              jump if not pdblk at all
        mov   pddfp(xl),wa                   load dfblk pointer from pdblk
    loop to find correct ffblk for this pdblk
bffc1   beq   wa,ffdfp(xr),bffc3             jump if this is the correct ffblk
        mov   ffnxt(xr),xr                   else link to next ffblk on chain
        bnz   xr,bffc1                       loop back if another entry to check
    here for bad argument
bffc2   erb   041,field function             argument is wrong datatype
```

```
    ffblk (continued)
    here after locating correct ffblk
bffc3   mov  ffofs(xr),wa           load field offset
        beq  wc,=ofne$,bffc5        jump if called by name
        add  wa,xl                  else point to value field
        mov  (xl),xr                load value
        bne  (xr),=b$trt,bffc4      jump if not trapped
        sub  wa,xl                  else restore name base,offset
        mov  wc,(xs)                save next code word over pdblk ptr
        jsr  acess                  access value
        ppm  exfal                  fail if access fails
        mov  (xs),wc                restore next code word
    here after getting value in (xr), xl is garbage
bffc4   mov  xr,(xs)                store value on stack (over pdblk)
        mov  wc,xr                  copy next code word
        mov  (xr),xl                load entry address
        bri  xl                     jump to routine for next code word
    here if called by name
bffc5   mov  wa,-(xs)               store name offset (base is set)
        lcw  xr                     get next code word
        bri  (xr)                   execute next code word
```

```
    icblk
    the routine for icblk is executed from the generated
    code to load an integer value onto the stack.
    (xr)                  pointer to icblk
b$icl   ent   bl$ic                    entry point (icblk)
        mov   xr,-(xs)                 stack result
        lcw   xr                       get next code word
        bri   (xr)                     execute it
```

```
    kvblk
    the routine for a kvblk is never executed.
b$kvt   ent   bl$kv                              entry point (kvblk)
```

```
nmblk
the routine for a nmblk is executed from the generated
code for the case of loading a name onto the stack
where the name is that of a natural variable which can
be preevaluated at compile time.
(xr)                 pointer to nmblk
```

```
b$nml   ent   bl$nm            entry point (nmblk)
        mov   xr,-(xs)         stack result
        lcw   xr               get next code word
        bri   (xr)             execute it
```

```
    pdblk
    the routine for a pdblk is never executed
b$pdt   ent   bl$pd                          entry point (pdblk)
```

```
    pfblk
    the routine for a pfblk is executed from the entry o$fnc
    to call a program defined function.
    (xl)                  pointer to pfblk
    the following stack entries are made before passing
    control to the program defined function.
                          saved value of first argument
                          .
                          saved value of last argument
                          saved value of first local
                          .
                          saved value of last local
                          saved value of function name
                          saved code block ptr (r$cod)
                          saved code pointer (-r$cod)
                          saved value of flprt
                          saved value of flptr
                          pointer to pfblk
    flptr --------------- zero (to be overwritten with offs)
b$pfc   ent   bl$pf                                   entry point (pfblk)
        mov   xl,bpfpf                                save pfblk ptr (need not be reloc)
        mov   xl,xr                                   copy for the moment
        mov   pfvbl(xr),xl                            point to vrblk for function
    loop to find old value of function
bpf01   mov   xl,wb                                   save pointer
        mov   vrval(xl),xl                            load value
        beq   (xl),=b$trt,bpf01                       loop if trblk
    set value to null and save old function value
        mov   xl,bpfsv                                save old value
        mov   wb,xl                                   point back to block with value
        mov   =nulls,vrval(xl)                        set value to null
        mov   fargs(xr),wa                            load number of arguments
        add   *pfarg,xr                               point to pfarg entries
        bze   wa,bpf04                                jump if no arguments
        mov   xs,xt                                   ptr to last arg
        wtb   wa                                      convert no. of args to bytes offset
        add   wa,xt                                   point before first arg
        mov   xt,bpfxt                                remember arg pointer
```

```
    pfblk (continued)
    loop to save old argument values and set new ones
bpf02   mov  (xr)+,xl                    load vrblk ptr for next argument
    loop through possible trblk chain to find value
bpf03   mov  xl,wc                       save pointer
        mov  vrval(xl),xl                load next value
        beq  (xl),=b$trt,bpf03           loop back if trblk
    save old value and get new value
        mov  xl,wa                       keep old value
        mov  bpfxt,xt                    point before next stacked arg
        mov  -(xt),wb                    load argument (new value)
        mov  wa,(xt)                     save old value
        mov  xt,bpfxt                    keep arg ptr for next time
        mov  wc,xl                       point back to block with value
        mov  wb,vrval(xl)                set new value
        bne  xs,bpfxt,bpf02              loop if not all done
    now process locals
bpf04   mov  bpfpf,xl                    restore pfblk pointer
        mov  pfnlo(xl),wa                load number of locals
        bze  wa,bpf07                    jump if no locals
        mov  =nulls,wb                   get null constant
        lct  wa,wa                       set local counter
    loop to process locals
bpf05   mov  (xr)+,xl                    load vrblk ptr for next local
    loop through possible trblk chain to find value
bpf06   mov  xl,wc                       save pointer
        mov  vrval(xl),xl                load next value
        beq  (xl),=b$trt,bpf06           loop back if trblk
    save old value and set null as new value
        mov  xl,-(xs)                    stack old value
        mov  wc,xl                       point back to block with value
        mov  wb,vrval(xl)                set null as new value
        bct  wa,bpf05                    loop till all locals processed
```

```
    pfblk (continued)
    here after processing arguments and locals
```

if **.cnpf**

```
bpf07   mov  r$cod,wa              load old code block pointer
```

else

```
bpf07   zer  xr                    zero reg xr in case
        bze  kvpfl,bpf7c           skip if profiling is off
        beq  kvpfl,=num02,bpf7a    branch on type of profile
    here if &profile = 1
        jsr  systm                 get current time
        sti  pfetm                 save for a sec
        sbi  pfstm                 find time used by caller
        jsr  icbld                 build into an icblk
        ldi  pfetm                 reload current time
        brn  bpf7b                 merge
     here if &profile = 2
bpf7a   ldi  pfstm                 get start time of calling stmt
        jsr  icbld                 assemble an icblk round it
        jsr  systm                 get now time
    both types of profile merge here
bpf7b   sti  pfstm                 set start time of 1st func stmt
        mnz  pffnc                 flag function entry
    no profiling merges here
bpf7c   mov  xr,-(xs)              stack icblk ptr (or zero)
        mov  r$cod,wa              load old code block pointer
```

fi

```
        scp  wb                    get code pointer
        sub  wa,wb                 make code pointer into offset
        mov  bpfpf,xl              recall pfblk pointer
        mov  bpfsv,-(xs)           stack old value of function name
        mov  wa,-(xs)              stack code block pointer
        mov  wb,-(xs)              stack code offset
        mov  flprt,-(xs)           stack old flprt
        mov  flptr,-(xs)           stack old failure pointer
        mov  xl,-(xs)              stack pointer to pfblk
        zer  -(xs)                 dummy zero entry for fail return
        chk                        check for stack overflow
        mov  xs,flptr              set new fail return value
        mov  xs,flprt              set new flprt
        mov  kvtra,wa              load trace value
        add  kvftr,wa              add ftrace value
        bnz  wa,bpf09              jump if tracing possible
        icv  kvfnc                 else bump fnclevel
    here to actually jump to function
bpf08   mov  pfcod(xl),xr          point to vrblk of entry label
        mov  vrlbl(xr),xr          point to target code
        beq  xr,=stndl,bpf17       test for undefined label
        bne  (xr),=b$trt,bpf8a     jump if not trapped
        mov  trlbl(xr),xr          else load ptr to real label code
bpf8a   bri  (xr)                  off to execute function
    here if tracing is possible
bpf09   mov  pfctr(xl),xr          load possible call trace trblk
        mov  pfvbl(xl),xl          load vrblk pointer for function
```

```
    mov  *vrval,wa             set name offset for variable
    bze  kvtra,bpf10           jump if trace mode is off
    bze  xr,bpf10              or if there is no call trace
here if call traced
    dcv  kvtra                 decrement trace count
    bze  trfnc(xr),bpf11       jump if print trace
    jsr  trxeq                 execute function type trace
```

```
    pfblk (continued)
    here to test for ftrace trace
bpf10  bze  kvftr,bpf16            jump if ftrace is off
       dcv  kvftr                  else decrement ftrace
    here for print trace
bpf11  jsr  prtsn                  print statement number
       jsr  prtnm                  print function name
       mov  =ch$pp,wa              load left paren
       jsr  prtch                  print left paren
       mov  num01(xs),xl           recover pfblk pointer
       bze  fargs(xl),bpf15        skip if no arguments
       zer  wb                     else set argument counter
       brn  bpf13                  jump into loop
    loop to print argument values
bpf12  mov  =ch$cm,wa              load comma
       jsr  prtch                  print to separate from last arg
    merge here first time (no comma required)
bpf13  mov  wb,(xs)                save arg ctr (over failoffs is ok)
       wtb  wb                     convert to byte offset
       add  wb,xl                  point to next argument pointer
       mov  pfarg(xl),xr           load next argument vrblk ptr
       sub  wb,xl                  restore pfblk pointer
       mov  vrval(xr),xr           load next value
       jsr  prtvl                  print argument value
```

```
    here after dealing with one argument
         mov  (xs),wb                          restore argument counter
         icv  wb                               increment argument counter
         blt  wb,fargs(xl),bpf12               loop if more to print
    merge here in no args case to print paren
bpf15    mov  =ch$rp,wa                        load right paren
         jsr  prtch                            print to terminate output
         jsr  prtnl                            terminate print line
    merge here to exit with test for fnclevel trace
bpf16    icv  kvfnc                            increment fnclevel
         mov  r$fnc,xl                         load ptr to possible trblk
         jsr  ktrex                            call keyword trace routine
    call function after trace tests complete
         mov  num01(xs),xl                     restore pfblk pointer
         brn  bpf08                            jump back to execute function
    here if calling a function whose entry label is undefined
bpf17    mov  num02(xs),flptr                  reset so exfal can return to evalx
         erb  286,function call                to undefined entry label
```

---

if **.cnra**
else

```
    rcblk
    the routine for an rcblk is executed from the generated
    code to load a real value onto the stack.
    (xr)                  pointer to rcblk
b$rcl   ent   bl$rc                                      entry point (rcblk)
        mov   xr,-(xs)                                   stack result
        lcw   xr                                         get next code word
        bri   (xr)                                       execute it
fi
```

```
    scblk
    the routine for an scblk is executed from the generated
    code to load a string value onto the stack.
    (xr)                  pointer to scblk
b$scl   ent   bl$sc                                     entry point (scblk)
        mov   xr,-(xs)                                  stack result
        lcw   xr                                        get next code word
        bri   (xr)                                      execute it
```

```
    tbblk
    the routine for a tbblk is never executed
b$tbt   ent   bl$tb                                  entry point (tbblk)
```

```
    teblk
    the routine for a teblk is never executed
b$tet   ent   bl$te                                   entry point (teblk)
```

```
    vcblk
    the routine for a vcblk is never executed
b$vct   ent   bl$vc                                 entry point (vcblk)
```

```
    vrblk
    the vrblk routines are executed from the generated code.
    there are six entries for vrblk covering various cases
b$vr$   ent   bl$$i                                mark start of vrblk entry points
    entry for vrget (trapped case). this routine is called
    from the generated code to load the value of a variable.
    this entry point is used if an access trace or input
    association is currently active.
    (xr)                  pointer to vrget field of vrblk
b$vra   ent   bl$$i                                entry point
        mov   xr,xl                                copy name base (vrget = 0)
        mov   *vrval,wa                            set name offset
        jsr   acess                                access value
        ppm   exfal                                fail if access fails
        mov   xr,-(xs)                             stack result
        lcw   xr                                   get next code word
        bri   (xr)                                 execute it
```

```
    vrblk (continued)
    entry for vrsto (error case. this routine is called from
    the executed code for an attempt to modify the value
    of a protected (pattern valued) natural variable.
b$vre   ent                                 entry point
        erb   042,attempt to change         value of protected variable
```

```
    vrblk (continued)
    entry for vrtra (untrapped case). this routine is called
    from the executed code to transfer to a label.
    (xr)                  pointer to vrtra field of vrblk
b$vrg   ent                                     entry point
        mov  vrlbo(xr),xr                       load code pointer
        mov  (xr),xl                            load entry address
        bri  xl                                 jump to routine for next code word
```

```
    vrblk (continued)
    entry for vrget (untrapped case). this routine is called
    from the generated code to load the value of a variable.
    (xr)                  points to vrget field of vrblk
b$vrl   ent                                     entry point
        mov  vrval(xr),-(xs)                    load value onto stack (vrget = 0)
        lcw  xr                                 get next code word
        bri  (xr)                               execute next code word
```

```
    vrblk (continued)
    entry for vrsto (untrapped case). this routine is called
    from the generated code to store the value of a variable.
    (xr)                 pointer to vrsto field of vrblk
b$vrs   ent                                     entry point
        mov  (xs),vrvlo(xr)                     store value, leave on stack
        lcw  xr                                 get next code word
        bri  (xr)                               execute next code word
```

```
    vrblk (continued)
    vrtra (trapped case). this routine is called from the
    generated code to transfer to a label when a label
    trace is currently active.
b$vrt   ent                                 entry point
        sub   *vrtra,xr                     point back to start of vrblk
        mov   xr,xl                         copy vrblk pointer
        mov   *vrval,wa                     set name offset
        mov   vrlbl(xl),xr                  load pointer to trblk
        bze   kvtra,bvrt2                   jump if trace is off
        dcv   kvtra                         else decrement trace count
        bze   trfnc(xr),bvrt1               jump if print trace case
        jsr   trxeq                         else execute full trace
        brn   bvrt2                         merge to jump to label
    here for print trace -- print colon ( label name )
bvrt1   jsr   prtsn                         print statement number
        mov   xl,xr                         copy vrblk pointer
        mov   =ch$cl,wa                     colon
        jsr   prtch                         print it
        mov   =ch$pp,wa                     left paren
        jsr   prtch                         print it
        jsr   prtvn                         print label name
        mov   =ch$rp,wa                     right paren
        jsr   prtch                         print it
        jsr   prtnl                         terminate line
        mov   vrlbl(xl),xr                  point back to trblk
    merge here to jump to label
bvrt2   mov   trlbl(xr),xr                  load pointer to actual code
        bri   (xr)                          execute statement at label
```

```
    vrblk (continued)
    entry for vrsto (trapped case). this routine is called
    from the generated code to store the value of a variable.
    this entry is used when a value trace or output
    association is currently active.
    (xr)                  pointer to vrsto field of vrblk
b$vrv   ent                                  entry point
        mov  (xs),wb                         load value (leave copy on stack)
        sub  *vrsto,xr                       point to vrblk
        mov  xr,xl                           copy vrblk pointer
        mov  *vrval,wa                       set offset
        jsr  asign                           call assignment routine
        ppm  exfal                           fail if assignment fails
        lcw  xr                              else get next code word
        bri  (xr)                            execute next code word
```

```
    xnblk
    the routine for an xnblk is never executed
b$xnt   ent   bl$xn                                  entry point (xnblk)
```

```
    xrblk
    the routine for an xrblk is never executed
b$xrt   ent   bl$xr                                  entry point (xrblk)
    mark entry address past last block action routine
b$yyy   ent   bl$$i                                  last block routine entry point
```

```
    the following section consists of the pattern matching
    routines. all pattern nodes contain a pointer (pcode)
    to one of the routines in this section (p$xxx).
    note that this section follows the b$xxx routines to
    enable a fast test for the pattern datatype.
p$aaa   ent   bl$$i                             entry to mark first pattern
    the entry conditions to the match routine are as follows
    (see o$pmn, o$pmv, o$pms and procedure match).
    stack contents.
                          name base (o$pmn only)
                          name offset (o$pmn only)
                          type (0-o$pmn, 1-o$pmv, 2-o$pms)
    pmhbs --------------- initial cursor (zero)
                          initial node pointer
    xs ------------------ =ndabo (anchored), =nduna (unanch)
    register values.
         (xs)             set as shown in stack diagram
         (xr)             pointer to initial pattern node
         (wb)             initial cursor (zero)
    global pattern values
         r$pms            pointer to subject string scblk
         pmssl            length of subject string in chars
         pmdfl            dot flag, initially zero
         pmhbs            set as shown in stack diagram
    control is passed by branching through the pcode
    field of the initial pattern node (bri (xr)).
```

```
description of algorithm
a pattern structure is represented as a linked graph
of nodes with the following structure.
     +------------------------------------+
     i                pcode               i
     +------------------------------------+
     i                pthen               i
     +------------------------------------+
     i                parm1               i
     +------------------------------------+
     i                parm2               i
     +------------------------------------+
pcode is a pointer to the routine which will perform
the match of this particular node type.
pthen is a pointer to the successor node. i.e. the node
to be matched if the attempt to match this node succeeds.
if this is the last node of the pattern pthen points
to the dummy node ndnth which initiates pattern exit.
parm1, parm2 are parameters whose use varies with the
particular node. they are only present if required.
alternatives are handled with the special alternative
node whose parameter points to the node to be matched
if there is a failure on the successor path.
the following example illustrates the manner in which
the structure is built up. the pattern is
(a / b / c) (d / e)   where / is alternation
in the diagram, the node marked + represents an
alternative node and the dotted line from a + node
represents the parameter pointer to the alternative.
+---+     +---+     +---+     +---+
i + i-----i a i-----i + i-----i d i-----
+---+     +---+  i  +---+     +---+
  .              i    .
  .              i    .
+---+     +---+  i  +---+
i + i-----i b i--i  i e i-----
+---+     +---+  i  +---+
  .              i
  .              i
+---+            i
i c i------------i
+---+
```

```
during the match, the registers are used as follows.
(xr)                 points to the current node
(xl)                 scratch
(xs)                 main stack pointer
(wb)                 cursor (number of chars matched)
(wa,wc)              scratch
to keep track of alternatives, the main stack is used as
a history stack and contains two word entries.
word 1               saved cursor value
word 2               node to match on failure
when a failure occurs, the most recent entry on this
stack is popped off to restore the cursor and point
to the node to be matched as an alternative. the entry
at the bottom of the stack points to the following
special nodes depending on the scan mode.
anchored mode        the bottom entry points to the
                     special node ndabo which causes an
                     abort. the cursor value stored
                     with this entry is always zero.
unanchored mode      the bottom entry points to the
                     special node nduna which moves the
                     anchor point and restarts the match
                     the cursor saved with this entry
                     is the number of characters which
                     lie before the initial anchor point
                     (i.e. the number of anchor moves).
                     this entry is three words long and
                     also contains the initial pattern.
entries are made on this history stack by alternative
nodes and by some special compound patterns as described
later on. the following global locations are used during
pattern matching.
r$pms                pointer to subject string
pmssl                length of subject string
pmdfl                flag set non-zero for dot patterns
pmhbs                base ptr for current history stack
the following exit points are available to match routines
succp                success in matching current node
failp                failure in matching current node
```

```
compound patterns
some patterns have implicit alternatives and their
representation in the pattern structure consists of a
linked set of nodes as indicated by these diagrams.
as before, the + represents an alternative node and
the dotted line from a + node is the parameter pointer
to the alternative pattern.
arb
---
     +---+            this node (p$arb) matches null
     i b i-----       and stacks cursor, successor ptr,
     +---+            cursor (copy) and a ptr to ndarc.
bal
---
     +---+            the p$bal node scans a balanced
     i b i-----       string and then stacks a pointer
     +---+            to itself on the history stack.
```

```
compound pattern structures (continued)
arbno
-----
     +---+            this alternative node matches null
+----i + i-----       the first time and stacks a pointer
i    +---+            to the argument pattern x.
i      .
i      .
i    +---+            node (p$aba) to stack cursor
i    i a i            and history stack base ptr.
i    +---+
i      i
i      i
i    +---+            this is the argument pattern. as
i    i x i            indicated, the successor of the
i    +---+            pattern is the p$abc node
i      i
i      i
i    +---+            this node (p$abc) pops pmhbs,
+----i c i            stacks old pmhbs and ptr to ndabd
     +---+            (unless optimization has occurred)
structure and execution of this pattern resemble those of
recursive pattern matching and immediate assignment.
the alternative node at the head of the structure matches
null initially but on subsequent failure ensures attempt
to match the argument.  before the argument is matched
p$aba stacks the cursor, pmhbs and a ptr to p$abb.  if
the argument cant be matched , p$abb removes this special
stack entry and fails.
if argument is matched , p$abc restores the outer pmhbs
value (saved by p$aba) .  then if the argument has left
alternatives on stack it stacks the inner value of pmhbs
and a ptr to ndabd. if argument left nothing on the stack
it optimises by removing items stacked by p$aba.  finally
a check is made that argument matched more than the null
string (check is intended to prevent useless looping).
if so the successor is again the alternative node at the
head of the structure , ensuring a possible extra attempt
to match the arg if necessary.  if not , the successor to
alternative is taken so as to terminate the loop.  p$abd
restores inner pmhbs ptr and fails , thus trying to match
alternatives left by the arbno argument.
```

```
compound pattern structures (continued)
breakx
------
     +---+            this node is a break node for
+----i b i            the argument to breakx, identical
i    +---+            to an ordinary break node.
i      i
i      i
i    +---+            this alternative node stacks a
i    i + i-----       pointer to the breakx node to
i    +---+            allow for subsequent failure
i      .
i      .
i    +---+            this is the breakx node itself. it
+----i x i            matches one character and then
     +---+            proceeds back to the break node.
fence
-----
     +---+            the fence node matches null and
     i f i-----       stacks a pointer to node ndabo to
     +---+            abort on a subsequent rematch
succeed
-------
     +---+            the node for succeed matches null
     i s i-----       and stacks a pointer to itself
     +---+            to repeat the match on a failure.
```

```
compound patterns (continued)
binary dot (pattern assignment)
-------------------------------
     +---+            this node (p$paa) saves the current
     i a i            cursor and a pointer to the
     +---+            special node ndpab on the stack.
       i
       i
     +---+            this is the structure for the
     i x i            pattern left argument of the
     +---+            pattern assignment call.
       i
       i
     +---+            this node (p$pac) saves the cursor,
     i c i-----       a ptr to itself, the cursor (copy)
     +---+            and a ptr to ndpad on the stack.
the function of the match routine for ndpab (p$pab)
is simply to unstack itself and fail back onto the stack.
the match routine for p$pac also sets the global pattern
flag pmdfl non-zero to indicate that pattern assignments
may have occured in the pattern match
if pmdfl is set at the end of the match (see p$nth), the
history stack is scanned for matching ndpab-ndpad pairs
and the corresponding pattern assignments are executed.
the function of the match routine for ndpad (p$pad)
is simply to remove its entry from the stack and fail.
this includes removing the special node pointer stored
in addition to the standard two entries on the stack.
```

```
compount pattern structures (continued)
fence (function)
----------------
     +---+           this node (p$fna) saves the
     i a i           current history stack and a
     +---+           pointer to ndfnb on the stack.
       i
       i
     +---+           this is the pattern structure
     i x i           given as the argument to the
     +---+           fence function.
       i
       i
     +---+           this node p$fnc restores the outer
     i c i           history stack ptr saved in p$fna,
     +---+           and stacks the inner stack base
                     ptr and a pointer to ndfnd on the
                     stack.
ndfnb (f$fnb) simply is the failure exit for pattern
argument failure, and it pops itself and fails onto the
stack.
the match routine p$fnc allows for an optimization when
the fence pattern leaves no alternatives.  in this case,
the ndfnb entry is popped, and the match continues.
ndfnd (p$fnd) is entered when the pattern fails after
going through a non-optimized p$fnc, and it pops the
stack back past the innter stack base created by p$fna
```

compound patterns (continued)
expression patterns (recursive pattern matches)
----

initial entry for a pattern node is to the routine p$exa.
if the evaluated result of the expression is itself a
pattern, then the following steps are taken to arrange
for proper recursive processing.

1) a pointer to the current node (the p$exa node) is
   stored on the history stack with a dummy cursor.
2) a special history stack entry is made in which the
   node pointer points to ndexb, and the cursor value
   is the saved value of pmhbs on entry to this node.
   the match routine for ndexb (p$exb) restores pmhbs
   from this cursor entry, pops off the p$exa node
   pointer and fails.
3) the resulting history stack pointer is saved in
   pmhbs to establish a new level of history stack.

after matching a pattern, the end of match routine gets
control (p$nth). this routine proceeds as follows.

1) load the current value of pmhbs and recognize the
   outer level case by the fact that the associated
   cursor in this case is the pattern match type code
   which is less than 3. terminate the match in this
   case and continue execution of the program.
2) otherwise make a special history stack entry in
   which the node pointer points to the special node
   ndexc and the cursor is the current value of pmhbs.
   the match routine for ndexc (p$exc) resets pmhbs to
   this (inner) value and and then fails.
3) using the history stack entry made on starting the
   expression (accessible with the current value of
   pmhbs), restore the p$exa node pointer and the old
   pmhbs setting. take the successor and continue.

an optimization is possible if the expression pattern
makes no entries on the history stack. in this case,
instead of building the p$exc node in step 2, it is more
efficient to simply pop off the p$exb entry and its
associated node pointer. the effect is the same.

```
compound patterns (continued)
binary dollar (immediate assignment)
------------------------------------
     +---+            this node (p$ima) stacks the cursor
     i a i            pmhbs and a ptr to ndimb and resets
     +---+            the stack ptr pmhbs.
       i
       i
     +---+            this is the left structure for the
     i x i            pattern left argument of the
     +---+            immediate assignment call.
       i
       i
     +---+            this node (p$imc) performs the
     i c i-----       assignment, pops pmhbs and stacks
     +---+            the old pmhbs and a ptr to ndimd.
the structure and execution of this pattern are similar
to those of the recursive expression pattern matching.
the match routine for ndimb (p$imb) restores the outer
level value of pmhbs, unstacks the saved cursor and fails
the match routine p$imc uses the current value of pmhbs
to locate the p$imb entry. this entry is used to make
the assignment and restore the outer level value of
pmhbs. finally, the inner level value of pmhbs and a
pointer to the special node ndimd are stacked.
the match routine for ndimd (p$imd) restores the inner
level value of pmhbs and fails back into the stack.
an optimization occurs if the inner pattern makes no
entries on the history stack. in this case, p$imc pops
the p$imb entry instead of making a p$imd entry.
```

```
    arbno
    see compound patterns section for stucture and
    algorithm for matching this node type.
    no parameters
p$aba   ent   bl$p0                                  p0blk
        mov   wb,-(xs)                               stack cursor
        mov   xr,-(xs)                               stack dummy node ptr
        mov   pmhbs,-(xs)                            stack old stack base ptr
        mov   =ndabb,-(xs)                           stack ptr to node ndabb
        mov   xs,pmhbs                               store new stack base ptr
        brn   succp                                  succeed
```

```
    arbno (remove p$aba special stack entry)
    no parameters (dummy pattern)
p$abb   ent                       entry point
        mov  wb,pmhbs             restore history stack base ptr
        brn  flpop                fail and pop dummy node ptr
```

```
    arbno (check if arg matched null string)
    no parameters (dummy pattern)
p$abc   ent   bl$p0                                 p0blk
        mov   pmhbs,xt                              keep p$abb stack base
        mov   num03(xt),wa                          load initial cursor
        mov   num01(xt),pmhbs                       restore outer stack base ptr
        beq   xt,xs,pabc1                           jump if no history stack entries
        mov   xt,-(xs)                              else save inner pmhbs entry
        mov   =ndabd,-(xs)                          stack ptr to special node ndabd
        brn   pabc2                                 merge
    optimise case of no extra entries on stack from arbno arg
pabc1   add   *num04,xs                             remove ndabb entry and cursor
    merge to check for matching of null string
pabc2   bne   wa,wb,succp                           allow further attempt if non-null
        mov   pthen(xr),xr                          bypass alternative node so as to ...
        brn   succp                                 ... refuse further match attempts
```

```
    arbno (try for alternatives in arbno argument)
    no parameters (dummy pattern)
p$abd   ent                         entry point
        mov  wb,pmhbs               restore inner stack base ptr
        brn  failp                  and fail
```

```
    abort
    no parameters
p$abo   ent   bl$p0                          p0blk
        brn   exfal                          signal statement failure
```

```
    alternation
    parm1                  alternative node
p$alt   ent   bl$p1                                      p1blk
        mov   wb,-(xs)                                   stack cursor
        mov   parm1(xr),-(xs)                            stack pointer to alternative
        chk                                              check for stack overflow
        brn   succp                                      if all ok, then succeed
```

```
    any (one character argument) (1-char string also)
    parm1                  character argument
p$ans   ent   bl$p1                     p1blk
        beq   wb,pmssl,failp            fail if no chars left
        mov   r$pms,xl                  else point to subject string
        plc   xl,wb                     point to current character
        lch   wa,(xl)                   load current character
        bne   wa,parm1(xr),failp        fail if no match
        icv   wb                        else bump cursor
        brn   succp                     and succeed
```

```
    any (multi-character argument case)
    parm1                  pointer to ctblk
    parm2                  bit mask to select bit in ctblk
p$any   ent   bl$p2                                        p2blk
    expression argument case merges here
pany1   beq   wb,pmssl,failp                               fail if no characters left
        mov   r$pms,xl                                     else point to subject string
        plc   xl,wb                                        get char ptr to current character
        lch   wa,(xl)                                      load current character
        mov   parm1(xr),xl                                 point to ctblk
        wtb   wa                                           change to byte offset
        add   wa,xl                                        point to entry in ctblk
        mov   ctchs(xl),wa                                 load word from ctblk
        anb   parm2(xr),wa                                 and with selected bit
        zrb   wa,failp                                     fail if no match
        icv   wb                                           else bump cursor
        brn   succp                                        and succeed
```

```
    any (expression argument)
    parm1                  expression pointer
p$ayd   ent   bl$p1                                   p1blk
        jsr   evals                                   evaluate string argument
        err   043,any evaluated                       argument is not a string
        ppm   failp                                   fail if evaluation failure
        ppm   pany1                                   merge multi-char case if ok
```

```
    p$arb                 initial arb match
    no parameters
    the p$arb node is part of a compound pattern structure
    for an arb pattern (see description of compound patterns)
p$arb   ent   bl$p0                                     p0blk
        mov   pthen(xr),xr                              load successor pointer
        mov   wb,-(xs)                                  stack dummy cursor
        mov   xr,-(xs)                                  stack successor pointer
        mov   wb,-(xs)                                  stack cursor
        mov   =ndarc,-(xs)                              stack ptr to special node ndarc
        bri   (xr)                                      execute next node matching null
```

```
    p$arc                  extend arb match
    no parameters (dummy pattern)
p$arc   ent                                        entry point
        beq  wb,pmssl,flpop                        fail and pop stack to successor
        icv  wb                                    else bump cursor
        mov  wb,-(xs)                              stack updated cursor
        mov  xr,-(xs)                              restack pointer to ndarc node
        mov  num02(xs),xr                          load successor pointer
        bri  (xr)                                  off to reexecute successor node
```

```
    bal
    no parameters
    the p$bal node is part of the compound structure built
    for bal (see section on compound patterns).
p$bal   ent  bl$p0                           p0blk
        zer  wc                              zero parentheses level counter
        mov  r$pms,xl                        point to subject string
        plc  xl,wb                           point to current character
        brn  pbal2                           jump into scan loop
    loop to scan out characters
pbal1   lch  wa,(xl)+                        load next character, bump pointer
        icv  wb                              push cursor for character
        beq  wa,=ch$pp,pbal3                 jump if left paren
        beq  wa,=ch$rp,pbal4                 jump if right paren
        bze  wc,pbal5                        else succeed if at outer level
    here after processing one character
pbal2   bne  wb,pmssl,pbal1                  loop back unless end of string
        brn  failp                           in which case, fail
    here on left paren
pbal3   icv  wc                              bump paren level
        brn  pbal2                           loop back to check end of string
    here for right paren
pbal4   bze  wc,failp                        fail if no matching left paren
        dcv  wc                              else decrement level counter
        bnz  wc,pbal2                        loop back if not at outer level
    here after successfully scanning a balanced string
pbal5   mov  wb,-(xs)                        stack cursor
        mov  xr,-(xs)                        stack ptr to bal node for extend
        brn  succp                           and succeed
```

```
    break (expression argument)
    parm1                  expression pointer
p$bkd  ent  bl$p1                          p1blk
       jsr  evals                          evaluate string expression
       err  044,break evaluated            argument is not a string
       ppm  failp                          fail if evaluation fails
       ppm  pbrk1                          merge with multi-char case if ok
```

```
    break (one character argument)
    parm1                  character argument
p$bks   ent   bl$p1                                   p1blk
        mov   pmssl,wc                                get subject string length
        sub   wb,wc                                   get number of characters left
        bze   wc,failp                                fail if no characters left
        lct   wc,wc                                   set counter for chars left
        mov   r$pms,xl                                point to subject string
        plc   xl,wb                                   point to current character
    loop to scan till break character found
pbks1   lch   wa,(xl)+                                load next char, bump pointer
        beq   wa,parm1(xr),succp                      succeed if break character found
        icv   wb                                      else push cursor
        bct   wc,pbks1                                loop back if more to go
        brn   failp                                   fail if end of string, no break chr
```

```
    break (multi-character argument)
    parm1                  pointer to ctblk
    parm2                  bit mask to select bit column
p$brk   ent   bl$p2                              p2blk
    expression argument merges here
pbrk1   mov   pmssl,wc                           load subject string length
        sub   wb,wc                              get number of characters left
        bze   wc,failp                           fail if no characters left
        lct   wc,wc                              set counter for characters left
        mov   r$pms,xl                           else point to subject string
        plc   xl,wb                              point to current character
        mov   xr,psave                           save node pointer
    loop to search for break character
pbrk2   lch   wa,(xl)+                           load next char, bump pointer
        mov   parm1(xr),xr                       load pointer to ctblk
        wtb   wa                                 convert to byte offset
        add   wa,xr                              point to ctblk entry
        mov   ctchs(xr),wa                       load ctblk word
        mov   psave,xr                           restore node pointer
        anb   parm2(xr),wa                       and with selected bit
        nzb   wa,succp                           succeed if break character found
        icv   wb                                 else push cursor
        bct   wc,pbrk2                           loop back unless end of string
        brn   failp                              fail if end of string, no break chr
```

```
    breakx (extension)
    this is the entry which causes an extension of a breakx
    match when failure occurs. see section on compound
    patterns for full details of breakx matching.
    no parameters
p$bkx   ent   bl$p0                p0blk
        icv   wb                   step cursor past previous break chr
        brn   succp                succeed to rematch break
```

```
    breakx (expression argument)
    see section on compound patterns for full structure of
    breakx pattern. the actual character matching uses a
    break node. however, the entry for the expression
    argument case is separated to get proper error messages.
    parm1                 expression pointer
p$bxd   ent  bl$p1                                   p1blk
        jsr  evals                                   evaluate string argument
        err  045,breakx evaluated                    argument is not a string
        ppm  failp                                   fail if evaluation fails
        ppm  pbrk1                                   merge with break if all ok
```

```
    cursor assignment
    parm1                  name base
    parm2                  name offset
p$cas   ent   bl$p2                            p2blk
        mov   xr,-(xs)                         save node pointer
        mov   wb,-(xs)                         save cursor
        mov   parm1(xr),xl                     load name base
        mti   wb                               load cursor as integer
        mov   parm2(xr),wb                     load name offset
        jsr   icbld                            get icblk for cursor value
        mov   wb,wa                            move name offset
        mov   xr,wb                            move value to assign
        jsr   asinp                            perform assignment
        ppm   flpop                            fail on assignment failure
        mov   (xs)+,wb                         else restore cursor
        mov   (xs)+,xr                         restore node pointer
        brn   succp                            and succeed matching null
```

```
    expression node (p$exa, initial entry)
    see compound patterns description for the structure and
    algorithms for handling expression nodes.
    parm1                 expression pointer
p$exa   ent  bl$p1                                  p1blk
        jsr  evalp                                  evaluate expression
        ppm  failp                                  fail if evaluation fails
        blo  wa,=p$aaa,pexa1                        jump if result is not a pattern
    here if result of expression is a pattern
        mov  wb,-(xs)                               stack dummy cursor
        mov  xr,-(xs)                               stack ptr to p$exa node
        mov  pmhbs,-(xs)                            stack history stack base ptr
        mov  =ndexb,-(xs)                           stack ptr to special node ndexb
        mov  xs,pmhbs                               store new stack base pointer
        mov  xl,xr                                  copy node pointer
        bri  (xr)                                   match first node in expression pat
    here if result of expression is not a pattern
pexa1   beq  wa,=b$scl,pexa2                        jump if it is already a string
        mov  xl,-(xs)                               else stack result
        mov  xr,xl                                  save node pointer
        jsr  gtstg                                  convert result to string
        err  046,expression does                    not evaluate to pattern
        mov  xr,wc                                  copy string pointer
        mov  xl,xr                                  restore node pointer
        mov  wc,xl                                  copy string pointer again
    merge here with string pointer in xl
pexa2   bze  sclen(xl),succp                        just succeed if null string
        brn  pstr1                                  else merge with string circuit
```

```
    expression node (p$exb, remove ndexb entry)
    see compound patterns description for the structure and
    algorithms for handling expression nodes.
    no parameters (dummy pattern)
p$exb   ent                     entry point
        mov  wb,pmhbs           restore outer level stack pointer
        brn  flpop              fail and pop p$exa node ptr
```

```
    expression node (p$exc, remove ndexc entry)
    see compound patterns description for the structure and
    algorithms for handling expression nodes.
    no parameters (dummy pattern)
p$exc   ent                     entry point
        mov  wb,pmhbs           restore inner stack base pointer
        brn  failp              and fail into expr pattern alternvs
```

```
    fail
    no parameters
p$fal   ent   bl$p0                                  p0blk
        brn   failp                                  just signal failure
```

```
    fence
    see compound patterns section for the structure and
    algorithm for matching this node type.
    no parameters
p$fen   ent   bl$p0                p0blk
        mov   wb,-(xs)             stack dummy cursor
        mov   =ndabo,-(xs)         stack ptr to abort node
        brn   succp                and succeed matching null
```

```
    fence (function)
    see compound patterns comments at start of this section
    for details of scheme
    no parameters
p$fna   ent  bl$p0                  p0blk
        mov  pmhbs,-(xs)            stack current history stack base
        mov  =ndfnb,-(xs)           stack indir ptr to p$fnb (failure)
        mov  xs,pmhbs               begin new history stack
        brn  succp                  succeed
```

```
    fence (function) (reset history stack and fail)
    no parameters (dummy pattern)
p$fnb   ent   bl$p0                                  p0blk
        mov   wb,pmhbs                               restore outer pmhbs stack base
        brn   failp                                  ...and fail
```

```
    fence (function) (make fence trap entry on stack)
    no parameters (dummy pattern)
p$fnc   ent   bl$p0                    p0blk
        mov   pmhbs,xt                 get inner stack base ptr
        mov   num01(xt),pmhbs          restore outer stack base
        beq   xt,xs,pfnc1              optimize if no alternatives
        mov   xt,-(xs)                 else stack inner stack base
        mov   =ndfnd,-(xs)             stack ptr to ndfnd
        brn   succp                    succeed
    here when fence function left nothing on the stack
pfnc1   add   *num02,xs                pop off p$fnb entry
        brn   succp                    succeed
```

```
    fence (function) (skip past alternatives on failure)
    no parameters (dummy pattern)
p$fnd   ent  bl$p0                p0blk
        mov  wb,xs                pop stack to fence() history base
        brn  flpop                pop base entry and fail
```

```
    immediate assignment (initial entry, save current cursor)
    see compound patterns description for details of the
    structure and algorithm for matching this node type.
    no parameters
p$ima   ent   bl$p0                  p0blk
        mov   wb,-(xs)               stack cursor
        mov   xr,-(xs)               stack dummy node pointer
        mov   pmhbs,-(xs)            stack old stack base pointer
        mov   =ndimb,-(xs)           stack ptr to special node ndimb
        mov   xs,pmhbs               store new stack base pointer
        brn   succp                  and succeed
```

```
    immediate assignment (remove cursor mark entry)
    see compound patterns description for details of the
    structure and algorithms for matching this node type.
    no parameters (dummy pattern)
p$imb   ent                                  entry point
        mov  wb,pmhbs                        restore history stack base ptr
        brn  flpop                           fail and pop dummy node ptr
```

```
     immediate assignment (perform actual assignment)
     see compound patterns description for details of the
     structure and algorithms for matching this node type.
     parm1                name base of variable
     parm2                name offset of variable
p$imc   ent  bl$p2                                 p2blk
        mov  pmhbs,xt                              load pointer to p$imb entry
        mov  wb,wa                                 copy final cursor
        mov  num03(xt),wb                          load initial cursor
        mov  num01(xt),pmhbs                       restore outer stack base pointer
        beq  xt,xs,pimc1                           jump if no history stack entries
        mov  xt,-(xs)                              else save inner pmhbs pointer
        mov  =ndimd,-(xs)                          and a ptr to special node ndimd
        brn  pimc2                                 merge
     here if no entries made on history stack
pimc1   add  *num04,xs                             remove ndimb entry and cursor
     merge here to perform assignment
pimc2   mov  wa,-(xs)                              save current (final) cursor
        mov  xr,-(xs)                              save current node pointer
        mov  r$pms,xl                              point to subject string
        sub  wb,wa                                 compute substring length
        jsr  sbstr                                 build substring
        mov  xr,wb                                 move result
        mov  (xs),xr                               reload node pointer
        mov  parm1(xr),xl                          load name base
        mov  parm2(xr),wa                          load name offset
        jsr  asinp                                 perform assignment
        ppm  flpop                                 fail if assignment fails
        mov  (xs)+,xr                              else restore node pointer
        mov  (xs)+,wb                              restore cursor
        brn  succp                                 and succeed
```

```
    immediate assignment (remove ndimd entry on failure)
    see compound patterns description for details of the
    structure and algorithms for matching this node type.
    no parameters (dummy pattern)
p$imd   ent                       entry point
        mov  wb,pmhbs             restore inner stack base pointer
        brn  failp                and fail
```

```
    len (integer argument)
    parm1                   integer argument
p$len   ent   bl$p1                                 p1blk
    expression argument case merges here
plen1   add   parm1(xr),wb                          push cursor indicated amount
        ble   wb,pmssl,succp                        succeed if not off end
        brn   failp                                 else fail
```

```
    len (expression argument)
    parm1                   expression pointer
p$lnd   ent  bl$p1                                   p1blk
        jsr  evali                                   evaluate integer argument
        err  047,len evaluated                       argument is not integer
        err  048,len evaluated                       argument is negative or too large
        ppm  failp                                   fail if evaluation fails
        ppm  plen1                                   merge with normal circuit if ok
```

```
    notany (expression argument)
    parm1                 expression pointer
p$nad   ent   bl$p1                               p1blk
        jsr   evals                               evaluate string argument
        err   049,notany evaluated                argument is not a string
        ppm   failp                               fail if evaluation fails
        ppm   pnay1                               merge with multi-char case if ok
```

```
    notany (one character argument)
    parm1                 character argument
p$nas   ent   bl$p1                          entry point
        beq   wb,pmssl,failp                 fail if no chars left
        mov   r$pms,xl                       else point to subject string
        plc   xl,wb                          point to current character in strin
        lch   wa,(xl)                        load current character
        beq   wa,parm1(xr),failp             fail if match
        icv   wb                             else bump cursor
        brn   succp                          and succeed
```

```
    notany (multi-character string argument)
    parm1                 pointer to ctblk
    parm2                 bit mask to select bit column
p$nay   ent  bl$p2                                        p2blk
    expression argument case merges here
pnay1   beq  wb,pmssl,failp                               fail if no characters left
        mov  r$pms,xl                                     else point to subject string
        plc  xl,wb                                        point to current character
        lch  wa,(xl)                                      load current character
        wtb  wa                                           convert to byte offset
        mov  parm1(xr),xl                                 load pointer to ctblk
        add  wa,xl                                        point to entry in ctblk
        mov  ctchs(xl),wa                                 load entry from ctblk
        anb  parm2(xr),wa                                 and with selected bit
        nzb  wa,failp                                     fail if character is matched
        icv  wb                                           else bump cursor
        brn  succp                                        and succeed
```

```
    end of pattern match
    this routine is entered on successful completion.
    see description of expression patterns in compound
    pattern section for handling of recursion in matching.
    this pattern also results from an attempt to convert the
    null string to a pattern via convert()
    no parameters (dummy pattern)
p$nth   ent   bl$p0                                   p0blk (dummy)
        mov   pmhbs,xt                                load pointer to base of stack
        mov   num01(xt),wa                            load saved pmhbs (or pattern type)
        ble   wa,=num02,pnth2                         jump if outer level (pattern type)
    here we are at the end of matching an expression pattern
        mov   wa,pmhbs                                restore outer stack base pointer
        mov   num02(xt),xr                            restore pointer to p$exa node
        beq   xt,xs,pnth1                             jump if no history stack entries
        mov   xt,-(xs)                                else stack inner stack base ptr
        mov   =ndexc,-(xs)                            stack ptr to special node ndexc
        brn   succp                                   and succeed
    here if no history stack entries during pattern
pnth1   add   *num04,xs                               remove p$exb entry and node ptr
        brn   succp                                   and succeed
    here if end of match at outer level
pnth2   mov   wb,pmssl                                save final cursor in safe place
        bze   pmdfl,pnth6                             jump if no pattern assignments
```

```
    end of pattern match (continued)
    now we must perform pattern assignments. this is done by
    scanning the history stack for matching ndpab-ndpad pairs
pnth3  dca  xt                          point past cursor entry
       mov  -(xt),wa                    load node pointer
       beq  wa,=ndpad,pnth4             jump if ndpad entry
       bne  wa,=ndpab,pnth5             jump if not ndpab entry
    here for ndpab entry, stack initial cursor
    note that there must be more entries on the stack.
       mov  num01(xt),-(xs)             stack initial cursor
       chk                              check for stack overflow
       brn  pnth3                       loop back if ok
    here for ndpad entry. the starting cursor from the
    matching ndpad entry is now the top stack entry.
pnth4  mov  num01(xt),wa                load final cursor
       mov  (xs),wb                     load initial cursor from stack
       mov  xt,(xs)                     save history stack scan ptr
       sub  wb,wa                       compute length of string
    build substring and perform assignment
       mov  r$pms,xl                    point to subject string
       jsr  sbstr                       construct substring
       mov  xr,wb                       copy substring pointer
       mov  (xs),xt                     reload history stack scan ptr
       mov  num02(xt),xl                load pointer to p$pac node with nam
       mov  parm2(xl),wa                load name offset
       mov  parm1(xl),xl                load name base
       jsr  asinp                       perform assignment
       ppm  exfal                       match fails if name eval fails
       mov  (xs)+,xt                    else restore history stack ptr
```

```
    end of pattern match (continued)
    here check for end of entries
pnth5   bne   xt,xs,pnth3                  loop if more entries to scan
    here after dealing with pattern assignments
pnth6   mov   pmhbs,xs                     wipe out history stack
        mov   (xs)+,wb                     load initial cursor
        mov   (xs)+,wc                     load match type code
        mov   pmssl,wa                     load final cursor value
        mov   r$pms,xl                     point to subject string
        zer   r$pms                        clear subject string ptr for gbcol
        bze   wc,pnth7                     jump if call by name
        beq   wc,=num02,pnth9              exit if statement level call
    here we have a call by value, build substring
        sub   wb,wa                        compute length of string
        jsr   sbstr                        build substring
        mov   xr,-(xs)                     stack result
        lcw   xr                           get next code word
        bri   (xr)                         execute it
    here for call by name, make stack entries for o$rpl
pnth7   mov   wb,-(xs)                     stack initial cursor
        mov   wa,-(xs)                     stack final cursor
```

---

if **.cnbf**
else

```
        bze   r$pmb,pnth8                  skip if subject not buffer
        mov   r$pmb,xl                     else get ptr to bcblk instead
```

fi

```
    here with xl pointing to scblk or bcblk
pnth8   mov   xl,-(xs)                     stack subject pointer
    here to obey next code word
pnth9   lcw   xr                           get next code word
        bri   (xr)                         execute next code word
```

```
    pos (integer argument)
    parm1                 integer argument
p$pos   ent   bl$p1                                  p1blk
    optimize pos if it is the first pattern element,
    unanchored mode, cursor is zero and pos argument
    is not beyond end of string.  force cursor position
    and number of unanchored moves.
    this optimization is performed invisible provided
    the argument is either a simple integer or an
    expression that is an untraced variable (that is,
    it has no side effects that would be lost by short-
    circuiting the normal logic of failing and moving the
    unanchored starting point.)
    pos (integer argument)
    parm1                 integer argument
        beq   wb,parm1(xr),succp                     succeed if at right location
        bnz   wb,failp                               don't look further if cursor not 0
        mov   pmhbs,xt                               get history stack base ptr
        bne   xr,-(xt),failp                         fail if pos is not first node
    expression argument circuit merges here
ppos2   bne   -(xt),=nduna,failp                     fail if not unanchored mode
        mov   parm1(xr),wb                           get desired cursor position
        bgt   wb,pmssl,exfal                         abort if off end
        mov   wb,num02(xt)                           fake number of unanchored moves
        brn   succp                                  continue match with adjusted cursor
```

```
    pos (expression argument)
    parm1                  expression pointer
p$psd   ent  bl$p1                                   p1blk
        jsr  evali                                   evaluate integer argument
        err  050,pos evaluated                       argument is not integer
        err  051,pos evaluated                       argument is negative or too large
        ppm  failp                                   fail if evaluation fails
        ppm  ppos1                                   process expression case
ppos1   beq  wb,parm1(xr),succp                      succeed if at right location
        bnz  wb,failp                                don't look further if cursor not 0
        bnz  evlif,failp                             fail if complex argument
        mov  pmhbs,xt                                get history stack base ptr
        mov  evlio,wa                                get original node ptr
        bne  wa,-(xt),failp                          fail if pos is not first node
        brn  ppos2                                   merge with integer argument code
```

```
    pattern assignment (initial entry, save cursor)
    see compound patterns description for the structure and
    algorithms for matching this node type.
    no parameters
p$paa   ent   bl$p0                          p0blk
        mov   wb,-(xs)                       stack initial cursor
        mov   =ndpab,-(xs)                   stack ptr to ndpab special node
        brn   succp                          and succeed matching null
```

```
    pattern assignment (remove saved cursor)
    see compound patterns description for the structure and
    algorithms for matching this node type.
    no parameters (dummy pattern)
```

p$pab	**ent**		entry point
	brn	failp	just fail (entry is already popped)

```
    pattern assignment (end of match, make assign entry)
    see compound patterns description for the structure and
    algorithms for matching this node type.
    parm1                 name base of variable
    parm2                 name offset of variable
p$pac   ent  bl$p2                                  p2blk
        mov  wb,-(xs)                               stack dummy cursor value
        mov  xr,-(xs)                               stack pointer to p$pac node
        mov  wb,-(xs)                               stack final cursor
        mov  =ndpad,-(xs)                           stack ptr to special ndpad node
        mnz  pmdfl                                  set dot flag non-zero
        brn  succp                                  and succeed
```

```
    pattern assignment (remove assign entry)
    see compound patterns description for the structure and
    algorithms for matching this node type.
    no parameters (dummy node)
p$pad   ent                                     entry point
        brn   flpop                             fail and remove p$pac node
```

```
    rem
    no parameters
p$rem   ent   bl$p0                     p0blk
        mov   pmssl,wb                  point cursor to end of string
        brn   succp                     and succeed
```

```
    rpos (expression argument)
    optimize rpos if it is the first pattern element,
    unanchored mode, cursor is zero and rpos argument
    is not beyond end of string.  force cursor position
    and number of unanchored moves.
    this optimization is performed invisibly provided
    the argument is either a simple integer or an
    expression that is an untraced variable (that is,
    it has no side effects that would be lost by short-
    circuiting the normal logic of failing and moving the
    unanchored starting point).
    parm1                 expression pointer
p$rpd   ent  bl$p1                      p1blk
        jsr  evali                      evaluate integer argument
        err  052,rpos evaluated         argument is not integer
        err  053,rpos evaluated         argument is negative or too large
        ppm  failp                      fail if evaluation fails
        ppm  prps1                      merge with normal case if ok
prps1   mov  pmssl,wc                   get length of string
        sub  wb,wc                      get number of characters remaining
        beq  wc,parm1(xr),succp         succeed if at right location
        bnz  wb,failp                   don't look further if cursor not 0
        bnz  evlif,failp                fail if complex argument
        mov  pmhbs,xt                   get history stack base ptr
        mov  evlio,wa                   get original node ptr
        bne  wa,-(xt),failp             fail if pos is not first node
        brn  prps2                      merge with integer arg code
```

```
    rpos (integer argument)
    parm1                 integer argument
p$rps   ent   bl$p1                                   p1blk
    rpos (integer argument)
    parm1                 integer argument
        mov   pmssl,wc                                get length of string
        sub   wb,wc                                   get number of characters remaining
        beq   wc,parm1(xr),succp                      succeed if at right location
        bnz   wb,failp                                don't look further if cursor not 0
        mov   pmhbs,xt                                get history stack base ptr
        bne   xr,-(xt),failp                          fail if rpos is not first node
    expression argument merges here
prps2   bne   -(xt),=nduna,failp                      fail if not unanchored mode
        mov   pmssl,wb                                point to end of string
        blt   wb,parm1(xr),failp                      fail if string not long enough
        sub   parm1(xr),wb                            else set new cursor
        mov   wb,num02(xt)                            fake number of unanchored moves
        brn   succp                                   continue match with adjusted cursor
```

```
    rtab (integer argument)
    parm1                  integer argument
p$rtb   ent   bl$p1                                  p1blk
    expression argument case merges here
prtb1   mov   wb,wc                                  save initial cursor
        mov   pmssl,wb                               point to end of string
        blt   wb,parm1(xr),failp                     fail if string not long enough
        sub   parm1(xr),wb                           else set new cursor
        bge   wb,wc,succp                            and succeed if not too far already
        brn   failp                                  in which case, fail
```

```
    rtab (expression argument)
    parm1                 expression pointer
p$rtd   ent   bl$p1                            p1blk
        jsr   evali                            evaluate integer argument
        err   054,rtab evaluated               argument is not integer
        err   055,rtab evaluated               argument is negative or too large
        ppm   failp                            fail if evaluation fails
        ppm   prtb1                            merge with normal case if success
```

```
    span (expression argument)
    parm1                  expression pointer
p$spd  ent  bl$p1                                p1blk
       jsr  evals                                evaluate string argument
       err  056,span evaluated                   argument is not a string
       ppm  failp                                fail if evaluation fails
       ppm  pspn1                                merge with multi-char case if ok
```

```
    span (multi-character argument case)
    parm1                  pointer to ctblk
    parm2                  bit mask to select bit column
p$spn   ent  bl$p2                                  p2blk
    expression argument case merges here
pspn1   mov  pmssl,wc                               copy subject string length
        sub  wb,wc                                  calculate number of characters left
        bze  wc,failp                               fail if no characters left
        mov  r$pms,xl                               point to subject string
        plc  xl,wb                                  point to current character
        mov  wb,psavc                               save initial cursor
        mov  xr,psave                               save node pointer
        lct  wc,wc                                  set counter for chars left
    loop to scan matching characters
pspn2   lch  wa,(xl)+                               load next character, bump pointer
        wtb  wa                                     convert to byte offset
        mov  parm1(xr),xr                           point to ctblk
        add  wa,xr                                  point to ctblk entry
        mov  ctchs(xr),wa                           load ctblk entry
        mov  psave,xr                               restore node pointer
        anb  parm2(xr),wa                           and with selected bit
        zrb  wa,pspn3                               jump if no match
        icv  wb                                     else push cursor
        bct  wc,pspn2                               loop back unless end of string
    here after scanning matching characters
pspn3   bne  wb,psavc,succp                         succeed if chars matched
        brn  failp                                  else fail if null string matched
```

```
    span (one character argument)
    parm1                  character argument
p$sps   ent   bl$p1                                   p1blk
        mov   pmssl,wc                                get subject string length
        sub   wb,wc                                   calculate number of characters left
        bze   wc,failp                                fail if no characters left
        mov   r$pms,xl                                else point to subject string
        plc   xl,wb                                   point to current character
        mov   wb,psavc                                save initial cursor
        lct   wc,wc                                   set counter for characters left
    loop to scan matching characters
psps1   lch   wa,(xl)+                                load next character, bump pointer
        bne   wa,parm1(xr),psps2                      jump if no match
        icv   wb                                      else push cursor
        bct   wc,psps1                                and loop unless end of string
    here after scanning matching characters
psps2   bne   wb,psavc,succp                          succeed if chars matched
        brn   failp                                   fail if null string matched
```

```
    multi-character string
    note that one character strings use the circuit for
    one character any arguments (p$an1).
    parm1                pointer to scblk for string arg
p$str   ent  bl$p1                                  p1blk
        mov  parm1(xr),xl                           get pointer to string
    merge here after evaluating expression with string value
pstr1   mov  xr,psave                               save node pointer
        mov  r$pms,xr                               load subject string pointer
        plc  xr,wb                                  point to current character
        add  sclen(xl),wb                           compute new cursor position
        bgt  wb,pmssl,failp                         fail if past end of string
        mov  wb,psavc                               save updated cursor
        mov  sclen(xl),wa                           get number of chars to compare
        plc  xl                                     point to chars of test string
        cmc  failp,failp                            compare, fail if not equal
        mov  psave,xr                               if all matched, restore node ptr
        mov  psavc,wb                               restore updated cursor
        brn  succp                                  and succeed
```

```
    succeed
    see section on compound patterns for details of the
    structure and algorithms for matching this node type
    no parameters
p$suc   ent  bl$p0                                   p0blk
        mov  wb,-(xs)                                stack cursor
        mov  xr,-(xs)                                stack pointer to this node
        brn  succp                                   succeed matching null
```

```
    tab (integer argument)
    parm1                 integer argument
p$tab   ent   bl$p1                                   p1blk
    expression argument case merges here
ptab1   bgt   wb,parm1(xr),failp                      fail if too far already
        mov   parm1(xr),wb                            else set new cursor position
        ble   wb,pmssl,succp                          succeed if not off end
        brn   failp                                   else fail
```

```
    tab (expression argument)
    parm1                   expression pointer
p$tbd  ent  bl$p1                                   p1blk
       jsr  evali                                   evaluate integer argument
       err  057,tab evaluated                       argument is not integer
       err  058,tab evaluated                       argument is negative or too large
       ppm  failp                                   fail if evaluation fails
       ppm  ptab1                                   merge with normal case if ok
```

```
    anchor movement
    no parameters (dummy node)
p$una   ent                             entry point
        mov  wb,xr                      copy initial pattern node pointer
        mov  (xs),wb                    get initial cursor
        beq  wb,pmssl,exfal             match fails if at end of string
        icv  wb                         else increment cursor
        mov  wb,(xs)                    store incremented cursor
        mov  xr,-(xs)                   restack initial node ptr
        mov  =nduna,-(xs)               restack unanchored node
        bri  (xr)                       rematch first node
```

```
    end of pattern match routines
    the following entry point marks the end of the pattern
    matching routines and also the end of the entry points
    referenced from the first word of blocks in dynamic store
p$yyy   ent   bl$$i                    mark last entry in pattern section
```

spitbol–snobol4 built-in label routines

```
the following section contains the routines for labels
which have a predefined meaning in snobol4.
control is passed directly to the label name entry point.
entry names are of the form l$xxx where xxx is the three
letter variable name identifier.
entries are in alphabetical order
```

```
    abort
l$abo   ent                               entry point
    merge here if execution terminates in error
labo1   mov  kvert,wa                     load error code
        bze  wa,labo3                     jump if no error has occured
```

```
if .csax
        jsr  sysax                        call after execution proc
fi
```

```
if .cera
```

```
if .csfn
        mov  kvstn,wc                     current statement
        jsr  filnm                        obtain file name for this statement
fi
```

```
if .csln
        mov  r$cod,xr                     current code block
        mov  cdsln(xr),wc                 line number
else
        zer  wc                           line number
fi
        zer  wb                           column number
        mov  wb                           column number
        jsr  sysea                        advise system of error
        ppm  stpr4                        if system does not want print
fi
        jsr  prtpg                        else eject printer
```

```
if .cera
        bze  xr,labo2                     did sysea request print
        jsr  prtst                        print text from sysea
fi
labo2   jsr  ermsg                        print error message
        zer  xr                           indicate no message to print
        brn  stopr                        jump to routine to stop run
    here if no error had occured
labo3   erb  036,goto abort with          no preceding error
```

```
    continue
l$cnt   ent                                     entry point
    merge here after execution error
lcnt1   mov  r$cnt,xr                           load continuation code block ptr
        bze  xr,lcnt3                           jump if no previous error
        zer  r$cnt                              clear flag
        mov  xr,r$cod                           else store as new code block ptr
        bne  (xr),=b$cdc,lcnt2                  jump if not complex go
        mov  stxoc,wa                           get offset of error
        bge  wa,stxof,lcnt4                     jump if error in goto evaluation
    here if error did not occur in complex failure goto
lcnt2   add  stxof,xr                           add failure offset
        lcp  xr                                 load code pointer
        mov  flptr,xs                           reset stack pointer
        lcw  xr                                 get next code word
        bri  (xr)                               execute next code word
    here if no previous error
lcnt3   icv  errft                              fatal error
        erb  037,goto continue                  with no preceding error
    here if error in evaluation of failure goto.
    cannot continue back to failure goto!
lcnt4   icv  errft                              fatal error
        erb  332,goto continue                  with error in failure goto
```

```
    end
l$end   ent                                    entry point
    merge here from end code circuit
lend0   mov  =endms,xr                         point to message /normal term.../
        brn  stopr                             jump to routine to stop run
```

```
    freturn
l$frt   ent                     entry point
        mov  =scfrt,wa          point to string /freturn/
        brn  retrn              jump to common return routine
```

```
    nreturn
l$nrt   ent                     entry point
        mov  =scnrt,wa          point to string /nreturn/
        brn  retrn              jump to common return routine
```

```
   return
l$rtn  ent                 entry point
       mov  =scrtn,wa      point to string /return/
       brn  retrn          jump to common return routine
```

```
    scontinue
l$scn   ent                                     entry point
        mov  r$cnt,xr                           load continuation code block ptr
        bze  xr,lscn2                           jump if no previous error
        zer  r$cnt                              clear flag
        bne  kvert,=nm320,lscn1                 error must be user interrupt
        beq  kvert,=nm321,lscn2                 detect scontinue loop
        mov  xr,r$cod                           else store as new code block ptr
        add  stxoc,xr                           add resume offset
        lcp  xr                                 load code pointer
        lcw  xr                                 get next code word
        bri  (xr)                               execute next code word
    here if no user interrupt
lscn1   icv  errft                              fatal error
        erb  331,goto scontinue                 with no user interrupt
    here if in scontinue loop or if no previous error
lscn2   icv  errft                              fatal error
        erb  321,goto scontinue                 with no preceding error
```

```
    undefined label
l$und   ent                                         entry point
        erb   038,goto undefined                    label
```

```
the following section contains coding for functions
which are predefined and available at the snobol level.
these routines receive control directly from the code or
indirectly through the o$fnc, o$fns or cfunc routines.
in both cases the conditions on entry are as follows
the arguments are on the stack. the number of arguments
has been adjusted to correspond to the svblk svnar field.
in certain functions the direct call is not permitted
and in these instances we also have.
(wa)                   actual number of arguments in call
control returns by placing the function result value on
on the stack and continuing execution with the next
word from the generated code.
the names of the entry points of these functions are of
the form s$xxx where xxx is the three letter code for
the system variable name. the functions are in order
alphabetically by their entry names.
```

```
if .c370
    abs
s$abs   ent                                   entry point
        mov  (xs)+,xr                         get argument
        jsr  gtnum                            make numeric
        err  xxx,abs argument                 not numeric
```

```
if .cnra
else
        beq  wa,=b$rcl,sabs1                  jump if real
fi
        ldi  icval(xr)                        load integer value
        ige  exixr                            no change if not negative
        ngi                                   produce absolute value
        ino  exint                            return integer if no overflow
        erb  xxx,abs caused integer           overflow
```

```
if .cnra
else
    here to process real argument
sabs1   ldr  rcval(xr)                        load real value
        rge  exixr                            no change if not negative
        ngr                                   produce absolute value
        rno  exrea                            return real if no overflow
        erb  xxx,abs caused real              overflow
fi
fi
```

```
if .c370
    and
s$and   ent                                   entry point
        mnz  wb                               signal two arguments
        jsr  sbool                            call string boolean routine
        err  xxx,and first argument           is not a string
        err  xxx,and second argument          is not a string
        err  xxx,and arguments                not same length
        ppm  exits                            null string arguments
    here to process (wc) words.  result is stacked.
sand1   mov  (xl)+,wa                         get next cfp$c chars from arg 1
        anb  (xr),wa                          and with characters from arg 2
        mov  wa,(xr)+                         put back in memory
        bct  wc,sand1                         loop over all words in string block
        brn  exits                            fetch next code word
```

fi

```
    any
s$any   ent                                 entry point
        mov  =p$ans,wb                      set pcode for single char case
        mov  =p$any,xl                      pcode for multi-char case
        mov  =p$ayd,wc                      pcode for expression case
        jsr  patst                          call common routine to build node
        err  059,any argument               is not a string or expression
        mov  xr,-(xs)                       stack result
        lcw  xr                             get next code word
        bri  (xr)                           execute it
```

if **.cnbf**

else

```
    append
s$apn   ent                         entry point
        mov  (xs)+,xl               get append argument
        mov  (xs)+,xr               get bcblk
        beq  (xr),=b$bct,sapn1      ok if first arg is bcblk
        erb  275,append first       argument is not a buffer
    here to do the append
sapn1   jsr  apndb                  do the append
        err  276,append second      argument is not a string
        ppm  exfal                  no room - fail
        brn  exnul                  exit with null result
```

fi

```
    apply
    apply does not permit the direct (fast) call so that
    wa contains the actual number of arguments passed.
s$app   ent                             entry point
        bze   wa,sapp3                  jump if no arguments
        dcv   wa                        else get applied func arg count
        mov   wa,wb                     copy
        wtb   wb                        convert to bytes
        mov   xs,xt                     copy stack pointer
        add   wb,xt                     point to function argument on stack
        mov   (xt),xr                   load function ptr (apply 1st arg)
        bze   wa,sapp2                  jump if no args for applied func
        lct   wb,wa                     else set counter for loop
    loop to move arguments up on stack
sapp1   dca   xt                        point to next argument
        mov   (xt),num01(xt)            move argument up
        bct   wb,sapp1                  loop till all moved
    merge here to call function (wa = number of arguments)
sapp2   ica   xs                        adjust stack ptr for apply 1st arg
        jsr   gtnvr                     get variable block addr for func
        ppm   sapp3                     jump if not natural variable
        mov   vrfnc(xr),xl              else point to function block
        brn   cfunc                     go call applied function
    here for invalid first argument
sapp3   erb   060,apply first arg       is not natural variable name
```

```
    arbno
    arbno builds a compound pattern. see description at
    start of pattern matching section for structure formed.
s$abn   ent                                 entry point
        zer   xr                            set parm1 = 0 for the moment
        mov   =p$alt,wb                     set pcode for alternative node
        jsr   pbild                         build alternative node
        mov   xr,xl                         save ptr to alternative pattern
        mov   =p$abc,wb                     pcode for p$abc
        zer   xr                            p0blk
        jsr   pbild                         build p$abc node
        mov   xl,pthen(xr)                  put alternative node as successor
        mov   xl,wa                         remember alternative node pointer
        mov   xr,xl                         copy p$abc node ptr
        mov   (xs),xr                       load arbno argument
        mov   wa,(xs)                       stack alternative node pointer
        jsr   gtpat                         get arbno argument as pattern
        err   061,arbno argument            is not pattern
        jsr   pconc                         concat arg with p$abc node
        mov   xr,xl                         remember ptr to concd patterns
        mov   =p$aba,wb                     pcode for p$aba
        zer   xr                            p0blk
        jsr   pbild                         build p$aba node
        mov   xl,pthen(xr)                  concatenate nodes
        mov   (xs),xl                       recall ptr to alternative node
        mov   xr,parm1(xl)                  point alternative back to argument
        lcw   xr                            get next code word
        bri   (xr)                          execute next code word
```

```
    arg
s$arg   ent                                 entry point
        jsr  gtsmi                          get second arg as small integer
        err  062,arg second argument        is not integer
        ppm  exfal                          fail if out of range or negative
        mov  xr,wa                          save argument number
        mov  (xs)+,xr                       load first argument
        jsr  gtnvr                          locate vrblk
        ppm  sarg1                          jump if not natural variable
        mov  vrfnc(xr),xr                   else load function block pointer
        bne  (xr),=b$pfc,sarg1              jump if not program defined
        bze  wa,exfal                       fail if arg number is zero
        bgt  wa,fargs(xr),exfal             fail if arg number is too large
        wtb  wa                             else convert to byte offset
        add  wa,xr                          point to argument selected
        mov  pfagb(xr),xr                   load argument vrblk pointer
        brn  exvnm                          exit to build nmblk
    here if 1st argument is bad
sarg1   erb  063,arg first argument         is not program function name
```

```
    array
s$arr   ent                                     entry point
        mov  (xs)+,xl                           load initial element value
        mov  (xs)+,xr                           load first argument
        jsr  gtint                              convert first arg to integer
        ppm  sar02                              jump if not integer
    here for integer first argument, build vcblk
        ldi  icval(xr)                          load integer value
        ile  sar10                              jump if zero or neg (bad dimension)
        mfi  wa,sar11                           else convert to one word, test ovfl
        jsr  vmake                              create vector
        ppm  sar11                              fail if too large
        brn  exsid                              exit setting idval
```

```
    array (continued)
    here if first argument is not an integer
sar02  mov  xr,-(xs)                      replace argument on stack
       jsr  xscni                         initialize scan of first argument
       err  064,array first argument      is not integer or string
       ppm  exnul                         dummy (unused) null string exit
       mov  r$xsc,-(xs)                   save prototype pointer
       mov  xl,-(xs)                      save default value
       zer  arcdm                         zero count of dimensions
       zer  arptr                         zero offset to indicate pass one
       ldi  intv1                         load integer one
       sti  arnel                         initialize element count
    the following code is executed twice. the first time
    (arptr eq 0), it is used to count the number of elements
    and number of dimensions. the second time (arptr gt 0) is
    used to actually fill in the dim,lbd fields of the arblk.
sar03  ldi  intv1                         load one as default low bound
       sti  arsvl                         save as low bound
       mov  =ch$cl,wc                     set delimiter one = colon
       mov  =ch$cm,xl                     set delimiter two = comma
       zer  wa                            retain blanks in prototype
       jsr  xscan                         scan next bound
       bne  wa,=num01,sar04               jump if not colon
    here we have a colon ending a low bound
       jsr  gtint                         convert low bound
       err  065,array first argument      lower bound is not integer
       ldi  icval(xr)                     load value of low bound
       sti  arsvl                         store low bound value
       mov  =ch$cm,wc                     set delimiter one = comma
       mov  wc,xl                         and delimiter two = comma
       zer  wa                            retain blanks in prototype
       jsr  xscan                         scan high bound
```

```
    array (continued)
    merge here to process upper bound
sar04   jsr   gtint                                 convert high bound to integer
        err   066,array first argument              upper bound is not integer
        ldi   icval(xr)                             get high bound
        sbi   arsvl                                 subtract lower bound
        iov   sar10                                 bad dimension if overflow
        ilt   sar10                                 bad dimension if negative
        adi   intv1                                 add 1 to get dimension
        iov   sar10                                 bad dimension if overflow
        mov   arptr,xl                              load offset (also pass indicator)
        bze   xl,sar05                              jump if first pass
    here in second pass to store lbd and dim in arblk
        add   (xs),xl                               point to current location in arblk
        sti   cfp$i(xl)                             store dimension
        ldi   arsvl                                 load low bound
        sti   (xl)                                  store low bound
        add   *ardms,arptr                          bump offset to next bounds
        brn   sar06                                 jump to check for end of bounds
    here in pass 1
sar05   icv   arcdm                                 bump dimension count
        mli   arnel                                 multiply dimension by count so far
        iov   sar11                                 too large if overflow
        sti   arnel                                 else store updated element count
    merge here after processing one set of bounds
sar06   bnz   wa,sar03                              loop back unless end of bounds
        bnz   arptr,sar09                           jump if end of pass 2
```

```
    array (continued)
    here at end of pass one, build arblk
        ldi  arnel                 get number of elements
        mfi  wb,sar11              get as addr integer, test ovflo
        wtb  wb                    else convert to length in bytes
        mov  *arsi$,wa             set size of standard fields
        lct  wc,arcdm              set dimension count to control loop
    loop to allow space for dimensions
sar07   add  *ardms,wa             allow space for one set of bounds
        bct  wc,sar07              loop back till all accounted for
        mov  wa,xl                 save size (=arofs)
    now allocate space for arblk
        add  wb,wa                 add space for elements
        ica  wa                    allow for arpro prototype field
        bgt  wa,mxlen,sar11        fail if too large
        jsr  alloc                 else allocate arblk
        mov  (xs),wb               load default value
        mov  xr,(xs)               save arblk pointer
        mov  wa,wc                 save length in bytes
        btw  wa                    convert length back to words
        lct  wa,wa                 set counter to control loop
    loop to clear entire arblk to default value
sar08   mov  wb,(xr)+              set one word
        bct  wa,sar08              loop till all set
```

```
    array (continued)
    now set initial fields of arblk
        mov  (xs)+,xr                   reload arblk pointer
        mov  (xs),wb                    load prototype
        mov  =b$art,(xr)                set type word
        mov  wc,arlen(xr)               store length in bytes
        zer  idval(xr)                  zero id till we get it built
        mov  xl,arofs(xr)               set prototype field ptr
        mov  arcdm,arndm(xr)            set number of dimensions
        mov  xr,wc                      save arblk pointer
        add  xl,xr                      point to prototype field
        mov  wb,(xr)                    store prototype ptr in arblk
        mov  *arlbd,arptr               set offset for pass 2 bounds scan
        mov  wb,r$xsc                   reset string pointer for xscan
        mov  wc,(xs)                    store arblk pointer on stack
        zer  xsofs                      reset offset ptr to start of string
        brn  sar03                      jump back to rescan bounds
    here after filling in bounds information (end pass two)
sar09   mov  (xs)+,xr                   reload pointer to arblk
        brn  exsid                      exit setting idval
    here for bad dimension
sar10   erb  067,array dimension        is zero, negative or out of range
    here if array is too large
sar11   erb  068,array size exceeds     maximum permitted
```

if **.cmth**

```
    atan
s$atn   ent                          entry point
        mov   (xs)+,xr               get argument
        jsr   gtrea                  convert to real
        err   301,atan argument      not numeric
        ldr   rcval(xr)              load accumulator with argument
        atn                          take arctangent
        brn   exrea                  overflow, out of range not possible
```

fi

---

if **.cbsp**

```
        backspace
s$bsp   ent                                 entry point
        jsr   iofcb                         call fcblk routine
        err   316,backspace argument        is not a suitable name
        err   316,backspace argument        is not a suitable name
        err   317,backspace file            does not exist
        jsr   sysbs                         call backspace file function
        err   317,backspace file            does not exist
        err   318,backspace file            does not permit backspace
        err   319,backspace caused          non-recoverable error
        brn   exnul                         return null as result
```

if **.cnbf**

else

```
    buffer
s$buf   ent                                      entry point
        mov  (xs)+,xl                            get initial value
        mov  (xs)+,xr                            get requested allocation
        jsr  gtint                               convert to integer
        err  269,buffer first                    argument is not integer
        ldi  icval(xr)                           get value
        ile  sbf01                               branch if negative or zero
        mfi  wa,sbf02                            move with overflow check
        jsr  alobf                               allocate the buffer
        jsr  apndb                               copy it in
        err  270,buffer second                   argument is not a string or buffer
        err  271,buffer initial                  value too big for allocation
        brn  exsid                               exit setting idval
    here for invalid allocation size
sbf01   erb  272,buffer first                    argument is not positive
    here for allocation size integer overflow
sbf02   erb  273,buffer size exceeds             value of maxlngth keyword
```

fi

```
    break
s$brk   ent                          entry point
        mov  =p$bks,wb               set pcode for single char case
        mov  =p$brk,xl               pcode for multi-char case
        mov  =p$bkd,wc               pcode for expression case
        jsr  patst                   call common routine to build node
        err  069,break argument      is not a string or expression
        mov  xr,-(xs)                stack result
        lcw  xr                      get next code word
        bri  (xr)                    execute it
```

```
    breakx
    breakx is a compound pattern. see description at start
    of pattern matching section for structure formed.
s$bkx   ent                          entry point
        mov  =p$bks,wb               pcode for single char argument
        mov  =p$brk,xl               pcode for multi-char argument
        mov  =p$bxd,wc               pcode for expression case
        jsr  patst                   call common routine to build node
        err  070,breakx argument     is not a string or expression
    now hook breakx node on at front end
        mov  xr,-(xs)                save ptr to break node
        mov  =p$bkx,wb               set pcode for breakx node
        jsr  pbild                   build it
        mov  (xs),pthen(xr)          set break node as successor
        mov  =p$alt,wb               set pcode for alternation node
        jsr  pbild                   build (parm1=alt=breakx node)
        mov  xr,wa                   save ptr to alternation node
        mov  (xs),xr                 point to break node
        mov  wa,pthen(xr)            set alternate node as successor
        lcw  xr                      result on stack
        bri  (xr)                    execute next code word
```

```
    char
s$chr   ent                              entry point
        jsr   gtsmi                      convert arg to integer
        err   281,char argument          not integer
        ppm   schr1                      too big error exit
        bge   wc,=cfp$a,schr1            see if out of range of host set
        mov   =num01,wa                  if not set scblk allocation
        mov   wc,wb                      save char code
        jsr   alocs                      allocate 1 bau scblk
        mov   xr,xl                      copy scblk pointer
        psc   xl                         get set to stuff char
        sch   wb,(xl)                    stuff it
        csc   xl                         complete store character
        zer   xl                         clear slop in xl
        mov   xr,-(xs)                   stack result
        lcw   xr                         get next code word
        bri   (xr)                       execute it
    here if char argument is out of range
schr1   erb   282,char argument          not in range
```

if **.cmth**

```
    chop
s$chp  ent                          entry point
       mov  (xs)+,xr                get argument
       jsr  gtrea                   convert to real
       err  302,chop argument       not numeric
       ldr  rcval(xr)               load accumulator with argument
       chp                          truncate to integer valued real
       brn  exrea                   no overflow possible
```

fi

```
    clear
s$clr   ent                                       entry point
        jsr   xscni                               initialize to scan argument
        err   071,clear argument                  is not a string
        ppm   sclr2                               jump if null
    loop to scan out names in first argument. variables in
    the list are flagged by setting vrget of vrblk to zero.
sclr1   mov   =ch$cm,wc                           set delimiter one = comma
        mov   wc,xl                               delimiter two = comma
        mnz   wa                                  skip/trim blanks in prototype
        jsr   xscan                               scan next variable name
        jsr   gtnvr                               locate vrblk
        err   072,clear argument                  has null variable name
        zer   vrget(xr)                           else flag by zeroing vrget field
        bnz   wa,sclr1                            loop back if stopped by comma
    here after flagging variables in argument list
sclr2   mov   hshtb,wb                            point to start of hash table
    loop through slots in hash table
sclr3   beq   wb,hshte,exnul                      exit returning null if none left
        mov   wb,xr                               else copy slot pointer
        ica   wb                                  bump slot pointer
        sub   *vrnxt,xr                           set offset to merge into loop
    loop through vrblks on one hash chain
sclr4   mov   vrnxt(xr),xr                        point to next vrblk on chain
        bze   xr,sclr3                            jump for next bucket if chain end
        bnz   vrget(xr),sclr5                     jump if not flagged
```

```
    clear (continued)
    here for flagged variable, do not set value to null
        jsr   setvr                               for flagged var, restore vrget
        brn   sclr4                               and loop back for next vrblk
    here to set value of a variable to null
    protected variables (arb, etc) are exempt
sclr5   beq   vrsto(xr),=b$vre,sclr4              check for protected variable
        mov   xr,xl                               copy vrblk pointer
    loop to locate value at end of possible trblk chain
sclr6   mov   xl,wa                               save block pointer
        mov   vrval(xl),xl                        load next value field
        beq   (xl),=b$trt,sclr6                   loop back if trapped
    now store the null value
        mov   wa,xl                               restore block pointer
        mov   =nulls,vrval(xl)                    store null constant value
        brn   sclr4                               loop back for next vrblk
```

```
    code
s$cod  ent                    entry point
       mov  (xs)+,xr          load argument
       jsr  gtcod             convert to code
       ppm  exfal             fail if conversion is impossible
       mov  xr,-(xs)          stack result
       zer  r$ccb             forget interim code block
       lcw  xr                get next code word
       bri  (xr)              execute it
```

```
    collect
s$col   ent                                     entry point
        mov   (xs)+,xr                          load argument
        jsr   gtint                             convert to integer
        err   073,collect argument              is not integer
        ldi   icval(xr)                         load collect argument
        sti   clsvi                             save collect argument
        zer   wb                                set no move up
        zer   r$ccb                             forget interim code block
```

---

if **.csed**

```
        zer   dnams                             collect sediment too
        jsr   gbcol                             perform garbage collection
        mov   xr,dnams                          record new sediment size
```

else

```
        jsr   gbcol                             perform garbage collection
```

fi

```
        mov   dname,wa                          point to end of memory
        sub   dnamp,wa                          subtract next location
        btw   wa                                convert bytes to words
        mti   wa                                convert words available as integer
        sbi   clsvi                             subtract argument
        iov   exfal                             fail if overflow
        ilt   exfal                             fail if not enough
        adi   clsvi                             else recompute available
        brn   exint                             and exit with integer result
```

if **.c370**

```
    compl
s$cmp   ent                              entry point
        zer  wb                          signal one argument
        jsr  sbool                       call string boolean routine
        ppm                              only one argument, cannot get here
        err  xxx,compl argument          is not a string
        ppm                              cannot have two strings unequal
        ppm  exits                       null string argument
    here to process (wa) characters.  result is stacked.
        lct  wc,wa                       prepare count
        plc  xl                          prepare to load chars from (xl)
        psc  xr                          prepare to store chars into (xr)
scmp1   lch  wa,(xl)+                    get next char from arg 1
        cmb  wa                          complement
        sch  wa,(xr)+                    store into result
        bct  wc,scmp1                    loop over all chars in string block
        csc                              complete store character
        brn  exits                       fetch next code word.
```

```
fi
    convert
s$cnv   ent                                  entry point
        jsr   gtstg                          convert second argument to string
        ppm   scv29                          error if second argument not string
        bze   wa,scv29                       or if null string
```

```
if .culc
        jsr   flstg                          fold lower case to upper case
fi
        mov   (xs),xl                        load first argument
        bne   (xl),=b$pdt,scv01              jump if not program defined
    here for program defined datatype
        mov   pddfp(xl),xl                   point to dfblk
        mov   dfnam(xl),xl                   load datatype name
        jsr   ident                          compare with second arg
        ppm   exits                          exit if ident with arg as result
        brn   exfal                          else fail
    here if not program defined datatype
scv01   mov   xr,-(xs)                       save string argument
        mov   =svctb,xl                      point to table of names to compare
        zer   wb                             initialize counter
        mov   wa,wc                          save length of argument string
    loop through table entries
scv02   mov   (xl)+,xr                       load next table entry, bump pointer
        bze   xr,exfal                       fail if zero marking end of list
        bne   wc,sclen(xr),scv05             jump if wrong length
        mov   xl,cnvtp                       else store table pointer
        plc   xr                             point to chars of table entry
        mov   (xs),xl                        load pointer to string argument
        plc   xl                             point to chars of string arg
        mov   wc,wa                          set number of chars to compare
        cmc   scv04,scv04                    compare, jump if no match
```

```
    convert (continued)
    here we have a match
scv03  mov  wb,xl                copy entry number
       ica  xs                   pop string arg off stack
       mov  (xs)+,xr             load first argument
       bsw  xl,cnvtt             jump to appropriate routine
       iff  0,scv06              string
       iff  1,scv07              integer
       iff  2,scv09              name
       iff  3,scv10              pattern
       iff  4,scv11              array
       iff  5,scv19              table
       iff  6,scv25              expression
       iff  7,scv26              code
       iff  8,scv27              numeric
```

---

if **.cnra**
else

```
       iff  cnvrt,scv08          real
```

fi

---

if **.cnbf**
else

```
       iff  cnvbt,scv28          buffer
```

fi

```
       esw                       end of switch table
    here if no match with table entry
scv04  mov  cnvtp,xl             restore table pointer, merge
    merge here if lengths did not match
scv05  icv  wb                   bump entry number
       brn  scv02                loop back to check next entry
    here to convert to string
scv06  mov  xr,-(xs)             replace string argument on stack
       jsr  gtstg                convert to string
       ppm  exfal                fail if conversion not possible
       mov  xr,-(xs)             stack result
       lcw  xr                   get next code word
       bri  (xr)                 execute it
```

```
    convert (continued)
    here to convert to integer
scv07  jsr  gtint                  convert to integer
       ppm  exfal                  fail if conversion not possible
       mov  xr,-(xs)               stack result
       lcw  xr                     get next code word
       bri  (xr)                   execute it
```

---

if **.cnra**
else

```
    here to convert to real
scv08  jsr  gtrea                  convert to real
       ppm  exfal                  fail if conversion not possible
       mov  xr,-(xs)               stack result
       lcw  xr                     get next code word
       bri  (xr)                   execute it
```

fi

```
    here to convert to name
scv09  beq  (xr),=b$nml,exixr      return if already a name
       jsr  gtnvr                  else try string to name convert
       ppm  exfal                  fail if conversion not possible
       brn  exvnm                  else exit building nmblk for vrblk
    here to convert to pattern
scv10  jsr  gtpat                  convert to pattern
       ppm  exfal                  fail if conversion not possible
       mov  xr,-(xs)               stack result
       lcw  xr                     get next code word
       bri  (xr)                   execute it
    convert to array
    if the first argument is a table, then we go through
    an intermediate array of addresses that is sorted to
    provide a result ordered by time of entry in the
    original table.  see c3.762.
scv11  mov  xr,-(xs)               save argument on stack
       zer  wa                     use table chain block addresses
       jsr  gtarr                  get an array
       ppm  exfal                  fail if empty table
       ppm  exfal                  fail if not convertible
       mov  (xs)+,xl               reload original arg
       bne  (xl),=b$tbt,exsid      exit if original not a table
       mov  xr,-(xs)               sort the intermediate array
       mov  =nulls,-(xs)           on first column
       zer  wa                     sort ascending
       jsr  sorta                  do sort
       ppm  exfal                  if sort fails, so shall we
       mov  xr,wb                  save array result
       ldi  ardim(xr)              load dim 1 (number of elements)
       mfi  wa                     get as one word integer
       lct  wa,wa                  copy to control loop
       add  *arvl2,xr              point to first element in array
    here for each row of this 2-column array
scv12  mov  (xr),xl                get teblk address
       mov  tesub(xl),(xr)+        replace with subscript
       mov  teval(xl),(xr)+        replace with value
```

```
        bct   wa,scv12              loop till all copied over
        mov   wb,xr                 retrieve array address
        brn   exsid                 exit setting id field
    convert to table
scv19   mov   (xr),wa               load first word of block
        mov   xr,-(xs)              replace arblk pointer on stack
        beq   wa,=b$tbt,exits       return arg if already a table
        bne   wa,=b$art,exfal       else fail if not an array
```

```
    convert (continued)
    here to convert an array to table
        bne  arndm(xr),=num02,exfal          fail if not 2-dim array
        ldi  ardm2(xr)                        load dim 2
        sbi  intv2                            subtract 2 to compare
        ine  exfal                            fail if dim2 not 2
    here we have an arblk of the right shape
        ldi  ardim(xr)                        load dim 1 (number of elements)
        mfi  wa                               get as one word integer
        lct  wb,wa                            copy to control loop
        add  =tbsi$,wa                        add space for standard fields
        wtb  wa                               convert length to bytes
        jsr  alloc                            allocate space for tbblk
        mov  xr,wc                            copy tbblk pointer
        mov  xr,-(xs)                         save tbblk pointer
        mov  =b$tbt,(xr)+                     store type word
        zer  (xr)+                            store zero for idval for now
        mov  wa,(xr)+                         store length
        mov  =nulls,(xr)+                     null initial lookup value
    loop to initialize bucket ptrs to point to table
scv20   mov  wc,(xr)+                         set bucket ptr to point to tbblk
        bct  wb,scv20                         loop till all initialized
        mov  *arvl2,wb                        set offset to first arblk element
    loop to copy elements from array to table
scv21   mov  num01(xs),xl                     point to arblk
        beq  wb,arlen(xl),scv24               jump if all moved
        add  wb,xl                            else point to current location
        add  *num02,wb                        bump offset
        mov  (xl),xr                          load subscript name
        dca  xl                               adjust ptr to merge (trval=1+1)
```

```
    convert (continued)
    loop to chase down trblk chain for value
scv22   mov  trval(xl),xl              point to next value
        beq  (xl),=b$trt,scv22         loop back if trapped
    here with name in xr, value in xl
scv23   mov  xl,-(xs)                  stack value
        mov  num01(xs),xl              load tbblk pointer
        jsr  tfind                     build teblk (note wb gt 0 by name)
        ppm  exfal                     fail if acess fails
        mov  (xs)+,teval(xl)           store value in teblk
        brn  scv21                     loop back for next element
    here after moving all elements to tbblk
scv24   mov  (xs)+,xr                  load tbblk pointer
        ica  xs                        pop arblk pointer
        brn  exsid                     exit setting idval
    convert to expression
```

---

if **.cevb**

```
scv25   zer  wb                        by value
        jsr  gtexp                     convert to expression
```

else

```
scv25   jsr  gtexp                     convert to expression
```

fi

```
        ppm  exfal                     fail if conversion not possible
        zer  r$ccb                     forget interim code block
        mov  xr,-(xs)                  stack result
        lcw  xr                        get next code word
        bri  (xr)                      execute it
    convert to code
scv26   jsr  gtcod                     convert to code
        ppm  exfal                     fail if conversion is not possible
        zer  r$ccb                     forget interim code block
        mov  xr,-(xs)                  stack result
        lcw  xr                        get next code word
        bri  (xr)                      execute it
    convert to numeric
scv27   jsr  gtnum                     convert to numeric
        ppm  exfal                     fail if unconvertible
scv31   mov  xr,-(xs)                  stack result
        lcw  xr                        get next code word
        bri  (xr)                      execute it
```

if **.cnbf**

else

```
    convert to buffer
scv28   mov  xr,-(xs)                 stack first arg for procedure
        jsr  gtstb                    get string or buffer
        ppm  exfal                    fail if conversion not possible
        bnz  wb,scv30                 jump if already a buffer
        mov  xr,xl                    save string pointer
        jsr  alobf                    allocate buffer of same size
        jsr  apndb                    copy in the string
        ppm                           already string - cant fail to cnv
        ppm                           must be enough room
        brn  exsid                    exit setting idval field
    here if argument is already a buffer
scv30   mov  wb,xr                    return buffer without conversion
        brn  scv31                    merge to return result
```

fi

```
    second argument not string or null
scv29  erb  074,convert second          argument is not a string
    copy
s$cop  ent                              entry point
       jsr  copyb                       copy the block
       ppm  exits                       return if no idval field
       brn  exsid                       exit setting id value
```

if **.cmth**

```
    cos
s$cos   ent                             entry point
        mov   (xs)+,xr                  get argument
        jsr   gtrea                     convert to real
        err   303,cos argument          not numeric
        ldr   rcval(xr)                 load accumulator with argument
        cos                             take cosine
        rno   exrea                     if no overflow, return result in ra
        erb   322,cos argument          is out of range
```

fi

```
    data
s$dat   ent                               entry point
        jsr   xscni                       prepare to scan argument
        err   075,data argument           is not a string
        err   076,data argument           is null
    scan out datatype name
        mov   =ch$pp,wc                   delimiter one = left paren
        mov   wc,xl                       delimiter two = left paren
        mnz   wa                          skip/trim blanks in prototype
        jsr   xscan                       scan datatype name
        bnz   wa,sdat1                    skip if left paren found
        erb   077,data argument           is missing a left paren
    here after scanning datatype name
```

---

if **.culc**

```
sdat1   mov   sclen(xr),wa                get length
        bze   wa,sdt1a                    avoid folding if null string
        jsr   flstg                       fold lower case to upper case
sdt1a   mov   xr,xl                       save name ptr
```

else

```
sdat1   mov   xr,xl                       save name ptr
```

fi

```
        mov   sclen(xr),wa                get length
        ctb   wa,scsi$                    compute space needed
        jsr   alost                       request static store for name
        mov   xr,-(xs)                    save datatype name
        mvw                               copy name to static
        mov   (xs),xr                     get name ptr
        zer   xl                          scrub dud register
        jsr   gtnvr                       locate vrblk for datatype name
        err   078,data argument           has null datatype name
        mov   xr,datdv                    save vrblk pointer for datatype
        mov   xs,datxs                    store starting stack value
        zer   wb                          zero count of field names
    loop to scan field names and stack vrblk pointers
sdat2   mov   =ch$rp,wc                   delimiter one = right paren
        mov   =ch$cm,xl                   delimiter two = comma
        mnz   wa                          skip/trim blanks in prototype
        jsr   xscan                       scan next field name
        bnz   wa,sdat3                    jump if delimiter found
        erb   079,data argument           is missing a right paren
    here after scanning out one field name
sdat3   jsr   gtnvr                       locate vrblk for field name
        err   080,data argument           has null field name
        mov   xr,-(xs)                    stack vrblk pointer
        icv   wb                          increment counter
        beq   wa,=num02,sdat2             loop back if stopped by comma
```

```
    data (continued)
    now build the dfblk
         mov  =dfsi$,wa            set size of dfblk standard fields
         add  wb,wa                add number of fields
         wtb  wa                   convert length to bytes
         mov  wb,wc                preserve no. of fields
         jsr  alost                allocate space for dfblk
         mov  wc,wb                get no of fields
         mov  datxs,xt             point to start of stack
         mov  (xt),wc              load datatype name
         mov  xr,(xt)              save dfblk pointer on stack
         mov  =b$dfc,(xr)+         store type word
         mov  wb,(xr)+             store number of fields (fargs)
         mov  wa,(xr)+             store length (dflen)
         sub  *pddfs,wa            compute pdblk length (for dfpdl)
         mov  wa,(xr)+             store pdblk length (dfpdl)
         mov  wc,(xr)+             store datatype name (dfnam)
         lct  wc,wb                copy number of fields
    loop to move field name vrblk pointers to dfblk
sdat4    mov  -(xt),(xr)+          move one field name vrblk pointer
         bct  wc,sdat4             loop till all moved
    now define the datatype function
         mov  wa,wc                copy length of pdblk for later loop
         mov  datdv,xr             point to vrblk
         mov  datxs,xt             point back on stack
         mov  (xt),xl              load dfblk pointer
         jsr  dffnc                define function
```

```
    data (continued)
    loop to build ffblks
    notice that the ffblks are constructed in reverse order
    so that the required offsets can be obtained from
    successive decrementation of the pdblk length (in wc).
sdat5  mov  *ffsi$,wa                 set length of ffblk
       jsr  alloc                     allocate space for ffblk
       mov  =b$ffc,(xr)               set type word
       mov  =num01,fargs(xr)          store fargs (always one)
       mov  datxs,xt                  point back on stack
       mov  (xt),ffdfp(xr)            copy dfblk ptr to ffblk
       dca  wc                        decrement old dfpdl to get next ofs
       mov  wc,ffofs(xr)              set offset to this field
       zer  ffnxt(xr)                 tentatively set zero forward ptr
       mov  xr,xl                     copy ffblk pointer for dffnc
       mov  (xs),xr                   load vrblk pointer for field
       mov  vrfnc(xr),xr              load current function pointer
       bne  (xr),=b$ffc,sdat6         skip if not currently a field func
    here we must chain an old ffblk ptr to preserve it in the
    case of multiple field functions with the same name
       mov  xr,ffnxt(xl)              link new ffblk to previous chain
    merge here to define field function
sdat6  mov  (xs)+,xr                  load vrblk pointer
       jsr  dffnc                     define field function
       bne  xs,datxs,sdat5            loop back till all done
       ica  xs                        pop dfblk pointer
       brn  exnul                     return with null result
```

```
    datatype
s$dtp   ent                 entry point
        mov  (xs)+,xr       load argument
        jsr  dtype          get datatype
        mov  xr,-(xs)       stack result
        lcw  xr             get next code word
        bri  (xr)           execute it
```

```
    date
s$dte   ent                         entry point
        mov  (xs)+,xr               load argument
        jsr  gtint                  convert to an integer
        err  330,date argument      is not integer
        jsr  sysdt                  call system date routine
        mov  num01(xl),wa           load length for sbstr
        bze  wa,exnul               return null if length is zero
        zer  wb                     set zero offset
        jsr  sbstr                  use sbstr to build scblk
        mov  xr,-(xs)               stack result
        lcw  xr                     get next code word
        bri  (xr)                   execute it
```

```
    define
s$def   ent                                entry point
        mov  (xs)+,xr                      load second argument
        zer  deflb                         zero label pointer in case null
        beq  xr,=nulls,sdf01               jump if null second argument
        jsr  gtnvr                         else find vrblk for label
        ppm  sdf12                         jump if not a variable name
        mov  xr,deflb                      else set specified entry
    scan function name
sdf01   jsr  xscni                         prepare to scan first argument
        err  081,define first              argument is not a string
        err  082,define first              argument is null
        mov  =ch$pp,wc                     delimiter one = left paren
        mov  wc,xl                         delimiter two = left paren
        mnz  wa                            skip/trim blanks in prototype
        jsr  xscan                         scan out function name
        bnz  wa,sdf02                      jump if left paren found
        erb  083,define first              argument is missing a left paren
    here after scanning out function name
sdf02   jsr  gtnvr                         get variable name
        err  084,define first              argument has null function name
        mov  xr,defvr                      save vrblk pointer for function nam
        zer  wb                            zero count of arguments
        mov  xs,defxs                      save initial stack pointer
        bnz  deflb,sdf03                   jump if second argument given
        mov  xr,deflb                      else default is function name
    loop to scan argument names and stack vrblk pointers
sdf03   mov  =ch$rp,wc                     delimiter one = right paren
        mov  =ch$cm,xl                     delimiter two = comma
        mnz  wa                            skip/trim blanks in prototype
        jsr  xscan                         scan out next argument name
        bnz  wa,sdf04                      skip if delimiter found
        erb  085,null arg name             or missing ) in define first arg.
```

```
    define (continued)
    here after scanning an argument name
sdf04  bne  xr,=nulls,sdf05              skip if non-null
       bze  wb,sdf06                     ignore null if case of no arguments
    here after dealing with the case of no arguments
sdf05  jsr  gtnvr                        get vrblk pointer
       ppm  sdf03                        loop back to ignore null name
       mov  xr,-(xs)                     stack argument vrblk pointer
       icv  wb                           increment counter
       beq  wa,=num02,sdf03              loop back if stopped by a comma
    here after scanning out function argument names
sdf06  mov  wb,defna                     save number of arguments
       zer  wb                           zero count of locals
    loop to scan local names and stack vrblk pointers
sdf07  mov  =ch$cm,wc                    set delimiter one = comma
       mov  wc,xl                        set delimiter two = comma
       mnz  wa                           skip/trim blanks in prototype
       jsr  xscan                        scan out next local name
       bne  xr,=nulls,sdf08              skip if non-null
       bze  wa,sdf09                     exit scan if end of string
    here after scanning out a local name
sdf08  jsr  gtnvr                        get vrblk pointer
       ppm  sdf07                        loop back to ignore null name
       icv  wb                           if ok, increment count
       mov  xr,-(xs)                     stack vrblk pointer
       bnz  wa,sdf07                     loop back if stopped by a comma
```

```
    define (continued)
    here after scanning locals, build pfblk
sdf09   mov  wb,wa                     copy count of locals
        add  defna,wa                  add number of arguments
        mov  wa,wc                     set sum args+locals as loop count
        add  =pfsi$,wa                 add space for standard fields
        wtb  wa                        convert length to bytes
        jsr  alloc                     allocate space for pfblk
        mov  xr,xl                     save pointer to pfblk
        mov  =b$pfc,(xr)+              store first word
        mov  defna,(xr)+               store number of arguments
        mov  wa,(xr)+                  store length (pflen)
        mov  defvr,(xr)+               store vrblk ptr for function name
        mov  wb,(xr)+                  store number of locals
        zer  (xr)+                     deal with label later
        zer  (xr)+                     zero pfctr
        zer  (xr)+                     zero pfrtr
        bze  wc,sdf11                  skip if no args or locals
        mov  xl,wa                     keep pfblk pointer
        mov  defxs,xt                  point before arguments
        lct  wc,wc                     get count of args+locals for loop
    loop to move locals and args to pfblk
sdf10   mov  -(xt),(xr)+               store one entry and bump pointers
        bct  wc,sdf10                  loop till all stored
        mov  wa,xl                     recover pfblk pointer
```

```
    define (continued)
    now deal with label
sdf11   mov  defxs,xs                pop stack
        mov  deflb,pfcod(xl)         store label vrblk in pfblk
        mov  defvr,xr                point back to vrblk for function
        jsr  dffnc                   define function
        brn  exnul                   and exit returning null
    here for erroneous label
sdf12   erb  086,define function     entry point is not defined label
```

```
    detach
s$det   ent                             entry point
        mov  (xs)+,xr                   load argument
        jsr  gtvar                      locate variable
        err  087,detach argument        is not appropriate name
        jsr  dtach                      detach i/o association from name
        brn  exnul                      return null result
```

```
    differ
s$dif   ent                 entry point
        mov  (xs)+,xr       load second argument
        mov  (xs)+,xl       load first argument
        jsr  ident          call ident comparison routine
        ppm  exfal          fail if ident
        brn  exnul          return null if differ
```

```
    dump
s$dmp   ent                                 entry point
        jsr   gtsmi                         load dump arg as small integer
        err   088,dump argument             is not integer
        err   089,dump argument             is negative or too large
        jsr   dumpr                         else call dump routine
        brn   exnul                         and return null as result
```

```
    dupl
s$dup   ent                                  entry point
        jsr  gtsmi                           get second argument as small integr
        err  090,dupl second argument        is not integer
        ppm  sdup7                           jump if negative or too big
        mov  xr,wb                           save duplication factor
        jsr  gtstg                           get first arg as string
        ppm  sdup4                           jump if not a string
    here for case of duplication of a string
        mti  wa                              acquire length as integer
        sti  dupsi                           save for the moment
        mti  wb                              get duplication factor as integer
        mli  dupsi                           form product
        iov  sdup3                           jump if overflow
        ieq  exnul                           return null if result length = 0
        mfi  wa,sdup3                        get as addr integer, check ovflo
    merge here with result length in wa
sdup1   mov  xr,xl                           save string pointer
        jsr  alocs                           allocate space for string
        mov  xr,-(xs)                        save as result pointer
        mov  xl,wc                           save pointer to argument string
        psc  xr                              prepare to store chars of result
        lct  wb,wb                           set counter to control loop
    loop through duplications
sdup2   mov  wc,xl                           point back to argument string
        mov  sclen(xl),wa                    get number of characters
        plc  xl                              point to chars in argument string
        mvc                                  move characters to result string
        bct  wb,sdup2                        loop till all duplications done
        zer  xl                              clear garbage value
        lcw  xr                              get next code word
        bri  (xr)                            execute next code word
```

```
    dupl (continued)
    here if too large, set max length and let alocs catch it
sdup3   mov  dname,wa                          set impossible length for alocs
        brn  sdup1                             merge back
    here if not a string
sdup4   jsr  gtpat                             convert argument to pattern
        err  091,dupl first argument           is not a string or pattern
    here to duplicate a pattern argument
        mov  xr,-(xs)                          store pattern on stack
        mov  =ndnth,xr                         start off with null pattern
        bze  wb,sdup6                          null pattern is result if dupfac=0
        mov  wb,-(xs)                          preserve loop count
    loop to duplicate by successive concatenation
sdup5   mov  xr,xl                             copy current value as right argumnt
        mov  num01(xs),xr                      get a new copy of left
        jsr  pconc                             concatenate
        dcv  (xs)                              count down
        bnz  (xs),sdup5                        loop
        ica  xs                                pop loop count
    here to exit after constructing pattern
sdup6   mov  xr,(xs)                           store result on stack
        lcw  xr                                get next code word
        bri  (xr)                              execute next code word
    fail if second arg is out of range
sdup7   ica  xs                                pop first argument
        brn  exfal                             fail
```

```
    eject
s$ejc   ent                                   entry point
        jsr   iofcb                           call fcblk routine
        err   092,eject argument              is not a suitable name
        ppm   sejc1                           null argument
        err   093,eject file does             not exist
        jsr   sysef                           call eject file function
        err   093,eject file does             not exist
        err   094,eject file does             not permit page eject
        err   095,eject caused                non-recoverable output error
        brn   exnul                           return null as result
    here to eject standard output file
sejc1   jsr   sysep                           call routine to eject printer
        brn   exnul                           exit with null result
```

```
    endfile
s$enf   ent                                     entry point
        jsr   iofcb                             call fcblk routine
        err   096,endfile argument              is not a suitable name
        err   097,endfile argument              is null
        err   098,endfile file                  does not exist
        jsr   sysen                             call endfile routine
        err   098,endfile file                  does not exist
        err   099,endfile file                  does not permit endfile
        err   100,endfile caused                non-recoverable output error
        mov   xl,wb                             remember vrblk ptr from iofcb call
        mov   xl,xr                             copy pointer
    loop to find trtrf block
senf1   mov   xr,xl                             remember previous entry
        mov   trval(xr),xr                      chain along
        bne   (xr),=b$trt,exnul                 skip out if chain end
        bne   trtyp(xr),=trtfc,senf1            loop if not found
        mov   trval(xr),trval(xl)               remove trtrf
        mov   trtrf(xr),enfch                   point to head of iochn
        mov   trfpt(xr),wc                      point to fcblk
        mov   wb,xr                             filearg1 vrblk from iofcb
        jsr   setvr                             reset it
        mov   =r$fcb,xl                         ptr to head of fcblk chain
        sub   *num02,xl                         adjust ready to enter loop
    find fcblk
senf2   mov   xl,xr                             copy ptr
        mov   num02(xl),xl                      get next link
        bze   xl,senf4                          stop if chain end
        beq   num03(xl),wc,senf3                jump if fcblk found
        brn   senf2                             loop
    remove fcblk
senf3   mov   num02(xl),num02(xr)               delete fcblk from chain
    loop which detaches all vbls on iochn chain
senf4   mov   enfch,xl                          get chain head
        bze   xl,exnul                          finished if chain end
        mov   trtrf(xl),enfch                   chain along
        mov   ionmo(xl),wa                      name offset
        mov   ionmb(xl),xl                      name base
        jsr   dtach                             detach name
        brn   senf4                             loop till done
```

```
    eq
s$eqf   ent                                 entry point
        jsr   acomp                         call arithmetic comparison routine
        err   101,eq first argument         is not numeric
        err   102,eq second argument        is not numeric
        ppm   exfal                         fail if lt
        ppm   exnul                         return null if eq
        ppm   exfal                         fail if gt
```

```
    eval
s$evl   ent                              entry point
        mov  (xs)+,xr                    load argument
```

---

if **.cevb**

else

```
        jsr  gtexp                       convert to expression
        err  103,eval argument           is not expression
```

fi

```
        lcw  wc                          load next code word
        bne  wc,=ofne$,sevl1             jump if called by value
        scp  xl                          copy code pointer
        mov  (xl),wa                     get next code word
        bne  wa,=ornm$,sevl2             by name unless expression
        bnz  num01(xs),sevl2             jump if by name
    here if called by value
sevl1   zer  wb                          set flag for by value
```

---

if **.cevb**

```
        mov  wc,-(xs)                    save code word
        jsr  gtexp                       convert to expression
        err  103,eval argument           is not expression
        zer  r$ccb                       forget interim code block
        zer  wb                          set flag for by value
```

else

```
        mov  wc,-(xs)                    save code word
```

fi

```
        jsr  evalx                       evaluate expression by value
        ppm  exfal                       fail if evaluation fails
        mov  xr,xl                       copy result
        mov  (xs),xr                     reload next code word
        mov  xl,(xs)                     stack result
        bri  (xr)                        jump to execute next code word
    here if called by name
sevl2   mov  =num01,wb                   set flag for by name
```

---

if **.cevb**

```
        jsr  gtexp                       convert to expression
        err  103,eval argument           is not expression
        zer  r$ccb                       forget interim code block
        mov  =num01,wb                   set flag for by name
```

fi

```
        jsr  evalx                       evaluate expression by name
        ppm  exfal                       fail if evaluation fails
        brn  exnam                       exit with name
```

---

if **.cnex**

else

```
    exit
s$ext   ent                                   entry point
        zer   wb                              clear amount of static shift
        zer   r$ccb                           forget interim code block
```

```
if .csed
        zer   dnams                           collect sediment too
        jsr   gbcol                           compact memory by collecting
        mov   xr,dnams                        record new sediment size
else
        jsr   gbcol                           compact memory by collecting
fi
        jsr   gbcol                           compact memory by collecting
        err   288,exit second argument        is not a string
        mov   xr,xl                           copy second arg string pointer
        jsr   gtstg                           convert arg to string
        err   104,exit first argument         is not suitable integer or string
        mov   xl,-(xs)                        save second argument
        mov   xr,xl                           copy first arg string ptr
        jsr   gtint                           check it is integer
        ppm   sext1                           skip if unconvertible
        zer   xl                              note it is integer
        ldi   icval(xr)                       get integer arg
    merge to call osint exit routine
sext1   mov   r$fcb,wb                        get fcblk chain header
        mov   =headv,xr                       point to v.v string
        mov   (xs)+,wa                        provide second argument scblk
        jsr   sysxi                           call external routine
        err   105,exit action not             available in this implementation
        err   106,exit action caused          irrecoverable error
        ieq   exnul                           return if argument 0
        igt   sext2                           skip if positive
        ngi                                   make positive
    check for option respecification
    sysxi returns 0 in wa when a file has been resumed,
    1 when this is a continuation of an exit(4) or exit(-4)
    action.
sext2   mfi   wc                              get value in work reg
        add   wc,wa                           prepare to test for continue
        beq   wa,=num05,sext5                 continued execution if 4 plus 1
        zer   gbcnt                           resuming execution so reset
        bge   wc,=num03,sext3                 skip if was 3 or 4
        mov   wc,-(xs)                        save value
        zer   wc                              set to read options
        jsr   prpar                           read syspp options
        mov   (xs)+,wc                        restore value
    deal with header option (fiddled by prpar)
sext3   mnz   headp                           assume no headers
        bne   wc,=num01,sext4                 skip if not 1
        zer   headp                           request header printing
    almost ready to resume running
sext4   jsr   systm                           get execution time start (sgd11)
        sti   timsx                           save as initial time
        ldi   kvstc                           reset to ensure ...
```

```
        sti   kvstl                             ... correct execution stats
        jsr   stgcc                             recompute countdown counters
        brn   exnul                             resume execution
    here after exit(4) or exit(-4) -- create save file
    or load module and continue execution.
    return integer 1 to signal the continuation of the
    original execution.
sext5   mov   =inton,xr                         integer one
        brn   exixr                             return as result
fi
```

if **.cmth**

```
    exp
s$exp   ent                          entry point
        mov  (xs)+,xr                get argument
        jsr  gtrea                   convert to real
        err  304,exp argument        not numeric
        ldr  rcval(xr)               load accumulator with argument
        etx                          take exponential
        rno  exrea                   if no overflow, return result in ra
        erb  305,exp produced        real overflow
```

fi

```
   field
s$fld   ent                                 entry point
        jsr  gtsmi                          get second argument (field number)
        err  107,field second               argument is not integer
        ppm  exfal                          fail if out of range
        mov  xr,wb                          else save integer value
        mov  (xs)+,xr                       load first argument
        jsr  gtnvr                          point to vrblk
        ppm  sfld1                          jump (error) if not variable name
        mov  vrfnc(xr),xr                   else point to function block
        bne  (xr),=b$dfc,sfld1              error if not datatype function
   here if first argument is a datatype function name
        bze  wb,exfal                       fail if argument number is zero
        bgt  wb,fargs(xr),exfal             fail if too large
        wtb  wb                             else convert to byte offset
        add  wb,xr                          point to field name
        mov  dfflb(xr),xr                   load vrblk pointer
        brn  exvnm                          exit to build nmblk
   here for bad first argument
sfld1   erb  108,field first argument       is not datatype name
```

```
   fence
s$fnc   ent                          entry point
        mov  =p$fnc,wb               set pcode for p$fnc
        zer  xr                      p0blk
        jsr  pbild                   build p$fnc node
        mov  xr,xl                   save pointer to it
        mov  (xs)+,xr                get argument
        jsr  gtpat                   convert to pattern
        err  259,fence argument      is not pattern
        jsr  pconc                   concatenate to p$fnc node
        mov  xr,xl                   save ptr to concatenated pattern
        mov  =p$fna,wb               set for p$fna pcode
        zer  xr                      p0blk
        jsr  pbild                   construct p$fna node
        mov  xl,pthen(xr)            set pattern as pthen
        mov  xr,-(xs)                set as result
        lcw  xr                      get next code word
        bri  (xr)                    execute next code word
```

```
    ge
s$gef  ent                                  entry point
       jsr  acomp                           call arithmetic comparison routine
       err  109,ge first argument           is not numeric
       err  110,ge second argument          is not numeric
       ppm  exfal                           fail if lt
       ppm  exnul                           return null if eq
       ppm  exnul                           return null if gt
```

```
    gt
s$gtf  ent                                   entry point
       jsr  acomp                            call arithmetic comparison routine
       err  111,gt first argument            is not numeric
       err  112,gt second argument           is not numeric
       ppm  exfal                            fail if lt
       ppm  exfal                            fail if eq
       ppm  exnul                            return null if gt
```

```
    host
s$hst   ent                                entry point
        mov  (xs)+,wc                      get fifth arg
        mov  (xs)+,wb                      get fourth arg
        mov  (xs)+,xr                      get third arg
        mov  (xs)+,xl                      get second arg
        mov  (xs)+,wa                      get first arg
        jsr  syshs                         enter syshs routine
        err  254,erroneous argument        for host
        err  255,error during              execution of host
        ppm  shst1                         store host string
        ppm  exnul                         return null result
        ppm  exixr                         return xr
        ppm  exfal                         fail return
        ppm  shst3                         store actual string
        ppm  shst4                         return copy of xr
    return host string
shst1   bze  xl,exnul                      null string if syshs uncooperative
        mov  sclen(xl),wa                  length
        zer  wb                            zero offset
    copy string and return
shst2   jsr  sbstr                         build copy of string
        mov  xr,-(xs)                      stack the result
        lcw  xr                            load next code word
        bri  (xr)                          execute it
    return actual string pointed to by xl
shst3   zer  wb                            treat xl like an scblk ptr
        sub  =cfp$f,wb                     by creating a negative offset
        brn  shst2                         join to copy string
    return copy of block pointed to by xr
shst4   mov  xr,-(xs)                      stack results
        jsr  copyb                         make copy of block
        ppm  exits                         if not an aggregate structure
        brn  exsid                         set current id value otherwise
```

```
    ident
s$idn  ent                entry point
       mov  (xs)+,xr      load second argument
       mov  (xs)+,xl      load first argument
       jsr  ident         call ident comparison routine
       ppm  exnul         return null if ident
       brn  exfal         fail if differ
```

```
    input
s$inp   ent                                     entry point
        zer   wb                                input flag
        jsr   ioput                             call input/output assoc. routine
        err   113,input third argument          is not a string
        err   114,inappropriate                 second argument for input
        err   115,inappropriate                 first argument for input
        err   116,inappropriate                 file specification for input
        ppm   exfal                             fail if file does not exist
        err   117,input file cannot             be read
        err   289,input channel                 currently in use
        brn   exnul                             return null string
```

if **.cnbf**
else

```
    insert
s$ins   ent                                 entry point
        mov  (xs)+,xl                       get string arg
        jsr  gtsmi                          get replace length
        err  277,insert third               argument not integer
        ppm  exfal                          fail if out of range
        mov  wc,wb                          copy to proper reg
        jsr  gtsmi                          get replace position
        err  278,insert second              argument not integer
        ppm  exfal                          fail if out of range
        bze  wc,exfal                       fail if zero
        dcv  wc                             decrement to get offset
        mov  wc,wa                          put in proper register
        mov  (xs)+,xr                       get buffer
        beq  (xr),=b$bct,sins1              press on if type ok
        erb  279,insert first               argument is not a buffer
    here when everything loaded up
sins1   jsr  insbf                          call to insert
        err  280,insert fourth              argument is not a string
        ppm  exfal                          fail if out of range
        brn  exnul                          else ok - exit with null
```

fi

```
    integer
s$int   ent                             entry point
        mov   (xs)+,xr                  load argument
        jsr   gtnum                     convert to numeric
        ppm   exfal                     fail if non-numeric
        beq   wa,=b$icl,exnul           return null if integer
        brn   exfal                     fail if real
```

```
    item
    item does not permit the direct (fast) call so that
    wa contains the actual number of arguments passed.
s$itm   ent                                     entry point
    deal with case of no args
        bnz   wa,sitm1                          jump if at least one arg
        mov   =nulls,-(xs)                      else supply garbage null arg
        mov   =num01,wa                         and fix argument count
    check for name/value cases
sitm1   scp   xr                                get current code pointer
        mov   (xr),xl                           load next code word
        dcv   wa                                get number of subscripts
        mov   wa,xr                             copy for arref
        beq   xl,=ofne$,sitm2                   jump if called by name
    here if called by value
        zer   wb                                set code for call by value
        brn   arref                             off to array reference routine
    here for call by name
sitm2   mnz   wb                                set code for call by name
        lcw   wa                                load and ignore ofne$ call
        brn   arref                             off to array reference routine
```

```
    le
s$lef   ent                                 entry point
        jsr  acomp                          call arithmetic comparison routine
        err  118,le first argument          is not numeric
        err  119,le second argument         is not numeric
        ppm  exnul                          return null if lt
        ppm  exnul                          return null if eq
        ppm  exfal                          fail if gt
```

```
    len
s$len   ent                                 entry point
        mov  =p$len,wb                      set pcode for integer arg case
        mov  =p$lnd,wa                      set pcode for expr arg case
        jsr  patin                          call common routine to build node
        err  120,len argument               is not integer or expression
        err  121,len argument               is negative or too large
        mov  xr,-(xs)                       stack result
        lcw  xr                             get next code word
        bri  (xr)                           execute it
```

```
    leq
s$leq   ent                                     entry point
        jsr   lcomp                             call string comparison routine
        err   122,leq first argument            is not a string
        err   123,leq second argument           is not a string
        ppm   exfal                             fail if llt
        ppm   exnul                             return null if leq
        ppm   exfal                             fail if lgt
```

```
    lge
s$lge   ent                                  entry point
        jsr   lcomp                          call string comparison routine
        err   124,lge first argument         is not a string
        err   125,lge second argument        is not a string
        ppm   exfal                          fail if llt
        ppm   exnul                          return null if leq
        ppm   exnul                          return null if lgt
```

```
    lgt
s$lgt   ent                                      entry point
        jsr  lcomp                               call string comparison routine
        err  126,lgt first argument              is not a string
        err  127,lgt second argument             is not a string
        ppm  exfal                               fail if llt
        ppm  exfal                               fail if leq
        ppm  exnul                               return null if lgt
```

```
    lle
s$lle   ent                                  entry point
        jsr   lcomp                          call string comparison routine
        err   128,lle first argument         is not a string
        err   129,lle second argument        is not a string
        ppm   exnul                          return null if llt
        ppm   exnul                          return null if leq
        ppm   exfal                          fail if lgt
```

```
    llt
s$llt   ent                                    entry point
        jsr   lcomp                            call string comparison routine
        err   130,llt first argument           is not a string
        err   131,llt second argument          is not a string
        ppm   exnul                            return null if llt
        ppm   exfal                            fail if leq
        ppm   exfal                            fail if lgt
```

```
    lne
s$lne   ent                                     entry point
        jsr   lcomp                             call string comparison routine
        err   132,lne first argument            is not a string
        err   133,lne second argument           is not a string
        ppm   exnul                             return null if llt
        ppm   exfal                             fail if leq
        ppm   exnul                             return null if lgt
```

if **.cmth**

```
    ln
s$lnf   ent                                   entry point
        mov  (xs)+,xr                         get argument
        jsr  gtrea                            convert to real
        err  306,ln argument not              numeric
        ldr  rcval(xr)                        load accumulator with argument
        req  slnf1                            overflow if argument is 0
        rlt  slnf2                            error if argument less than 0
        lnf                                   take natural logarithm
        rno  exrea                            if no overflow, return result in ra
slnf1   erb  307,ln produced real             overflow
    here for bad argument
slnf2   erb  307,ln produced realreal
```

fi

```
    local
s$loc   ent                                   entry point
        jsr  gtsmi                            get second argument (local number)
        err  134,local second                 argument is not integer
        ppm  exfal                            fail if out of range
        mov  xr,wb                            save local number
        mov  (xs)+,xr                         load first argument
        jsr  gtnvr                            point to vrblk
        ppm  sloc1                            jump if not variable name
        mov  vrfnc(xr),xr                     else load function pointer
        bne  (xr),=b$pfc,sloc1                jump if not program defined
    here if we have a program defined function name
        bze  wb,exfal                         fail if second arg is zero
        bgt  wb,pfnlo(xr),exfal               or too large
        add  fargs(xr),wb                     else adjust offset to include args
        wtb  wb                               convert to bytes
        add  wb,xr                            point to local pointer
        mov  pfagb(xr),xr                     load vrblk pointer
        brn  exvnm                            exit building nmblk
    here if first argument is no good
sloc1   erb  135,local first arg              is not a program function name
```

---

if **.cnld**

else

```
    load
s$lod   ent                                   entry point
        jsr   gtstg                           load library name
        err   136,load second argument        is not a string
        mov   xr,xl                           save library name
        jsr   xscni                           prepare to scan first argument
        err   137,load first argument         is not a string
        err   138,load first argument         is null
        mov   xl,-(xs)                        stack library name
        mov   =ch$pp,wc                       set delimiter one = left paren
        mov   wc,xl                           set delimiter two = left paren
        mnz   wa                              skip/trim blanks in prototype
        jsr   xscan                           scan function name
        mov   xr,-(xs)                        save ptr to function name
        bnz   wa,slod1                        jump if left paren found
        erb   139,load first argument         is missing a left paren
    here after successfully scanning function name
slod1   jsr   gtnvr                           locate vrblk
        err   140,load first argument         has null function name
        mov   xr,lodfn                        save vrblk pointer
        zer   lodna                           zero count of arguments
    loop to scan argument datatype names
slod2   mov   =ch$rp,wc                       delimiter one is right paren
        mov   =ch$cm,xl                       delimiter two is comma
        mnz   wa                              skip/trim blanks in prototype
        jsr   xscan                           scan next argument name
        icv   lodna                           bump argument count
        bnz   wa,slod3                        jump if ok delimiter was found
        erb   141,load first argument         is missing a right paren
```

```
    load (continued)
    come here to analyze the datatype pointer in (xr). this
    code is used both for arguments (wa=1,2) and for the
    result datatype (with wa set to zero).
```
---

if **.culc**

```
slod3   mov  wa,wb                  save scan mode
        mov  sclen(xr),wa           datatype length
        bze  wa,sld3a               bypass if null string
        jsr  flstg                  fold to upper case
sld3a   mov  wb,wa                  restore scan mode
        mov  xr,-(xs)               stack datatype name pointer
```

else

```
slod3   mov  xr,-(xs)               stack datatype name pointer
```

fi

```
        mov  =num01,wb              set string code in case
        mov  =scstr,xl              point to /string/
        jsr  ident                  check for match
        ppm  slod4                  jump if match
        mov  (xs),xr                else reload name
        add  wb,wb                  set code for integer (2)
        mov  =scint,xl              point to /integer/
        jsr  ident                  check for match
        ppm  slod4                  jump if match
```
---

if **.cnra**

else

```
        mov  (xs),xr                else reload string pointer
        icv  wb                     set code for real (3)
        mov  =screa,xl              point to /real/
        jsr  ident                  check for match
        ppm  slod4                  jump if match
```

fi

---

if **.cnlf**

```
        mov  (xs),xr                reload string pointer
        icv  wb                     code for file (4, or 3 if no reals)
        mov  =scfil,xl              point to /file/
        jsr  ident                  check for match
        ppm  slod4                  jump if match
```

fi

```
        zer  wb                     else get code for no convert
    merge here with proper datatype code in wb
slod4   mov  wb,(xs)                store code on stack
        beq  wa,=num02,slod2        loop back if arg stopped by comma
        bze  wa,slod5               jump if that was the result type
    here we scan out the result type (arg stopped by ) )
        mov  mxlen,wc               set dummy (impossible) delimiter 1
        mov  wc,xl                  and delimiter two
        mnz  wa                     skip/trim blanks in prototype
        jsr  xscan                  scan result name
        zer  wa                     set code for processing result
        brn  slod3                  jump back to process result name
```

```
    load (continued)
    here after processing all args and result
slod5   mov  lodna,wa                          get number of arguments
        mov  wa,wc                             copy for later
        wtb  wa                                convert length to bytes
        add  *efsi$,wa                         add space for standard fields
        jsr  alloc                             allocate efblk
        mov  =b$efc,(xr)                       set type word
        mov  wc,fargs(xr)                      set number of arguments
        zer  efuse(xr)                         set use count (dffnc will set to 1)
        zer  efcod(xr)                         zero code pointer for now
        mov  (xs)+,efrsl(xr)                   store result type code
        mov  lodfn,efvar(xr)                   store function vrblk pointer
        mov  wa,eflen(xr)                      store efblk length
        mov  xr,wb                             save efblk pointer
        add  wa,xr                             point past end of efblk
        lct  wc,wc                             set number of arguments for loop
    loop to set argument type codes from stack
slod6   mov  (xs)+,-(xr)                       store one type code from stack
        bct  wc,slod6                          loop till all stored
    now load the external function and perform definition
        mov  (xs)+,xr                          load function string name
```

---

if **.culc**

```
        mov  sclen(xr),wa                      function name length
        jsr  flstg                             fold to upper case
```

fi

```
        mov  (xs),xl                           load library name
        mov  wb,(xs)                           store efblk pointer
        jsr  sysld                             call function to load external func
        err  142,load function                 does not exist
        err  143,load function                 caused input error during load
        err  328,load function                 - insufficient memory
        mov  (xs)+,xl                          recall efblk pointer
        mov  xr,efcod(xl)                      store code pointer
        mov  lodfn,xr                          point to vrblk for function
        jsr  dffnc                             perform function definition
        brn  exnul                             return null result
```

fi

```
    lpad
s$lpd   ent                                       entry point
        jsr   gtstg                               get pad character
        err   144,lpad third argument             is not a string
        plc   xr                                  point to character (null is blank)
        lch   wb,(xr)                             load pad character
        jsr   gtsmi                               get pad length
        err   145,lpad second argument            is not integer
        ppm   slpd4                               skip if negative or large
    merge to check first arg
slpd1   jsr   gtstg                               get first argument (string to pad)
        err   146,lpad first argument             is not a string
        bge   wa,wc,exixr                         return 1st arg if too long to pad
        mov   xr,xl                               else move ptr to string to pad
    now we are ready for the pad
    (xl)                  pointer to string to pad
    (wb)                  pad character
    (wc)                  length to pad string to
        mov   wc,wa                               copy length
        jsr   alocs                               allocate scblk for new string
        mov   xr,-(xs)                            save as result
        mov   sclen(xl),wa                        load length of argument
        sub   wa,wc                               calculate number of pad characters
        psc   xr                                  point to chars in result string
        lct   wc,wc                               set counter for pad loop
    loop to perform pad
slpd2   sch   wb,(xr)+                            store pad character, bump ptr
        bct   wc,slpd2                            loop till all pad chars stored
        csc   xr                                  complete store characters
    now copy string
        bze   wa,slpd3                            exit if null string
        plc   xl                                  else point to chars in argument
        mvc                                       move characters to result string
        zer   xl                                  clear garbage xl
    here to exit with result on stack
slpd3   lcw   xr                                  load next code word
        bri   (xr)                                execute it
    here if 2nd arg is negative or large
slpd4   zer   wc                                  zero pad count
        brn   slpd1                               merge
```

```
    lt
s$ltf   ent                                  entry point
        jsr   acomp                          call arithmetic comparison routine
        err   147,lt first argument          is not numeric
        err   148,lt second argument         is not numeric
        ppm   exnul                          return null if lt
        ppm   exfal                          fail if eq
        ppm   exfal                          fail if gt
```

```
    ne
s$nef   ent                                   entry point
        jsr   acomp                           call arithmetic comparison routine
        err   149,ne first argument           is not numeric
        err   150,ne second argument          is not numeric
        ppm   exnul                           return null if lt
        ppm   exfal                           fail if eq
        ppm   exnul                           return null if gt
```

```
    notany
s$nay   ent                          entry point
        mov  =p$nas,wb               set pcode for single char arg
        mov  =p$nay,xl               pcode for multi-char arg
        mov  =p$nad,wc               set pcode for expr arg
        jsr  patst                   call common routine to build node
        err  151,notany argument     is not a string or expression
        mov  xr,-(xs)                stack result
        lcw  xr                      get next code word
        bri  (xr)                    execute it
```

```
    opsyn
s$ops   ent                                        entry point
        jsr   gtsmi                                load third argument
        err   152,opsyn third argument             is not integer
        err   153,opsyn third argument             is negative or too large
        mov   wc,wb                                if ok, save third argumnet
        mov   (xs)+,xr                             load second argument
        jsr   gtnvr                                locate variable block
        err   154,opsyn second                     arg is not natural variable name
        mov   vrfnc(xr),xl                         if ok, load function block pointer
        bnz   wb,sops2                             jump if operator opsyn case
    here for function opsyn (third arg zero)
        mov   (xs)+,xr                             load first argument
        jsr   gtnvr                                get vrblk pointer
        err   155,opsyn first arg                  is not natural variable name
    merge here to perform function definition
sops1   jsr   dffnc                                call function definer
        brn   exnul                                exit with null result
    here for operator opsyn (third arg non-zero)
sops2   jsr   gtstg                                get operator name
        ppm   sops5                                jump if not string
        bne   wa,=num01,sops5                      error if not one char long
        plc   xr                                   else point to character
        lch   wc,(xr)                              load character name
```

```
    opsyn (continued)
    now set to search for matching unary or binary operator
    name as appropriate. note that there are =opbun undefined
    binary operators and =opuun undefined unary operators.
        mov  =r$uub,wa                        point to unop pointers in case
        mov  =opnsu,xr                        point to names of unary operators
        add  =opbun,wb                        add no. of undefined binary ops
        beq  wb,=opuun,sops3                  jump if unop (third arg was 1)
        mov  =r$uba,wa                        else point to binary operator ptrs
        mov  =opsnb,xr                        point to names of binary operators
        mov  =opbun,wb                        set number of undefined binops
    merge here to check list (wb = number to check)
sops3   lct  wb,wb                            set counter to control loop
    loop to search for name match
sops4   beq  wc,(xr),sops6                    jump if names match
        ica  wa                               else push pointer to function ptr
        ica  xr                               bump pointer
        bct  wb,sops4                         loop back till all checked
    here if bad operator name
sops5   erb  156,opsyn first arg              is not correct operator name
    come here on finding a match in the operator name table
sops6   mov  wa,xr                            copy pointer to function block ptr
        sub  *vrfnc,xr                        make it look like dummy vrblk
        brn  sops1                            merge back to define operator
```

if **.c370**

```
    or
s$orf   ent                                 entry point
        mnz  wb                             signal two arguments
        jsr  sbool                          call string boolean routine
        err  xxx,or first argument          is not a string
        err  xxx,or second argument         is not a string
        err  xxx,or arguments               not same length
        ppm  exits                          null string arguments
    here to process (wc) words.  result is stacked.
sorf1   mov  (xl)+,wa                       get next cfp$c chars from arg 1
        orb  (xr),wa                        or with characters from arg 2
        mov  wa,(xr)+                       put back in memory
        bct  wc,sorf1                       loop over all words in string block
        brn  exits                          fetch next code word
```

fi

```
    output
s$oup   ent                              entry point
        mov  =num03,wb                   output flag
        jsr  ioput                       call input/output assoc. routine
        err  157,output third            argument is not a string
        err  158,inappropriate           second argument for output
        err  159,inappropriate           first argument for output
        err  160,inappropriate           file specification for output
        ppm  exfal                       fail if file does not exist
        err  161,output file cannot      be written to
        err  290,output channel          currently in use
        brn  exnul                       return null string
```

```
    pos
s$pos   ent                                 entry point
        mov  =p$pos,wb                      set pcode for integer arg case
        mov  =p$psd,wa                      set pcode for expression arg case
        jsr  patin                          call common routine to build node
        err  162,pos argument               is not integer or expression
        err  163,pos argument               is negative or too large
        mov  xr,-(xs)                       stack result
        lcw  xr                             get next code word
        bri  (xr)                           execute it
```

```
    prototype
s$pro   ent                                   entry point
        mov  (xs)+,xr                         load argument
        mov  tblen(xr),wb                     length if table, vector (=vclen)
        btw  wb                               convert to words
        mov  (xr),wa                          load type word of argument block
        beq  wa,=b$art,spro4                  jump if array
        beq  wa,=b$tbt,spro1                  jump if table
        beq  wa,=b$vct,spro3                  jump if vector
```

if **.cnbf**

else

```
        beq  wa,=b$bct,spr05                  jump if buffer
```

fi

```
        erb  164,prototype argument           is not valid object
    here for table
spro1   sub  =tbsi$,wb                        subtract standard fields
    merge for vector
spro2   mti  wb                               convert to integer
        brn  exint                            exit with integer result
    here for vector
spro3   sub  =vcsi$,wb                        subtract standard fields
        brn  spro2                            merge
    here for array
spro4   add  arofs(xr),xr                     point to prototype field
        mov  (xr),xr                          load prototype
        mov  xr,-(xs)                         stack result
        lcw  xr                               get next code word
        bri  (xr)                             execute it
```

if **.cnbf**

else

```
    here for buffer
spr05   mov  bcbuf(xr),xr                     point to bfblk
        mti  bfalc(xr)                        load allocated length
        brn  exint                            exit with integer allocation
```

fi

```
    remdr
s$rmd   ent                                       entry point
--------------------------------------------------------------------------
if .cmth
        jsr   arith                               get two integers or two reals
        err   166,remdr first argument            is not numeric
        err   165,remdr second                    argument is not numeric
        ppm   srm06                               if real
else
        mov   (xs),xr                             load second argument
        jsr   gtint                               convert to integer
        err   165,remdr second                    argument is not integer
        mov   xr,(xs)                             place converted arg in stack
        jsr   arith                               convert args
        ppm   srm04                               first arg not integer
        ppm                                       second arg checked above
--------------------------------------------------------------------------
if .cnra
else
        ppm   srm01                               first arg real
fi
fi
    both arguments integer
        zer   wb                                  set positive flag
        ldi   icval(xr)                           load left argument value
        ige   srm01                               jump if positive
        mnz   wb                                  set negative flag
srm01   rmi   icval(xl)                           get remainder
        iov   srm05                               error if overflow
    make sign of result match sign of first argument
        bze   wb,srm03                            if result should be positive
        ile   exint                               if should be negative, and is
srm02   ngi                                       adjust sign of result
        brn   exint                               return result
srm03   ilt   srm02                               should be pos, and result negative
        brn   exint                               should be positive, and is
    fail first argument
srm04   erb   166,remdr first argument            is not numeric
    fail if overflow
srm05   erb   167,remdr caused                    integer overflow
--------------------------------------------------------------------------
if .cmth
    here with 1st argument in (xr), 2nd in (xl), both real
    result = n1 - chop(n1/n2)*n2
srm06   zer   wb                                  set positive flag
        ldr   rcval(xr)                           load left argument value
        rge   srm07                               jump if positive
        mnz   wb                                  set negative flag
srm07   dvr   rcval(xl)                           compute n1/n2
        rov   srm10                               jump if overflow
        chp                                       chop result
        mlr   rcval(xl)                           times n2
        sbr   rcval(xr)                           compute difference
    make sign of result match sign of first argument
    -result is in ra at this point
```

```
        bze   wb,srm09                      if result should be positive
        rle   exrea                         if should be negative, and is
srm08   ngr                                 adjust sign of result
        brn   exrea                         return result
srm09   rlt   srm08                         should be pos, and result negative
        brn   exrea                         should be positive, and is
    fail if overflow
srm10   erb   312,remdr caused              real overflow
fi
```

```
    replace
    the actual replace operation uses an scblk whose cfp$a
    chars contain the translated versions of all the chars.
    the table pointer is remembered from call to call and
    the table is only built when the arguments change.
    we also perform an optimization gleaned from spitbol 370.
    if the second argument is &alphabet, there is no need to
    to build a replace table.  the third argument can be
    used directly as the replace table.
s$rpl   ent                                 entry point
        jsr  gtstg                          load third argument as string
        err  168,replace third              argument is not a string
        mov  xr,xl                          save third arg ptr
        jsr  gtstg                          get second argument
        err  169,replace second             argument is not a string
    check to see if this is the same table as last time
        bne  xr,r$ra2,srpl1                 jump if 2nd argument different
        beq  xl,r$ra3,srpl4                 jump if args same as last time
    here we build a new replace table (note wa = 2nd arg len)
srpl1   mov  sclen(xl),wb                   load 3rd argument length
        bne  wa,wb,srpl6                    jump if arguments not same length
        beq  xr,kvalp,srpl5                 jump if 2nd arg is alphabet string
        bze  wb,srpl6                       jump if null 2nd argument
        mov  xl,r$ra3                       save third arg for next time in
        mov  xr,r$ra2                       save second arg for next time in
        mov  kvalp,xl                       point to alphabet string
        mov  sclen(xl),wa                   load alphabet scblk length
        mov  r$rpt,xr                       point to current table (if any)
        bnz  xr,srpl2                       jump if we already have a table
    here we allocate a new table
        jsr  alocs                          allocate new table
        mov  wc,wa                          keep scblk length
        mov  xr,r$rpt                       save table pointer for next time
    merge here with pointer to new table block in (xr)
srpl2   ctb  wa,scsi$                       compute length of scblk
        mvw                                 copy to get initial table values
```

```
    replace (continued)
    now we must plug selected entries as required. note that
    we are short of index registers for the following loop.
    hence the need to repeatedly re-initialise char ptr xl
        mov  r$ra2,xl                 point to second argument
        lct  wb,wb                    number of chars to plug
        zer  wc                       zero char offset
        mov  r$ra3,xr                 point to 3rd arg
        plc  xr                       get char ptr for 3rd arg
    loop to plug chars
srpl3   mov  r$ra2,xl                 point to 2nd arg
        plc  xl,wc                    point to next char
        icv  wc                       increment offset
        lch  wa,(xl)                  get next char
        mov  r$rpt,xl                 point to translate table
        psc  xl,wa                    convert char to offset into table
        lch  wa,(xr)+                 get translated char
        sch  wa,(xl)                  store in table
        csc  xl                       complete store characters
        bct  wb,srpl3                 loop till done
```

```
    replace (continued)
    here to use r$rpt as replace table.
srpl4   mov  r$rpt,xl                        replace table to use
    here to perform translate using table in xl.
```

---

if **.cnbf**

```
srpl5   jsr  gtstg                           get first argument
        err  170,replace first               argument is not a string
```

else

```
    if first arg is a buffer, perform translate in place.
srpl5   jsr  gtstb                           get first argument
        err  170,replace first               argument is not a string or buffer
        bnz  wb,srpl7                        branch if buffer
```

fi

```
        bze  wa,exnul                        return null if null argument
        mov  xl,-(xs)                        stack replace table to use
        mov  xr,xl                           copy pointer
        mov  wa,wc                           save length
        ctb  wa,schar                        get scblk length
        jsr  alloc                           allocate space for copy
        mov  xr,wb                           save address of copy
        mvw                                  move scblk contents to copy
        mov  (xs)+,xr                        unstack replace table
        plc  xr                              point to chars of table
        mov  wb,xl                           point to string to translate
        plc  xl                              point to chars of string
        mov  wc,wa                           set number of chars to translate
        trc                                  perform translation
srpl8   mov  wb,-(xs)                        stack result
        lcw  xr                              load next code word
        bri  (xr)                            execute it
    error point
srpl6   erb  171,null or unequally           long 2nd, 3rd args to replace
```

---

if **.cnbf**

else

```
    here to perform replacement within buffer
srpl7   bze  wa,srpl8                        return buffer unchanged if empty
        mov  xr,wc                           copy bfblk pointer to wc
        mov  xl,xr                           translate table to xr
        plc  xr                              point to chars of table
        mov  wc,xl                           point to string to translate
        plc  xl                              point to chars of string
        trc                                  perform translation
        brn  srpl8                           stack result and exit
```

fi

```
    rewind
s$rew   ent                                     entry point
        jsr   iofcb                             call fcblk routine
        err   172,rewind argument               is not a suitable name
        err   173,rewind argument               is null
        err   174,rewind file does              not exist
        jsr   sysrw                             call system rewind function
        err   174,rewind file does              not exist
        err   175,rewind file does              not permit rewind
        err   176,rewind caused                 non-recoverable error
        brn   exnul                             exit with null result if no error
```

```
    reverse
s$rvs   ent                                         entry point
```

if **.cnbf**

```
        jsr   gtstg                                 load string argument
        err   177,reverse argument                  is not a string
```

else

```
        jsr   gtstb                                 load string or buffer argument
        err   177,reverse argument                  is not a string or buffer
        bnz   wb,srvs3                              branch if buffer
```

fi

```
        bze   wa,exixr                              return argument if null
        mov   xr,xl                                 else save pointer to string arg
        jsr   alocs                                 allocate space for new scblk
        mov   xr,-(xs)                              store scblk ptr on stack as result
        psc   xr                                    prepare to store in new scblk
        plc   xl,wc                                 point past last char in argument
        lct   wc,wc                                 set loop counter
    loop to move chars in reverse order
srvs1   lch   wb,-(xl)                              load next char from argument
        sch   wb,(xr)+                              store in result
        bct   wc,srvs1                              loop till all moved
    here when complete to execute next code word
srvs4   csc   xr                                    complete store characters
        zer   xl                                    clear garbage xl
srvs2   lcw   xr                                    load next code word
        bri   (xr)                                  execute it
```

if **.cnbf**

else

```
    here if argument is a buffer.  perform reverse in place.
srvs3   mov   wb,-(xs)                              stack buffer as result
        bze   wa,srvs2                              return buffer unchanged if empty
        mov   xr,xl                                 copy bfblk pointer to xl
        psc   xr                                    prepare to store at first char
        plc   xl,wa                                 point past last char in argument
        rsh   wa,1                                  operate on half the string
        lct   wc,wa                                 set loop counter
    loop to swap chars from end to end.  note that in the
    case of an odd count, the middle char is not touched.
srvs5   lch   wb,-(xl)                              load next char from end
        lch   wa,(xr)                               load next char from front
        sch   wb,(xr)+                              store end char in front
        sch   wa,(xl)                               store front char at end
        bct   wc,srvs5                              loop till all moved
        brn   srvs4                                 complete store
```

fi

```
    rpad
s$rpd   ent                                         entry point
        jsr   gtstg                                 get pad character
        err   178,rpad third argument               is not a string
        plc   xr                                    point to character (null is blank)
        lch   wb,(xr)                               load pad character
        jsr   gtsmi                                 get pad length
        err   179,rpad second argument              is not integer
        ppm   srpd3                                 skip if negative or large
    merge to check first arg.
srpd1   jsr   gtstg                                 get first argument (string to pad)
        err   180,rpad first argument               is not a string
        bge   wa,wc,exixr                           return 1st arg if too long to pad
        mov   xr,xl                                 else move ptr to string to pad
    now we are ready for the pad
    (xl)                  pointer to string to pad
    (wb)                  pad character
    (wc)                  length to pad string to
        mov   wc,wa                                 copy length
        jsr   alocs                                 allocate scblk for new string
        mov   xr,-(xs)                              save as result
        mov   sclen(xl),wa                          load length of argument
        sub   wa,wc                                 calculate number of pad characters
        psc   xr                                    point to chars in result string
        lct   wc,wc                                 set counter for pad loop
    copy argument string
        bze   wa,srpd2                              jump if argument is null
        plc   xl                                    else point to argument chars
        mvc                                         move characters to result string
        zer   xl                                    clear garbage xl
    loop to supply pad characters
srpd2   sch   wb,(xr)+                              store pad character, bump ptr
        bct   wc,srpd2                              loop till all pad chars stored
        csc   xr                                    complete character storing
        lcw   xr                                    load next code word
        bri   (xr)                                  execute it
    here if 2nd arg is negative or large
srpd3   zer   wc                                    zero pad count
        brn   srpd1                                 merge
```

```
    rtab
s$rtb   ent                           entry point
        mov  =p$rtb,wb                set pcode for integer arg case
        mov  =p$rtd,wa                set pcode for expression arg case
        jsr  patin                    call common routine to build node
        err  181,rtab argument        is not integer or expression
        err  182,rtab argument        is negative or too large
        mov  xr,-(xs)                 stack result
        lcw  xr                       get next code word
        bri  (xr)                     execute it
```

```
if .cust
    set
s$set   ent                                       entry point
        mov  (xs)+,r$io2                          save third arg (whence)
```

```
if .cusr
        mov  (xs)+,xr                             get second arg (offset)
        jsr  gtrea                                convert to real
        err  324,set second argument              not numeric
        ldr  rcval(xr)                            load accumulator with argument
else
        mov  (xs)+,r$io1                          save second arg (offset)
fi
        jsr  iofcb                                call fcblk routine
        err  291,set first argument               is not a suitable name
        err  292,set first argument               is null
        err  295,set file does                    not exist
```

```
if .cusr
else
        mov  r$io1,wb                             load second arg
fi
        mov  r$io2,wc                             load third arg
        jsr  sysst                                call system set routine
        err  293,inappropriate                    second argument to set
        err  294,inappropriate                    third argument to set
        err  295,set file does                    not exist
        err  296,set file does                    not permit setting file pointer
        err  297,set caused non-recoverable       i/o error
```

```
if .cusr
        rti  exrea                                return real position if not able
        brn  exint                                to return integer position
else
        brn  exint                                otherwise return position
fi
```

fi

```
    tab
s$tab   ent                              entry point
        mov  =p$tab,wb                   set pcode for integer arg case
        mov  =p$tbd,wa                   set pcode for expression arg case
        jsr  patin                       call common routine to build node
        err  183,tab argument            is not integer or expression
        err  184,tab argument            is negative or too large
        mov  xr,-(xs)                    stack result
        lcw  xr                          get next code word
        bri  (xr)                        execute it
```

```
    rpos
s$rps   ent                                   entry point
        mov   =p$rps,wb                       set pcode for integer arg case
        mov   =p$rpd,wa                       set pcode for expression arg case
        jsr   patin                           call common routine to build node
        err   185,rpos argument               is not integer or expression
        err   186,rpos argument               is negative or too large
        mov   xr,-(xs)                        stack result
        lcw   xr                              get next code word
        bri   (xr)                            execute it
```

---

if **.cnsr**
else

```
    rsort
s$rsr   ent                                  entry point
        mnz   wa                             mark as rsort
        jsr   sorta                          call sort routine
        ppm   exfal                          if conversion fails, so shall we
        brn   exsid                          return, setting idval
fi
```

```
    setexit
s$stx   ent                                 entry point
        mov  (xs)+,xr                       load argument
        mov  stxvr,wa                       load old vrblk pointer
        zer  xl                             load zero in case null arg
        beq  xr,=nulls,sstx1                jump if null argument (reset call)
        jsr  gtnvr                          else get specified vrblk
        ppm  sstx2                          jump if not natural variable
        mov  vrlbl(xr),xl                   else load label
        beq  xl,=stndl,sstx2                jump if label is not defined
        bne  (xl),=b$trt,sstx1              jump if not trapped
        mov  trlbl(xl),xl                   else load ptr to real label code
    here to set/reset setexit trap
sstx1   mov  xr,stxvr                       store new vrblk pointer (or null)
        mov  xl,r$sxc                       store new code ptr (or zero)
        beq  wa,=nulls,exnul                return null if null result
        mov  wa,xr                          else copy vrblk pointer
        brn  exvnm                          and return building nmblk
    here if bad argument
sstx2   erb  187,setexit argument           is not label name or null
```

---

if **.cmth**

```
    sin
s$sin   ent                                 entry point
        mov  (xs)+,xr                       get argument
        jsr  gtrea                          convert to real
        err  308,sin argument               not numeric
        ldr  rcval(xr)                      load accumulator with argument
        sin                                 take sine
        rno  exrea                          if no overflow, return result in ra
        erb  323,sin argument               is out of range
```

fi

---

if **.cmth**

```
    sqrt
s$sqr   ent                              entry point
        mov  (xs)+,xr                    get argument
        jsr  gtrea                       convert to real
        err  313,sqrt argument           not numeric
        ldr  rcval(xr)                   load accumulator with argument
        rlt  ssqr1                       negative number
        sqr                              take square root
        brn  exrea                       no overflow possible, result in ra
    here if bad argument
ssqr1   erb  314,sqrt argument           negative
```

fi

---

if **.cnsr**
else

```
    sort
s$srt   ent                 entry point
        zer   wa            mark as sort
        jsr   sorta         call sort routine
        ppm   exfal         if conversion fails, so shall we
        brn   exsid         return, setting idval
fi
```

```
    span
s$spn   ent                          entry point
        mov  =p$sps,wb               set pcode for single char arg
        mov  =p$spn,xl               set pcode for multi-char arg
        mov  =p$spd,wc               set pcode for expression arg
        jsr  patst                   call common routine to build node
        err  188,span argument       is not a string or expression
        mov  xr,-(xs)                stack result
        lcw  xr                      get next code word
        bri  (xr)                    execute it
```

```
    size
s$si$   ent                                     entry point
```

```
if .cnbf
        jsr   gtstg                             load string argument
        err   189,size argument                 is not a string
else
        jsr   gtstb                             load string argument
        err   189,size argument                 is not a string or buffer
fi
    merge with bfblk or scblk ptr in xr.  wa has length.
        mti   wa                                load length as integer
        brn   exint                             exit with integer result
```

```
    stoptr
s$stt   ent                               entry point
        zer   xl                          indicate stoptr case
        jsr   trace                       call trace procedure
        err   190,stoptr first            argument is not appropriate name
        err   191,stoptr second           argument is not trace type
        brn   exnul                       return null
```

```
    substr
s$sub   ent                                  entry point
        jsr   gtsmi                          load third argument
        err   192,substr third               argument is not integer
        ppm   exfal                          jump if negative or too large
        mov   xr,sbssv                       save third argument
        jsr   gtsmi                          load second argument
        err   193,substr second              argument is not integer
        ppm   exfal                          jump if out of range
        mov   xr,wc                          save second argument
        bze   wc,exfal                       jump if second argument zero
        dcv   wc                             else decrement for ones origin
```

---

if **.cnbf**

```
        jsr   gtstg                          load first argument
        err   194,substr first               argument is not a string
```

else

```
        jsr   gtstb                          load first argument
        err   194,substr first               argument is not a string or buffer
```

fi

```
    merge with bfblk or scblk ptr in xr.  wa has length
        mov   wc,wb                          copy second arg to wb
        mov   sbssv,wc                       reload third argument
        bnz   wc,ssub2                       skip if third arg given
        mov   wa,wc                          else get string length
        bgt   wb,wc,exfal                    fail if improper
        sub   wb,wc                          reduce by offset to start
    merge
ssub2   mov   wa,xl                          save string length
        mov   wc,wa                          set length of substring
        add   wb,wc                          add 2nd arg to 3rd arg
        bgt   wc,xl,exfal                    jump if improper substring
        mov   xr,xl                          copy pointer to first arg
        jsr   sbstr                          build substring
        mov   xr,-(xs)                       stack result
        lcw   xr                             get next code word
        bri   (xr)                           execute it
```

```
    table
s$tbl   ent                                   entry point
        mov  (xs)+,xl                         get initial lookup value
        ica  xs                               pop second argument
        jsr  gtsmi                            load argument
        err  195,table argument               is not integer
        err  196,table argument               is out of range
        bnz  wc,stbl1                         jump if non-zero
        mov  =tbnbk,wc                        else supply default value
    merge here with number of headers in wc
stbl1   jsr  tmake                            make table
        brn  exsid                            exit setting idval
```

if **.cmth**

```
    tan
s$tan   ent                                  entry point
        mov  (xs)+,xr                        get argument
        jsr  gtrea                           convert to real
        err  309,tan argument                not numeric
        ldr  rcval(xr)                       load accumulator with argument
        tan                                  take tangent
        rno  exrea                           if no overflow, return result in ra
        erb  310,tan produced                real overflow or argument is out of range
```

fi

```
    time
s$tim   ent                                   entry point
        jsr   systm                           get timer value
        sbi   timsx                           subtract starting time
        brn   exint                           exit with integer value
```

```
    trace
s$tra   ent                                      entry point
        beq  num03(xs),=nulls,str02              jump if first argument is null
        mov  (xs)+,xr                            load fourth argument
        zer  xl                                  tentatively set zero pointer
        beq  xr,=nulls,str01                     jump if 4th argument is null
        jsr  gtnvr                               else point to vrblk
        ppm  str03                               jump if not variable name
        mov  xr,xl                               else save vrblk in trfnc
    here with vrblk or zero in xl
str01   mov  (xs)+,xr                            load third argument (tag)
        zer  wb                                  set zero as trtyp value for now
        jsr  trbld                               build trblk for trace call
        mov  xr,xl                               move trblk pointer for trace
        jsr  trace                               call trace procedure
        err  198,trace first argument            is not appropriate name
        err  199,trace second                    argument is not trace type
        brn  exnul                               return null
    here to call system trace toggle routine
str02   jsr  systt                               call it
        add  *num04,xs                           pop trace arguments
        brn  exnul                               return
    here for bad fourth argument
str03   erb  197,trace fourth                    arg is not function name or null
```

```
    trim
s$trm   ent                                      entry point
---------------------------------------------------------------------------
if .cnbf
        jsr   gtstg                              load argument as string
        err   200,trim argument                  is not a string
else
        jsr   gtstb                              load argument as string
        err   200,trim argument                  is not a string or buffer
        bnz   wb,strm0                           branch if buffer
fi
        bze   wa,exnul                           return null if argument is null
        mov   xr,xl                              copy string pointer
        ctb   wa,schar                           get block length
        jsr   alloc                              allocate copy same size
        mov   xr,wb                              save pointer to copy
        mvw                                      copy old string block to new
        mov   wb,xr                              restore ptr to new block
        jsr   trimr                              trim blanks (wb is non-zero)
        mov   xr,-(xs)                           stack result
        lcw   xr                                 get next code word
        bri   (xr)                               execute it
---------------------------------------------------------------------------
if .cnbf
else
    argument is a buffer, perform trim in place.
strm0   mov   wb,-(xs)                           stack buffer as result
        bze   wa,strm6                           return buffer unchanged if empty
        mov   xr,xl                              get bfblk ptr
        mov   wb,xr                              copy bcblk ptr to xr
        plc   xl,wa                              point past last character
        mov   =ch$bl,wc                          load blank character
    loop through characters from right to left
strm1   lch   wb,-(xl)                           load next character
---------------------------------------------------------------------------
if .caht
        beq   wb,=ch$ht,strm2                    jump if horizontal tab
fi
        bne   wb,wc,strm3                        jump if non-blank found
strm2   dcv   wa                                 else decrement character count
        bnz   wa,strm1                           loop back if more to check
    here when buffer trim complete
strm3   mov   wa,bclen(xr)                       set new length in bcblk
        mov   bcbuf(xr),xr                       get bfblk ptr
        mov   wa,wb                              copy length
        ctb   wb,0                               words needed converted to bytes
        sub   wa,wb                              number of zeros needed
        psc   xr,wa                              ready for storing zeros
        zer   wc                                 set zero char
    loop to zero pad last word of characters
strm4   bze   wb,strm5                           loop while more to be done
        sch   wc,(xr)+                           store zero character
        dcv   wb                                 decrement count
        brn   strm4                              continue loop
strm5   csc   xr                                 complete store characters
```

```
strm6   lcw   xr                    get next code word
        bri   (xr)                  execute it
fi
```

```
    unload
s$unl   ent                                 entry point
        mov   (xs)+,xr                      load argument
        jsr   gtnvr                         point to vrblk
        err   201,unload argument           is not natural variable name
        mov   =stndf,xl                     get ptr to undefined function
        jsr   dffnc                         undefine named function
        brn   exnul                         return null as result
```

if **.c370**

```
   xor
s$xor   ent                                     entry point
        mnz  wb                                 signal two arguments
        jsr  sbool                              call string boolean routine
        err  xxx,xor first argument             is not a string
        err  xxx,xor second argument            is not a string
        err  xxx,xor arguments                  not same length
        ppm  exits                              null string arguments
   here to process (wc) words.  result is stacked.
sxor1   mov  (xl)+,wa                           get next cfp$c chars from arg 1
        xob  (xr),wa                            xor with characters from arg 2
        mov  wa,(xr)+                           put back in memory
        bct  wc,sxor1                           loop over all words in string block
        brn  exits                              fetch next code word
fi
```

```
the following section contains utility routines used for
various purposes throughout the system. these differ
from the procedures in the utility procedures section in
they are not in procedure form and they do not return
to their callers. they are accessed with a branch type
instruction after setting the registers to appropriate
parameter values.
the register values required for each routine are
documented at the start of each routine. registers not
mentioned may contain any values except that xr,xl
can only contain proper collectable pointers.
some of these routines will tolerate garbage pointers
in xl,xr on entry. this is always documented and in
each case, the routine clears these garbage values before
exiting after completing its task.
the routines have names consisting of five letters
and are assembled in alphabetical order.
```

```
    arref -- array reference
    (xl)                 may be non-collectable
    (xr)                 number of subscripts
    (wb)                 set zero/nonzero for value/name
                         the value in wb must be collectable
    stack                subscripts and array operand
    brn  arref           jump to call function
    arref continues by executing the next code word with
    the result name or value placed on top of the stack.
    to deal with the problem of accessing subscripts in the
    order of stacking, xl is used as a subscript pointer
    working below the stack pointer.
arref   rtn   exits
        mov   xr,wa                        copy number of subscripts
        mov   xs,xt                        point to stack front
        wtb   xr                           convert to byte offset
        add   xr,xt                        point to array operand on stack
        ica   xt                           final value for stack popping
        mov   xt,arfxs                     keep for later
        mov   -(xt),xr                     load array operand pointer
        mov   xr,r$arf                     keep array pointer
        mov   xt,xr                        save pointer to subscripts
        mov   r$arf,xl                     point xl to possible vcblk or tbblk
        mov   (xl),wc                      load first word
        beq   wc,=b$art,arf01              jump if arblk
        beq   wc,=b$vct,arf07              jump if vcblk
        beq   wc,=b$tbt,arf10              jump if tbblk
        erb   235,subscripted operand      is not table or array
    here for array (arblk)
arf01   bne   wa,arndm(xl),arf09           jump if wrong number of dims
        ldi   intv0                        get initial subscript of zero
        mov   xr,xt                        point before subscripts
        zer   wa                           initial offset to bounds
        brn   arf03                        jump into loop
    loop to compute subscripts by multiplications
arf02   mli   ardm2(xr)                    multiply total by next dimension
    merge here first time
arf03   mov   -(xt),xr                     load next subscript
        sti   arfsi                        save current subscript
        ldi   icval(xr)                    load integer value in case
        beq   (xr),=b$icl,arf04            jump if it was an integer
```

```
    arref (continued)
        jsr   gtint                         convert to integer
        ppm   arf12                         jump if not integer
        ldi   icval(xr)                     if ok, load integer value
    here with integer subscript in (ia)
arf04   mov   r$arf,xr                      point to array
        add   wa,xr                         offset to next bounds
        sbi   arlbd(xr)                     subtract low bound to compare
        iov   arf13                         out of range fail if overflow
        ilt   arf13                         out of range fail if too small
        sbi   ardim(xr)                     subtract dimension
        ige   arf13                         out of range fail if too large
        adi   ardim(xr)                     else restore subscript offset
        adi   arfsi                         add to current total
        add   *ardms,wa                     point to next bounds
        bne   xt,xs,arf02                   loop back if more to go
    here with integer subscript computed
        mfi   wa                            get as one word integer
        wtb   wa                            convert to offset
        mov   r$arf,xl                      point to arblk
        add   arofs(xl),wa                  add offset past bounds
        ica   wa                            adjust for arpro field
        bnz   wb,arf08                      exit with name if name call
    merge here to get value for value call
arf05   jsr   acess                         get value
        ppm   arf13                         fail if acess fails
    return value
arf06   mov   arfxs,xs                      pop stack entries
        zer   r$arf                         finished with array pointer
        mov   xr,-(xs)                      stack result
        lcw   xr                            get next code word
        bri   (xr)                          execute it
```

```
    arref (continued)
    here for vector
arf07   bne   wa,=num01,arf09              error if more than 1 subscript
        mov   (xs),xr                      else load subscript
        jsr   gtint                        convert to integer
        ppm   arf12                        error if not integer
        ldi   icval(xr)                    else load integer value
        sbi   intv1                        subtract for ones offset
        mfi   wa,arf13                     get subscript as one word
        add   =vcvls,wa                    add offset for standard fields
        wtb   wa                           convert offset to bytes
        bge   wa,vclen(xl),arf13           fail if out of range subscript
        bze   wb,arf05                     back to get value if value call
    return name
arf08   mov   arfxs,xs                     pop stack entries
        zer   r$arf                        finished with array pointer
        brn   exnam                        else exit with name
    here if subscript count is wrong
arf09   erb   236,array referenced         with wrong number of subscripts
    table
arf10   bne   wa,=num01,arf11              error if more than 1 subscript
        mov   (xs),xr                      else load subscript
        jsr   tfind                        call table search routine
        ppm   arf13                        fail if failed
        bnz   wb,arf08                     exit with name if name call
        brn   arf06                        else exit with value
    here for bad table reference
arf11   erb   237,table referenced         with more than one subscript
    here for bad subscript
arf12   erb   238,array subscript          is not integer
    here to signal failure
arf13   zer   r$arf                        finished with array pointer
        brn   exfal                        fail
```

```
    cfunc -- call a function
    cfunc is used to call a snobol level function. it is
    used by the apply function (s$app), the function
    trace routine (trxeq) and the main function call entry
    (o$fnc, o$fns). in the latter cases, cfunc is used only
    if the number of arguments is incorrect.
    (xl)                 pointer to function block
    (wa)                 actual number of arguments
    (xs)                 points to stacked arguments
    brn  cfunc           jump to call function
    cfunc continues by executing the function
cfunc   rtn  exfal
        blt  wa,fargs(xl),cfnc1            jump if too few arguments
        beq  wa,fargs(xl),cfnc3            jump if correct number of args
    here if too many arguments supplied, pop them off
        mov  wa,wb                         copy actual number
        sub  fargs(xl),wb                  get number of extra args
        wtb  wb                            convert to bytes
        add  wb,xs                         pop off unwanted arguments
        brn  cfnc3                         jump to go off to function
    here if too few arguments
cfnc1   mov  fargs(xl),wb                  load required number of arguments
        beq  wb,=nini9,cfnc3               jump if case of var num of args
        sub  wa,wb                         calculate number missing
        lct  wb,wb                         set counter to control loop
    loop to supply extra null arguments
cfnc2   mov  =nulls,-(xs)                  stack a null argument
        bct  wb,cfnc2                      loop till proper number stacked
    merge here to jump to function
cfnc3   bri  (xl)                          jump through fcode field
```

```
    exfal -- exit signalling snobol failure
    (xl,xr)              may be non-collectable
    brn  exfal           jump to fail
    exfal continues by executing the appropriate fail goto
exfal  rtn  (xl)
       mov  flptr,xs                        pop stack
       mov  (xs),xr                         load failure offset
       add  r$cod,xr                        point to failure code location
       lcp  xr                              set code pointer
       lcw  xr                              load next code word
       mov  (xr),xl                         load entry address
       bri  xl                              jump to execute next code word
```

```
    exint -- exit with integer result
    (xl,xr)              may be non-collectable
    (ia)                 integer value
    brn  exint           jump to exit with integer
    exint continues by executing the next code word
    which it does by falling through to exixr
exint   rtn   xl
        zer   xl                                     clear dud value
        jsr   icbld                                  build icblk
```

```
    exixr -- exit with result in (xr)
    (xr)                  result
    (xl)                  may be non-collectable
    brn  exixr            jump to exit with result in (xr)
    exixr continues by executing the next code word
    which it does by falling through to exits.
exixr   rtn   icbld                                       which it does by falling through to exits.
        mov   xr,-(xs)                                    stack result
    exits -- exit with result if any stacked
    (xr,xl)               may be non-collectable
    brn  exits            enter exits routine
exits   rtn   xr,-(xs)
        lcw   xr                                          load next code word
        mov   (xr),xl                                     load entry address
        bri   xl                                          jump to execute next code word
```

```
    exnam -- exit with name in (xl,wa)
    (xl)                 name base
    (wa)                 name offset
    (xr)                 may be non-collectable
    brn  exnam           jump to exit with name in (xl,wa)
    exnam continues by executing the next code word
exnam   rtn   xl
        mov   xl,-(xs)                                 stack name base
        mov   wa,-(xs)                                 stack name offset
        lcw   xr                                       load next code word
        bri   (xr)                                     execute it
```

```
    exnul -- exit with null result
    (xl,xr)              may be non-collectable
    brn  exnul           jump to exit with null value
    exnul continues by executing the next code word
exnul   rtn   (xr)
        mov   =nulls,-(xs)                      stack null value
        lcw   xr                                load next code word
        mov   (xr),xl                           load entry address
        bri   xl                                jump to execute next code word
```

if **.cnra**
else

```
    exrea -- exit with real result
    (xl,xr)              may be non-collectable
    (ra)                 real value
    brn  exrea           jump to exit with real value
    exrea continues by executing the next code word
exrea   rtn   xl
        zer   xl                                        clear dud value
        jsr   rcbld                                     build rcblk
        brn   exixr                                     jump to exit with result in xr
```

fi

```
    exsid -- exit setting id field
    exsid is used to exit after building any of the following
    blocks (arblk, tbblk, pdblk, vcblk). it sets the idval.
    (xr)                 ptr to block with idval field
    (xl)                 may be non-collectable
    brn  exsid           jump to exit after setting id field
    exsid continues by executing the next code word
exsid   rtn   exixr
        mov   curid,wa                  load current id value
        bne   wa,=cfp$m,exsi1           jump if no overflow
        zer   wa                        else reset for wraparound
    here with old idval in wa
exsi1   icv   wa                        bump id value
        mov   wa,curid                  store for next time
        mov   wa,idval(xr)              store id value
        brn   exixr                     exit with result in (xr)
```

```
    exvnm -- exit with name of variable
    exvnm exits after stacking a value which is a nmblk
    referencing the name of a given natural variable.
    (xr)                  vrblk pointer
    (xl)                  may be non-collectable
    brn  exvnm            exit with vrblk pointer in xr
exvnm   rtn   exixr
        mov   xr,xl                          copy name base pointer
        mov   *nmsi$,wa                      set size of nmblk
        jsr   alloc                          allocate nmblk
        mov   =b$nml,(xr)                    store type word
        mov   xl,nmbas(xr)                   store name base
        mov   *vrval,nmofs(xr)               store name offset
        brn   exixr                          exit with result in xr
```

```
    flpop -- fail and pop in pattern matching
    flpop pops the node and cursor on the stack and then
    drops through into failp to cause pattern failure
    (xl,xr)              may be non-collectable
    brn  flpop           jump to fail and pop stack
flpop   rtn   exixr
        add   *num02,xs                        pop two entries off stack
```

```
    failp -- failure in matching pattern node
    failp is used after failing to match a pattern node.
    see pattern match routines for details of use.
    (xl,xr)              may be non-collectable
    brn  failp           signal failure to match
    failp continues by matching an alternative from the stack
failp   rtn   *num02,xs
        mov   (xs)+,xr                load alternative node pointer
        mov   (xs)+,wb                restore old cursor
        mov   (xr),xl                 load pcode entry pointer
        bri   xl                      jump to execute code for node
```

```
    indir -- compute indirect reference
    (wb)                  nonzero/zero for by name/value
    brn  indir            jump to get indirect ref on stack
    indir continues by executing the next code word
indir   rtn   xl
        mov   (xs)+,xr                              load argument
        beq   (xr),=b$nml,indr2                     jump if a name
        jsr   gtnvr                                 else convert to variable
        err   239,indirection operand               is not name
        bze   wb,indr1                              skip if by value
        mov   xr,-(xs)                              else stack vrblk ptr
        mov   *vrval,-(xs)                          stack name offset
        lcw   xr                                    load next code word
        mov   (xr),xl                               load entry address
        bri   xl                                    jump to execute next code word
    here to get value of natural variable
indr1   bri   (xr)                                  jump through vrget field of vrblk
    here if operand is a name
indr2   mov   nmbas(xr),xl                          load name base
        mov   nmofs(xr),wa                          load name offset
        bnz   wb,exnam                              exit if called by name
        jsr   acess                                 else get value first
        ppm   exfal                                 fail if access fails
        brn   exixr                                 else return with value in xr
```

```
    match -- initiate pattern match
    (wb)                  match type code
    brn  match            jump to initiate pattern match
    match continues by executing the pattern match. see
    pattern match routines (p$xxx) for full details.
match   rtn   exixr
        mov   (xs)+,xr                          load pattern operand
        jsr   gtpat                             convert to pattern
        err   240,pattern match                 right operand is not pattern
        mov   xr,xl                             if ok, save pattern pointer
        bnz   wb,mtch1                          jump if not match by name
        mov   (xs),wa                           else load name offset
        mov   xl,-(xs)                          save pattern pointer
        mov   num02(xs),xl                      load name base
        jsr   acess                             access subject value
        ppm   exfal                             fail if access fails
        mov   (xs),xl                           restore pattern pointer
        mov   xr,(xs)                           stack subject string val for merge
        zer   wb                                restore type code
    merge here with subject value on stack
```

```
if .cnbf
mtch1   jsr   gtstg                             convert subject to string
        err   241,pattern match                 left operand is not a string
        mov   wb,-(xs)                          stack match type code
else
mtch1   mov   wb,wc                             save match type in wc
        jsr   gtstb                             convert subject to string
        err   241,pattern match                 left operand is not a string or buffer
        mov   wb,r$pmb                          set to zero/bcblk if string/buffer
        mov   wc,-(xs)                          stack match type code
fi
        mov   xr,r$pms                          if ok, store subject string pointer
        mov   wa,pmssl                          and length
        zer   -(xs)                             stack initial cursor (zero)
        zer   wb                                set initial cursor
        mov   xs,pmhbs                          set history stack base ptr
        zer   pmdfl                             reset pattern assignment flag
        mov   xl,xr                             set initial node pointer
        bnz   kvanc,mtch2                       jump if anchored
    here for unanchored
        mov   xr,-(xs)                          stack initial node pointer
        mov   =nduna,-(xs)                      stack pointer to anchor move node
        bri   (xr)                              start match of first node
    here in anchored mode
mtch2   zer   -(xs)                             dummy cursor value
        mov   =ndabo,-(xs)                      stack pointer to abort node
        bri   (xr)                              start match of first node
```

```
    retrn -- return from function
    (wa)                 string pointer for return type
    brn  retrn           jump to return from (snobol) func
    retrn continues by executing the code at the return point
    the stack is cleaned of any garbage left by other
    routines which may have altered flptr since function
    entry by using flprt, reserved for use only by
    function call and return.
retrn   rtn  (xr)
        bnz  kvfnc,rtn01                    jump if not level zero
        erb  242,function return            from level zero
    here if not level zero return
rtn01   mov  flprt,xs                       pop stack
        ica  xs                             remove failure offset
        mov  (xs)+,xr                       pop pfblk pointer
        mov  (xs)+,flptr                    pop failure pointer
        mov  (xs)+,flprt                    pop old flprt
        mov  (xs)+,wb                       pop code pointer offset
        mov  (xs)+,wc                       pop old code block pointer
        add  wc,wb                          make old code pointer absolute
        lcp  wb                             restore old code pointer
        mov  wc,r$cod                       restore old code block pointer
        dcv  kvfnc                          decrement function level
        mov  kvtra,wb                       load trace
        add  kvftr,wb                       add ftrace
        bze  wb,rtn06                       jump if no tracing possible
    here if there may be a trace
        mov  wa,-(xs)                       save function return type
        mov  xr,-(xs)                       save pfblk pointer
        mov  wa,kvrtn                       set rtntype for trace function
        mov  r$fnc,xl                       load fnclevel trblk ptr (if any)
        jsr  ktrex                          execute possible fnclevel trace
        mov  pfvbl(xr),xl                   load vrblk ptr (sgd13)
        bze  kvtra,rtn02                    jump if trace is off
        mov  pfrtr(xr),xr                   else load return trace trblk ptr
        bze  xr,rtn02                       jump if not return traced
        dcv  kvtra                          else decrement trace count
        bze  trfnc(xr),rtn03                jump if print trace
        mov  *vrval,wa                      else set name offset
        mov  num01(xs),kvrtn                make sure rtntype is set right
        jsr  trxeq                          execute full trace
```

```
    retrn (continued)
    here to test for ftrace
rtn02   bze   kvftr,rtn05                 jump if ftrace is off
        dcv   kvftr                       else decrement ftrace
    here for print trace of function return
rtn03   jsr   prtsn                       print statement number
        mov   num01(xs),xr                load return type
        jsr   prtst                       print it
        mov   =ch$bl,wa                   load blank
        jsr   prtch                       print it
        mov   0(xs),xl                    load pfblk ptr
        mov   pfvbl(xl),xl                load function vrblk ptr
        mov   *vrval,wa                   set vrblk name offset
        bne   xr,=scfrt,rtn04             jump if not freturn case
    for freturn, just print function name
        jsr   prtnm                       print name
        jsr   prtnl                       terminate print line
        brn   rtn05                       merge
    here for return or nreturn, print function name = value
rtn04   jsr   prtnv                       print name = value
    here after completing trace
rtn05   mov   (xs)+,xr                    pop pfblk pointer
        mov   (xs)+,wa                    pop return type string
    merge here if no trace required
rtn06   mov   wa,kvrtn                    set rtntype keyword
        mov   pfvbl(xr),xl                load pointer to fn vrblk
```

```
    retrn (continued)
    get value of function
rtn07   mov  xl,rtnbp                    save block pointer
        mov  vrval(xl),xl                load value
        beq  (xl),=b$trt,rtn07           loop back if trapped
        mov  xl,rtnfv                    else save function result value
        mov  (xs)+,rtnsv                 save original function value
```

---

if **.cnpf**

```
        mov  fargs(xr),wb                get number of arguments
```

else

```
        mov  (xs)+,xl                    pop saved pointer
        bze  xl,rtn7c                    no action if none
        bze  kvpfl,rtn7c                 jump if no profiling
        jsr  prflu                       else profile last func stmt
        beq  kvpfl,=num02,rtn7a          branch on value of profile keywd
    here if &profile = 1. start time must be frigged to
    appear earlier than it actually is, by amount used before
    the call.
        ldi  pfstm                       load current time
        sbi  icval(xl)                   frig by subtracting saved amount
        brn  rtn7b                       and merge
    here if &profile = 2
rtn7a   ldi  icval(xl)                   load saved time
    both profile types merge here
rtn7b   sti  pfstm                       store back correct start time
    merge here if no profiling
rtn7c   mov  fargs(xr),wb                get number of args
```

fi

```
        add  pfnlo(xr),wb                add number of locals
        bze  wb,rtn10                    jump if no args/locals
        lct  wb,wb                       else set loop counter
        add  pflen(xr),xr                and point to end of pfblk
    loop to restore functions and locals
rtn08   mov  -(xr),xl                    load next vrblk pointer
    loop to find value block
rtn09   mov  xl,wa                       save block pointer
        mov  vrval(xl),xl                load pointer to next value
        beq  (xl),=b$trt,rtn09           loop back if trapped
        mov  wa,xl                       else restore last block pointer
        mov  (xs)+,vrval(xl)             restore old variable value
        bct  wb,rtn08                    loop till all processed
    now restore function value and exit
rtn10   mov  rtnbp,xl                    restore ptr to last function block
        mov  rtnsv,vrval(xl)             restore old function value
        mov  rtnfv,xr                    reload function result
        mov  r$cod,xl                    point to new code block
        mov  kvstn,kvlst                 set lastno from stno
        mov  cdstm(xl),kvstn             reset proper stno value
```

---

if **.csln**

```
        mov  kvlin,kvlln                 set lastline from line
        mov  cdsln(xl),kvlin             reset proper line value
```

fi

mov	`kvrtn,wa`	load return type
beq	`wa,=scrtn,exixr`	exit with result in xr if return
beq	`wa,=scfrt,exfal`	fail if freturn

```
    retrn (continued)
    here for nreturn
        beq  (xr),=b$nml,rtn11          jump if is a name
        jsr  gtnvr                      else try convert to variable name
        err  243,function result        in nreturn is not name
        mov  xr,xl                      if ok, copy vrblk (name base) ptr
        mov  *vrval,wa                  set name offset
        brn  rtn12                      and merge
    here if returned result is a name
rtn11   mov  nmbas(xr),xl               load name base
        mov  nmofs(xr),wa               load name offset
    merge here with returned name in (xl,wa)
rtn12   mov  xl,xr                      preserve xl
        lcw  wb                         load next word
        mov  xr,xl                      restore xl
        beq  wb,=ofne$,exnam            exit if called by name
        mov  wb,-(xs)                   else save code word
        jsr  acess                      get value
        ppm  exfal                      fail if access fails
        mov  xr,xl                      if ok, copy result
        mov  (xs),xr                    reload next code word
        mov  xl,(xs)                    store result on stack
        mov  (xr),xl                    load routine address
        bri  xl                         jump to execute next code word
```

```
    stcov -- signal statement counter overflow
    brn  stcov            jump to signal statement count oflo
    permit up to 10 more statements to be obeyed so that
    setexit trap can regain control.
    stcov continues by issuing the error message
stcov   rtn   xl
        icv   errft                                  fatal error
        ldi   intvt                                  get 10
        adi   kvstl                                  add to former limit
        sti   kvstl                                  store as new stlimit
        ldi   intvt                                  get 10
        sti   kvstc                                  set as new count
        jsr   stgcc                                  recompute countdown counters
        erb   244,statement count                    exceeds value of stlimit keyword
```

```
    stmgo -- start execution of new statement
    (xr)                  pointer to cdblk for new statement
    brn  stmgo            jump to execute new statement
    stmgo continues by executing the next statement
stmgo   rtn   244,statement countunt
        mov   xr,r$cod                     set new code block pointer
        dcv   stmct                        see if time to check something
        bze   stmct,stgo2                  jump if so
        mov   kvstn,kvlst                  set lastno
        mov   cdstm(xr),kvstn              set stno
```

---

if **.csln**

```
        mov   kvlin,kvlln                  set lastline
        mov   cdsln(xr),kvlin              set line
```

fi

```
        add   *cdcod,xr                    point to first code word
        lcp   xr                           set code pointer
    here to execute first code word of statement
stgo1   lcw   xr                           load next code word
        zer   xl                           clear garbage xl
        bri   (xr)                         execute it
    check profiling, polling, stlimit, statement tracing
stgo2   bze   kvpfl,stgo3                  skip if no profiling
        jsr   prflu                        else profile the statement in kvstn
    here when finished with profiling
stgo3   mov   kvstn,kvlst                  set lastno
        mov   cdstm(xr),kvstn              set stno
```

---

if **.csln**

```
        mov   kvlin,kvlln                  set lastline
        mov   cdsln(xr),kvlin              set line
```

fi

```
        add   *cdcod,xr                    point to first code word
        lcp   xr                           set code pointer
```

---

if **.cpol**

```
    here to check for polling
        mov   stmcs,-(xs)                  save present count start on stack
        dcv   polct                        poll interval within stmct
        bnz   polct,stgo4                  jump if not poll time yet
        zer   wa                           =0 for poll
        mov   kvstn,wb                     statement number
        mov   xr,xl                        make collectable
        jsr   syspl                        allow interactive access
        err   syspl                        allow interactive access
        ppm                                single step
        ppm                                expression evaluation
        mov   xl,xr                        restore code block pointer
        mov   wa,polcs                     poll interval start value
        jsr   stgcc                        recompute counter values
```

fi

```
    check statement limit
stgo4   ldi   kvstc                        get stmt count
        ilt   stgo5                        omit counting if negative
        mti   (xs)+                        reload start value of counter
```

```
        ngi                              negate
        adi   kvstc                      stmt count minus counter
        sti   kvstc                      replace it
        ile   stcov                      fail if stlimit reached
        bze   r$stc,stgo5                jump if no statement trace
        zer   xr                         clear garbage value in xr
        mov   r$stc,xl                   load pointer to stcount trblk
        jsr   ktrex                      execute keyword trace
    reset stmgo counter
stgo5   mov   stmcs,stmct                reset counter
        brn   stgo1                      fetch next code word
```

```
    stopr -- terminate run
    (xr)                 points to ending message
    brn stopr            jump to terminate run
    terminate run and print statistics.  on entry xr points
    to ending message or is zero if message  printed already.
stopr   rtn   stgo1
```

---

if **.csax**

```
        bze   xr,stpra                          skip if sysax already called
        jsr   sysax                             call after execution proc
stpra   add   rsmem,dname                       use the reserve memory
```

else

```
        add   rsmem,dname                       use the reserve memory
```

fi

```
        bne   xr,=endms,stpr0                   skip if not normal end message
        bnz   exsts,stpr3                       skip if exec stats suppressed
        zer   erich                             clear errors to int.ch. flag
    look to see if an ending message is supplied
stpr0   jsr   prtpg                             eject printer
        bze   xr,stpr1                          skip if no message
        jsr   prtst                             print message
    merge here if no message to print
stpr1   jsr   prtis                             print blank line
```

---

if **.csfn**

```
        bnz   gbcfl,stpr5                       if in garbage collection, skip
        mov   =stpm7,xr                         point to message /in file xxx/
        jsr   prtst                             print it
        mov   =prtmf,profs                      set column offset
        mov   kvstn,wc                          get statement number
        jsr   filnm                             get file name
        mov   xl,xr                             prepare to print
        jsr   prtst                             print file name
        jsr   prtis                             print to interactive channel
```

fi

---

if **.csln**

---

if **.csfn**

else

```
        bnz   gbcfl,stpr5                       if in garbage collection, skip
```

fi

```
        mov   r$cod,xr                          get code pointer
        mti   cdsln(xr)                         get source line number
        mov   =stpm6,xr                         point to message /in line xxx/
        jsr   prtmx                             print it
```

fi

```
stpr5   mti   kvstn                             get statement number
        mov   =stpm1,xr                         point to message /in statement xxx/
        jsr   prtmx                             print it
        jsr   systm                             get current time
        sbi   timsx                             minus start time = elapsed exec tim
        sti   stpti                             save for later
        mov   =stpm3,xr                         point to msg /execution time msec /
        jsr   prtmx                             print it
```

ldi	`kvstl`	get statement limit
ilt	`stpr2`	skip if negative
sbi	`kvstc`	minus counter = course count
sti	`stpsi`	save
mov	`stmcs,wa`	refine with counter start value
sub	`stmct,wa`	minus current counter
mti	`wa`	convert to integer
adi	`stpsi`	add in course count
sti	`stpsi`	save
mov	`=stpm2,xr`	point to message /stmts executed/
jsr	`prtmx`	print it

---

if **.ctmd**
else

ldi	`stpti`	reload elapsed time
mli	`intth`	*1000 (microsecs)
iov	`stpr2`	jump if we cannot compute
dvi	`stpsi`	divide by statement count
iov	`stpr2`	jump if overflow
mov	`=stpm4,xr`	point to msg (mcsec per statement /
jsr	`prtmx`	print it

fi

```
    stopr (continued)
    merge to skip message (overflow or negative stlimit)
stpr2   mti   gbcnt                     load count of collections
        mov   =stpm5,xr                 point to message /regenerations /
        jsr   prtmx                     print it
        jsr   prtmm                     print memory usage
        jsr   prtis                     one more blank for luck
    check if dump requested
```

---

if **.cnpf**

```
stpr3   mov   kvdmp,xr                  load dump keyword
```

else

```
stpr3   jsr   prflr                     print profile if wanted
        mov   kvdmp,xr                  load dump keyword
```

fi

```
        jsr   dumpr                     execute dump if requested
        mov   r$fcb,xl                  get fcblk chain head
        mov   kvabe,wa                  load abend value
        mov   kvcod,wb                  load code value
        jsr   sysej                     exit to system
```

---

if **.cera**

```
    here after sysea call and suppressing error msg print
stpr4   rtn   sysej
        add   rsmem,dname               use the reserve memory
        bze   exsts,stpr1               if execution stats requested
        brn   stpr3                     check if dump or profile needed
```

fi

```
    succp -- signal successful match of a pattern node
    see pattern match routines for details
    (xr)                 current node
    (wb)                 current cursor
    (xl)                 may be non-collectable
    brn  succp           signal successful pattern match
    succp continues by matching the successor node
succp   rtn  stpr3
        mov  pthen(xr),xr                  load successor node
        mov  (xr),xl                       load node code entry address
        bri  xl                            jump to match successor node
```

```
    sysab -- print /abnormal end/ and terminate
sysab   rtn   xl
        mov   =endab,xr                          point to message
        mov   =num01,kvabe                       set abend flag
        jsr   prtnl                              skip to new line
        brn   stopr                              jump to pack up
```

```
    systu -- print /time up/ and terminate
systu   rtn   stopr
        mov   =endtu,xr                        point to message
        mov   strtu,wa                         get chars /tu/
        mov   wa,kvcod                         put in kvcod
        mov   timup,wa                         check state of timeup switch
        mnz   timup                            set switch
        bnz   wa,stopr                         stop run if already set
        erb   245,translation/execution        time expired
```

```
the following section contains procedures which are
used for various purposes throughout the system.
each procedure is preceded by a description of the
calling sequence. usually the arguments are in registers
but arguments can also occur on the stack and as
parameters assembled after the jsr instruction.
the following considerations apply to these descriptions.
1)   the stack pointer (xs) is not changed unless the
     change is explicitly documented in the call.
2)   registers whose entry values are not mentioned
     may contain any value except that xl,xr may only
     contain proper (collectable) pointer values.
     this condition on means that the called routine
     may if it chooses preserve xl,xr by stacking.
3)   registers not mentioned on exit contain the same
     values as they did on entry except that values in
     xr,xl may have been relocated by the collector.
4)   registers which are destroyed on exit may contain
     any value except that values in xl,xr are proper
     (collectable) pointers.
5)   the code pointer register points to the current
     code location on entry and is unchanged on exit.
in the above description, a collectable pointer is one
which either points outside the dynamic region or
points to the start of a block in the dynamic region.
in those cases where the calling sequence contains
parameters which are used as alternate return points,
these parameters may be replaced by error codes
assembled with the err instruction. this will result
in the posting of the error if the return is taken.
the procedures all have names consisting of five letters
and are in alphabetical order by their names.
```

```
    acess - access variable value with trace/input checks
    acess loads the value of a variable. trace and input
    associations are tested for and executed as required.
    acess also handles the special cases of pseudo-variables.
    (xl)                variable name base
    (wa)                variable name offset
    jsr  acess          call to access value
    ppm  loc            transfer loc if access failure
    (xr)                variable value
    (wa,wb,wc)          destroyed
    (xl,ra)             destroyed
    failure can occur if an input association causes an end
    of file condition or if the evaluation of an expression
    associated with an expression variable fails.
acess   prc  r,1                               entry point (recursive)
        mov  xl,xr                             copy name base
        add  wa,xr                             point to variable location
        mov  (xr),xr                           load variable value
    loop here to check for successive trblks
acs02   bne  (xr),=b$trt,acs18                 jump if not trapped
    here if trapped
        beq  xr,=trbkv,acs12                   jump if keyword variable
        bne  xr,=trbev,acs05                   jump if not expression variable
    here for expression variable, evaluate variable
        mov  evexp(xl),xr                      load expression pointer
        zer  wb                                evaluate by value
        jsr  evalx                             evaluate expression
        ppm  acs04                             jump if evaluation failure
        brn  acs02                             check value for more trblks
```

```
    acess (continued)
    here on reading end of file
acs03   add  *num03,xs                         pop trblk ptr, name base and offset
        mov  xr,dnamp                          pop unused scblk
    merge here when evaluation of expression fails
acs04   exi  1                                 take alternate (failure) return
    here if not keyword or expression variable
acs05   mov  trtyp(xr),wb                      load trap type code
        bnz  wb,acs10                          jump if not input association
        bze  kvinp,acs09                       ignore input assoc if input is off
    here for input association
        mov  xl,-(xs)                          stack name base
        mov  wa,-(xs)                          stack name offset
        mov  xr,-(xs)                          stack trblk pointer
        mov  kvtrm,actrm                       temp to hold trim keyword
        mov  trfpt(xr),xl                      get file ctrl blk ptr or zero
        bnz  xl,acs06                          jump if not standard input file
        beq  trter(xr),=v$ter,acs21            jump if terminal
    here to read from standard input file
        mov  cswin,wa                          length for read buffer
        jsr  alocs                             build string of appropriate length
        jsr  sysrd                             read next standard input image
        ppm  acs03                             jump to fail exit if end of file
        brn  acs07                             else merge with other file case
    here for input from other than standard input file
acs06   mov  xl,wa                             fcblk ptr
        jsr  sysil                             get input record max length (to wa)
        bnz  wc,acs6a                          jump if not binary file
        mov  wc,actrm                          disable trim for binary file
acs6a   jsr  alocs                             allocate string of correct size
        mov  xl,wa                             fcblk ptr
        jsr  sysin                             call system input routine
        ppm  acs03                             jump to fail exit if end of file
        ppm  acs22                             error
        ppm  acs23                             error
```

```
    acess (continued)
    merge here after obtaining input record
acs07   mov  actrm,wb                    load trim indicator
        jsr  trimr                       trim record as required
        mov  xr,wb                       copy result pointer
        mov  (xs),xr                     reload pointer to trblk
    loop to chase to end of trblk chain and store value
acs08   mov  xr,xl                       save pointer to this trblk
        mov  trnxt(xr),xr                load forward pointer
        beq  (xr),=b$trt,acs08           loop if this is another trblk
        mov  wb,trnxt(xl)                else store result at end of chain
        mov  (xs)+,xr                    restore initial trblk pointer
        mov  (xs)+,wa                    restore name offset
        mov  (xs)+,xl                    restore name base pointer
    come here to move to next trblk
acs09   mov  trnxt(xr),xr                load forward ptr to next value
        brn  acs02                       back to check if trapped
    here to check for access trace trblk
acs10   bne  wb,=trtac,acs09             loop back if not access trace
        bze  kvtra,acs09                 ignore access trace if trace off
        dcv  kvtra                       else decrement trace count
        bze  trfnc(xr),acs11             jump if print trace
```

```
    acess (continued)
    here for full function trace
        jsr   trxeq                               call routine to execute trace
        brn   acs09                               jump for next trblk
    here for case of print trace
acs11   jsr   prtsn                               print statement number
        jsr   prtnv                               print name = value
        brn   acs09                               jump back for next trblk
    here for keyword variable
acs12   mov   kvnum(xl),xr                        load keyword number
        bge   xr,=k$v$$,acs14                     jump if not one word value
        mti   kvabe(xr)                           else load value as integer
    common exit with keyword value as integer in (ia)
acs13   jsr   icbld                               build icblk
        brn   acs18                               jump to exit
    here if not one word keyword value
acs14   bge   xr,=k$s$$,acs15                     jump if special case
        sub   =k$v$$,xr                           else get offset
        wtb   xr                                  convert to byte offset
        add   =ndabo,xr                           point to pattern value
        brn   acs18                               jump to exit
    here if special keyword case
acs15   mov   kvrtn,xl                            load rtntype in case
        ldi   kvstl                               load stlimit in case
        sub   =k$s$$,xr                           get case number
        bsw   xr,k$$n$                            switch on keyword number
```

---

if **.csfn**

```
        iff   k$$fl,acs26                         file
        iff   k$$lf,acs27                         lastfile
```

fi

---

if **.culk**

```
        iff   k$$lc,acs24                         lcase
        iff   k$$uc,acs25                         ucase
```

fi

```
        iff   k$$al,acs16                         jump if alphabet
        iff   k$$rt,acs17                         rtntype
        iff   k$$sc,acs19                         stcount
        iff   k$$sl,acs13                         stlimit
        iff   k$$et,acs20                         errtext
        esw                                       end switch on keyword number
```

acess (continued)

---

if **.culk**

```
    lcase
acs24   mov  =lcase,xr                  load pointer to lcase string
        brn  acs18                      common return
    ucase
acs25   mov  =ucase,xr                  load pointer to ucase string
        brn  acs18                      common return
```

fi

---

if **.csfn**

```
    file
acs26   mov  kvstn,wc                   load current stmt number
        brn  acs28                      merge to obtain file name
    lastfile
acs27   mov  kvlst,wc                   load last stmt number
    merge here to map statement number in wc to file name
acs28   jsr  filnm                      obtain file name for this stmt
        brn  acs17                      merge to return string in xl
```

fi

```
    alphabet
acs16   mov  kvalp,xl                   load pointer to alphabet string
    rtntype merges here
acs17   mov  xl,xr                      copy string ptr to proper reg
    common return point
acs18   exi                             return to acess caller
    here for stcount (ia has stlimit)
acs19   ilt  acs29                      if counting suppressed
        mov  stmcs,wa                   refine with counter start value
        sub  stmct,wa                   minus current counter
        mti  wa                         convert to integer
        adi  kvstl                      add stlimit
acs29   sbi  kvstc                      stcount = limit - left
        brn  acs13                      merge back with integer result
    errtext
acs20   mov  r$etx,xr                   get errtext string
        brn  acs18                      merge with result
    here to read a record from terminal
acs21   mov  =rilen,wa                  buffer length
        jsr  alocs                      allocate buffer
        jsr  sysri                      read record
        ppm  acs03                      endfile
        brn  acs07                      merge with record read
    error returns
acs22   mov  xr,dnamp                   pop unused scblk
        erb  202,input from file        caused non-recoverable error
acs23   mov  xr,dnamp                   pop unused scblk
        erb  203,input file record      has incorrect format
        enp                             end procedure acess
```

```
    acomp -- compare two arithmetic values
    1(xs)                first argument
    0(xs)                second argument
    jsr  acomp           call to compare values
    ppm  loc             transfer loc if arg1 is non-numeric
    ppm  loc             transfer loc if arg2 is non-numeric
    ppm  loc             transfer loc for arg1 lt arg2
    ppm  loc             transfer loc for arg1 eq arg2
    ppm  loc             transfer loc for arg1 gt arg2
    (normal return is never given)
    (wa,wb,wc,ia,ra)     destroyed
    (xl,xr)              destroyed
acomp  prc  n,5                          entry point
       jsr  arith                        load arithmetic operands
       ppm  acmp7                        jump if first arg non-numeric
       ppm  acmp8                        jump if second arg non-numeric
```

---

if **.cnra**

else

```
       ppm  acmp4                        jump if real arguments
```

fi

```
    here for integer arguments
       sbi  icval(xl)                    subtract to compare
       iov  acmp3                        jump if overflow
       ilt  acmp5                        else jump if arg1 lt arg2
       ieq  acmp2                        jump if arg1 eq arg2
    here if arg1 gt arg2
acmp1  exi  5                            take gt exit
    here if arg1 eq arg2
acmp2  exi  4                            take eq exit
```

```
    acomp (continued)
    here for integer overflow on subtract
acmp3   ldi   icval(xl)                                  load second argument
        ilt   acmp1                                      gt if negative
        brn   acmp5                                      else lt
```

---

if **.cnra**
else

```
    here for real operands
acmp4   sbr   rcval(xl)                                  subtract to compare
        rov   acmp6                                      jump if overflow
        rgt   acmp1                                      else jump if arg1 gt
        req   acmp2                                      jump if arg1 eq arg2
```

fi

```
    here if arg1 lt arg2
acmp5   exi   3                                          take lt exit
```

---

if **.cnra**
else

```
    here if overflow on real subtraction
acmp6   ldr   rcval(xl)                                  reload arg2
        rlt   acmp1                                      gt if negative
        brn   acmp5                                      else lt
```

fi

```
    here if arg1 non-numeric
acmp7   exi   1                                          take error exit
    here if arg2 non-numeric
acmp8   exi   2                                          take error exit
        enp                                              end procedure acomp
```

```
    alloc                 allocate block of dynamic storage
    (wa)                  length required in bytes
    jsr  alloc            call to allocate block
    (xr)                  pointer to allocated block
    a possible alternative to aov ... and following stmt is -
    mov  dname,xr .  sub  wa,xr .  blo xr,dnamp,aloc2 .
    mov  dnamp,xr .  add  wa,xr
alloc   prc   e,0                                   entry point
    common exit point
aloc1   mov   dnamp,xr                              point to next available loc
        aov   wa,xr,aloc2                           point past allocated block
        bgt   xr,dname,aloc2                        jump if not enough room
        mov   xr,dnamp                              store new pointer
        sub   wa,xr                                 point back to start of allocated bk
        exi                                         return to caller
    here if insufficient room, try a garbage collection
aloc2   mov   wb,allsv                              save wb
alc2a   zer   wb                                    set no upward move for gbcol
        jsr   gbcol                                 garbage collect
```

---

if **.csed**

```
        mov   xr,wb                                 remember new sediment size
```

fi

```
    see if room after gbcol or sysmm call
aloc3   mov   dnamp,xr                              point to first available loc
        aov   wa,xr,alc3a                           point past new block
        blo   xr,dname,aloc4                        jump if there is room now
    failed again, see if we can get more core
alc3a   jsr   sysmm                                 try to get more memory
        wtb   xr                                    convert to baus (sgd05)
        add   xr,dname                              bump ptr by amount obtained
        bnz   xr,aloc3                              jump if got more core
```

---

if **.csed**

```
        bze   dnams,alc3b                           jump if there was no sediment
        zer   dnams                                 try collecting the sediment
        brn   dnams                                 try collecting the sediment
    sysmm failed and there was no sediment to collect
alc3b   add   rsmem,dname                           get the reserve memory
```

else

```
        add   rsmem,dname                           get the reserve memory
```

fi

```
        zer   rsmem                                 only permissible once
        icv   errft                                 fatal error
        erb   errft                                 fatal error
```

```
    here after successful garbage collection
aloc4   sti   allia                          save ia
```

if **.csed**

```
        mov   wb,dnams                       record new sediment size
```

fi

```
        mov   dname,wb                       get dynamic end adrs
        sub   dnamp,wb                       compute free store
        btw   wb                             convert bytes to words
        mti   wb                             put free store in ia
        mli   alfsf                          multiply by free store factor
        iov   aloc5                          jump if overflowed
        mov   dname,wb                       dynamic end adrs
        sub   dnamb,wb                       compute total amount of dynamic
        btw   wb                             convert to words
        mov   wb,aldyn                       store it
        sbi   aldyn                          subtract from scaled up free store
        igt   aloc5                          jump if sufficient free store
        jsr   sysmm                          try to get more store
        wtb   xr                             convert to baus (sgd05)
        add   xr,dname                       adjust dynamic end adrs
    merge to restore ia and wb
aloc5   ldi   allia                          recover ia
        mov   allsv,wb                       restore wb
        brn   aloc1                          jump back to exit
        enp                                  end procedure alloc
```

if **.cnbf**

else

```
    alobf -- allocate buffer
    this routines allocates a new buffer.  as the bfblk
    and bcblk come in pairs, both are allocated here,
    and xr points to the bcblk on return.  the bfblk
    and bcblk are set to the null buffer, and the idval
    is zero on return.
    (wa)                  buffer size in characters
    jsr  alobf            call to create buffer
    (xr)                  bcblk ptr
    (wa,wb)               destroyed
alobf   prc   e,0                                  entry point
        bgt   wa,kvmxl,alb01                       check for maxlngth exceeded
        mov   wa,wb                                hang onto allocation size
        ctb   wa,bfsi$                             get total block size
        add   *bcsi$,wa                            add in allocation for bcblk
        jsr   alloc                                allocate frame
        mov   =b$bct,(xr)                          set type
        zer   idval(xr)                            no id yet
        zer   bclen(xr)                            no defined length
        mov   xl,wa                                save xl
        mov   xr,xl                                copy bcblk ptr
        add   *bcsi$,xl                            bias past partially built bcblk
        mov   =b$bft,(xl)                          set bfblk type word
        mov   wb,bfalc(xl)                         set allocated size
        mov   xl,bcbuf(xr)                         set pointer in bcblk
        zer   bfchr(xl)                            clear first word (null pad)
        mov   wa,xl                                restore entry xl
        exi                                        return to caller
    here for mxlen exceeded
alb01   erb   273,buffer size exceeds              value of maxlngth keyword
        enp                                        end procedure alobf
```

fi

```
    alocs -- allocate string block
    alocs is used to build a frame for a string block into
    which the actual characters are placed by the caller.
    all strings are created with a call to alocs (the
    exceptions occur in trimr and s$rpl procedures).
    (wa)                 length of string to be allocated
    jsr  alocs           call to allocate scblk
    (xr)                 pointer to resulting scblk
    (wa)                 destroyed
    (wc)                 character count (entry value of wa)
    the resulting scblk has the type word and the length
    filled in and the last word is cleared to zero characters
    to ensure correct right padding of the final word.
alocs  prc  e,0                        entry point
       bgt  wa,kvmxl,alcs2             jump if length exceeds maxlength
       mov  wa,wc                      else copy length
       ctb  wa,scsi$                   compute length of scblk in bytes
       mov  dnamp,xr                   point to next available location
       aov  wa,xr,alcs0                point past block
       blo  xr,dname,alcs1             jump if there is room
    insufficient memory
alcs0  zer  xr                         else clear garbage xr value
       jsr  alloc                      and use standard allocator
       add  wa,xr                      point past end of block to merge
    merge here with xr pointing beyond new block
alcs1  mov  xr,dnamp                   set updated storage pointer
       zer  -(xr)                      store zero chars in last word
       dca  wa                         decrement length
       sub  wa,xr                      point back to start of block
       mov  =b$scl,(xr)                set type word
       mov  wc,sclen(xr)               store length in chars
       exi                             return to alocs caller
    come here if string is too long
alcs2  erb  205,string length          exceeds value of maxlngth keyword
       enp                             end procedure alocs
```

```
    alost -- allocate space in static region
    (wa)                  length required in bytes
    jsr  alost            call to allocate space
    (xr)                  pointer to allocated block
    (wb)                  destroyed
    note that the coding ensures that the resulting value
    of state is always less than dnamb. this fact is used
    in testing a variable name for being in the static region
alost   prc  e,0                                entry point
    merge back here after allocating new chunk
alst1   mov  state,xr                           point to current end of area
        aov  wa,xr,alst2                        point beyond proposed block
        bge  xr,dnamb,alst2                     jump if overlap with dynamic area
        mov  xr,state                           else store new pointer
        sub  wa,xr                              point back to start of block
        exi                                     return to alost caller
    here if no room, prepare to move dynamic storage up
alst2   mov  wa,alsta                           save wa
        bge  wa,*e$sts,alst3                    skip if requested chunk is large
        mov  *e$sts,wa                          else set to get large enough chunk
    here with amount to move up in wa
alst3   jsr  alloc                              allocate block to ensure room
        mov  xr,dnamp                           and delete it
        mov  wa,wb                              copy move up amount
        jsr  gbcol                              call gbcol to move dynamic area up
```

---

if **.csed**

```
        mov  xr,dnams                           remember new sediment size
```

fi

```
        mov  alsta,wa                           restore wa
        brn  alst1                              loop back to try again
        enp                                     end procedure alost
```

if **.cnbf**
else

```
    apndb -- append string to buffer
    this routine is used by buffer handling routines to
    append data to an existing bfblk.
    (xr)                 existing bcblk to be appended
    (xl)                 convertable to string
    jsr  apndb           call to append to buffer
    ppm  loc             thread if (xl) cant be converted
    ppm  loc             if not enough room
    (wa,wb)              destroyed
    if more characters are specified than can be inserted,
    then no action is taken and the second return is taken.
apndb   prc  e,2                                   entry point
        mov  bclen(xr),wa                          load offset to insert
        zer  wb                                    replace section is null
        jsr  insbf                                 call to insert at end
        ppm  apn01                                 convert error
        ppm  apn02                                 no room
        exi                                        return to caller
    here to take convert failure exit
apn01   exi  1                                     return to caller alternate
    here for no fit exit
apn02   exi  2                                     alternate exit to caller
        enp                                        end procedure apndb
```

fi

```
arith -- fetch arithmetic operands
arith is used by functions and operators which expect
two numeric arguments (operands) which must both be
integer or both be real. arith fetches two arguments from
the stack and performs any necessary conversions.
1(xs)                 first argument (left operand)
0(xs)                 second argument (right operand)
jsr  arith            call to fetch numeric arguments
ppm  loc              transfer loc for opnd 1 non-numeric
ppm  loc              transfer loc for opnd 2 non-numeric
```

---

if **.cnra**

else

```
ppm  loc              transfer loc for real operands
```

fi

```
for integer args, control returns past the parameters
(ia)                  left operand value
(xr)                  ptr to icblk for left operand
(xl)                  ptr to icblk for right operand
(xs)                  popped twice
(wa,wb,ra)            destroyed
```

---

if **.cnra**

else

```
for real arguments, control returns to the location
specified by the third parameter.
(ra)                  left operand value
(xr)                  ptr to rcblk for left operand
(xl)                  ptr to rcblk for right operand
(wa,wb,wc)            destroyed
(xs)                  popped twice
```

fi

```
    arith (continued)
    entry point
```

```
if .cnra
arith   prc   n,2                                   entry point
else
arith   prc   n,3                                   entry point
fi
        mov   (xs)+,xl                              load right operand
        mov   (xs)+,xr                              load left operand
        mov   (xl),wa                               get right operand type word
        beq   wa,=b$icl,arth1                       jump if integer
```

```
if .cnra
else
        beq   wa,=b$rcl,arth4                       jump if real
fi
        mov   xr,-(xs)                              else replace left arg on stack
        mov   xl,xr                                 copy left arg pointer
        jsr   gtnum                                 convert to numeric
        ppm   arth6                                 jump if unconvertible
        mov   xr,xl                                 else copy converted result
        mov   (xl),wa                               get right operand type word
        mov   (xs)+,xr                              reload left argument
```

```
if .cnra
else
        beq   wa,=b$rcl,arth4                       jump if right arg is real
fi
    here if right arg is an integer
arth1   bne   (xr),=b$icl,arth3                     jump if left arg not integer
    exit for integer case
arth2   ldi   icval(xr)                             load left operand value
        exi                                         return to arith caller
    here for right operand integer, left operand not
arth3   jsr   gtnum                                 convert left arg to numeric
        ppm   arth7                                 jump if not convertible
        beq   wa,=b$icl,arth2                       jump back if integer-integer
```

```
if .cnra
else
    here we must convert real-integer to real-real
        mov   xr,-(xs)                              put left arg back on stack
        ldi   icval(xl)                             load right argument value
        itr                                         convert to real
        jsr   rcbld                                 get real block for right arg, merge
        mov   xr,xl                                 copy right arg ptr
        mov   (xs)+,xr                              load left argument
        brn   arth5                                 merge for real-real case
```

```
    arith (continued)
    here if right argument is real
arth4   beq   (xr),=b$rcl,arth5            jump if left arg real
        jsr   gtrea                        else convert to real
        ppm   arth7                        error if unconvertible
    here for real-real
arth5   ldr   rcval(xr)                    load left operand value
        exi   3                            take real-real exit
fi
    here for error converting right argument
arth6   ica   xs                           pop unwanted left arg
        exi   2                            take appropriate error exit
    here for error converting left operand
arth7   exi   1                            take appropriate error return
        enp                                end procedure arith
```

```
    asign -- perform assignment
    asign performs the assignment of a value to a variable
    with appropriate checks for output associations and
    value trace associations which are executed as required.
    asign also handles the special cases of assignment to
    pattern and expression variables.
    (wb)                  value to be assigned
    (xl)                  base pointer for variable
    (wa)                  offset for variable
    jsr  asign            call to assign value to variable
    ppm  loc              transfer loc for failure
    (xr,xl,wa,wb,wc)      destroyed
    (ra)                  destroyed
    failure occurs if the evaluation of an expression
    associated with an expression variable fails.
asign   prc  r,1                                   entry point (recursive)
    merge back here to assign result to expression variable.
asg01   add  wa,xl                                 point to variable value
        mov  (xl),xr                               load variable value
        beq  (xr),=b$trt,asg02                     jump if trapped
        mov  wb,(xl)                               else perform assignment
        zer  xl                                    clear garbage value in xl
        exi                                        and return to asign caller
    here if value is trapped
asg02   sub  wa,xl                                 restore name base
        beq  xr,=trbkv,asg14                       jump if keyword variable
        bne  xr,=trbev,asg04                       jump if not expression variable
    here for assignment to expression variable
        mov  evexp(xl),xr                          point to expression
        mov  wb,-(xs)                              store value to assign on stack
        mov  =num01,wb                             set for evaluation by name
        jsr  evalx                                 evaluate expression by name
        ppm  asg03                                 jump if evaluation fails
        mov  (xs)+,wb                              else reload value to assign
        brn  asg01                                 loop back to perform assignment
```

```
    asign (continued)
    here for failure during expression evaluation
asg03  ica  xs                              remove stacked value entry
       exi  1                               take failure exit
    here if not keyword or expression variable
asg04  mov  xr,-(xs)                        save ptr to first trblk
    loop to chase down trblk chain and assign value at end
asg05  mov  xr,wc                           save ptr to this trblk
       mov  trnxt(xr),xr                    point to next trblk
       beq  (xr),=b$trt,asg05               loop back if another trblk
       mov  wc,xr                           else point back to last trblk
       mov  wb,trval(xr)                    store value at end of chain
       mov  (xs)+,xr                        restore ptr to first trblk
    loop to process trblk entries on chain
asg06  mov  trtyp(xr),wb                    load type code of trblk
       beq  wb,=trtvl,asg08                 jump if value trace
       beq  wb,=trtou,asg10                 jump if output association
    here to move to next trblk on chain
asg07  mov  trnxt(xr),xr                    point to next trblk on chain
       beq  (xr),=b$trt,asg06               loop back if another trblk
       exi                                  else end of chain, return to caller
    here to process value trace
asg08  bze  kvtra,asg07                     ignore value trace if trace off
       dcv  kvtra                           else decrement trace count
       bze  trfnc(xr),asg09                 jump if print trace
       jsr  trxeq                           else execute function trace
       brn  asg07                           and loop back
```

```
    asign (continued)
    here for print trace
asg09   jsr   prtsn                         print statement number
        jsr   prtnv                         print name = value
        brn   asg07                         loop back for next trblk
    here for output association
asg10   bze   kvoup,asg07                   ignore output assoc if output off
asg1b   mov   xr,xl                         copy trblk pointer
        mov   trnxt(xr),xr                  point to next trblk
        beq   (xr),=b$trt,asg1b             loop back if another trblk
        mov   xl,xr                         else point back to last trblk
```

---

if **.cnbf**

```
        mov   trval(xr),-(xs)               stack value to output
```

else

```
        mov   trval(xr),xr                  get value to output
        beq   (xr),=b$bct,asg11             branch if buffer
        mov   xr,-(xs)                      stack value to output
```

fi

```
        jsr   gtstg                         convert to string
        ppm   asg12                         get datatype name if unconvertible
    merge with string or buffer to output in xr
asg11   mov   trfpt(xl),wa                  fcblk ptr
        bze   wa,asg13                      jump if standard output file
    here for output to file
asg1a   jsr   sysou                         call system output routine
        err   206,output caused             file overflow
        err   207,output caused             non-recoverable error
        exi                                 else all done, return to caller
    if not printable, get datatype name instead
asg12   jsr   dtype                         call datatype routine
        brn   asg11                         merge
    here to print a string to standard output or terminal
```

---

if **.csou**

```
asg13   beq   trter(xl),=v$ter,asg1a        jump if terminal output
        icv   wa                            signal standard output
        brn   asg1a                         use sysou to perform output
```

else

---

if **.cnbf**

```
asg13   jsr   prtst                         print string value
```

else

```
asg13   bne   (xr),=b$bct,asg1c             branch if not buffer
        mov   xr,-(xs)                      stack buffer
        jsr   gtstg                         convert to string
        ppm                                 always succeeds
asg1c   jsr   prtst                         print string value
```

fi

```
        beq   trter(xl),=v$ter,asg20        jump if terminal output
        jsr   prtnl                         end of line
        exi                                 return to caller
```

fi

```
    asign (continued)
    here for keyword assignment
asg14  mov  kvnum(xl),xl                    load keyword number
       beq  xl,=k$etx,asg19                 jump if errtext
       mov  wb,xr                           copy value to be assigned
       jsr  gtint                           convert to integer
       err  208,keyword value               assigned is not integer
       ldi  icval(xr)                       else load value
       beq  xl,=k$stl,asg16                 jump if special case of stlimit
       mfi  wa,asg18                        else get addr integer, test ovflow
       bgt  wa,mxlen,asg18                  fail if too large
       beq  xl,=k$ert,asg17                 jump if special case of errtype
```

if **.cnpf**
else

```
       beq  xl,=k$pfl,asg21                 jump if special case of profile
```

fi

```
       beq  xl,=k$mxl,asg24                 jump if special case of maxlngth
       beq  xl,=k$fls,asg26                 jump if special case of fullscan
       blt  xl,=k$p$$,asg15                 jump unless protected
       erb  209,keyword in assignment       is protected
    here to do assignment if not protected
asg15  mov  wa,kvabe(xl)                    store new value
       exi                                  return to asign caller
    here for special case of stlimit
    since stcount is maintained as (stlimit-stcount)
    it is also necessary to modify stcount appropriately.
asg16  sbi  kvstl                           subtract old limit
       adi  kvstc                           add old counter
       sti  kvstc                           store course counter value
       ldi  kvstl                           check if counting suppressed
       ilt  asg25                           do not refine if so
       mov  stmcs,wa                        refine with counter breakout
       sub  stmct,wa                        values
       mti  wa                              convert to integer
       ngi                                  current-start value
       adi  kvstc                           add in course counter value
       sti  kvstc                           save refined value
asg25  ldi  icval(xr)                       reload new limit value
       sti  kvstl                           store new limit value
       jsr  stgcc                           recompute countdown counters
       exi                                  return to asign caller
    here for special case of errtype
asg17  ble  wa,=nini9,error                 ok to signal if in range
    here if value assigned is out of range
asg18  erb  210,keyword value               assigned is negative or too large
    here for special case of errtext
asg19  mov  wb,-(xs)                        stack value
       jsr  gtstg                           convert to string
       err  211,value assigned              to keyword errtext not a string
       mov  xr,r$etx                        make assignment
       exi                                  return to caller
```

if **.csou**
else

```
    print string to terminal
asg20   jsr   prttr                          print
        exi                                  return
fi
```

```
if .cnpf
else
    here for keyword profile
asg21   bgt   wa,=num02,asg18                moan if not 0,1, or 2
        bze   wa,asg15                       just assign if zero
        bze   pfdmp,asg22                    branch if first assignment
        beq   wa,pfdmp,asg23                 also if same value as before
        erb   268,inconsistent               value assigned to keyword profile
asg22   mov   wa,pfdmp                       note value on first assignment
asg23   mov   wa,kvpfl                       store new value
        jsr   stgcc                          recompute countdown counts
        jsr   systm                          get the time
        sti   pfstm                          fudge some kind of start time
        exi                                  return to asign caller
fi
    here for keyword maxlngth
asg24   bge   wa,=mnlen,asg15                if acceptable value
        erb   287,value assigned             to keyword maxlngth is too small
    here for keyword fullscan
asg26   bnz   wa,asg15                       if acceptable value
        erb   274,value assigned             to keyword fullscan is zero
        enp                                  end procedure asign
```

```
    asinp -- assign during pattern match
    asinp is like asign and has a similar calling sequence
    and effect. the difference is that the global pattern
    variables are saved and restored if required.
    (xl)                 base pointer for variable
    (wa)                 offset for variable
    (wb)                 value to be assigned
    jsr  asinp           call to assign value to variable
    ppm  loc             transfer loc if failure
    (xr,xl)              destroyed
    (wa,wb,wc,ra)        destroyed
asinp   prc  r,1                          entry point, recursive
        add  wa,xl                        point to variable
        mov  (xl),xr                      load current contents
        beq  (xr),=b$trt,asnp1            jump if trapped
        mov  wb,(xl)                      else perform assignment
        zer  xl                           clear garbage value in xl
        exi                               return to asinp caller
    here if variable is trapped
asnp1   sub  wa,xl                        restore base pointer
        mov  pmssl,-(xs)                  stack subject string length
        mov  pmhbs,-(xs)                  stack history stack base ptr
        mov  r$pms,-(xs)                  stack subject string pointer
        mov  pmdfl,-(xs)                  stack dot flag
        jsr  asign                        call full-blown assignment routine
        ppm  asnp2                        jump if failure
        mov  (xs)+,pmdfl                  restore dot flag
        mov  (xs)+,r$pms                  restore subject string pointer
        mov  (xs)+,pmhbs                  restore history stack base pointer
        mov  (xs)+,pmssl                  restore subject string length
        exi                               return to asinp caller
    here if failure in asign call
asnp2   mov  (xs)+,pmdfl                  restore dot flag
        mov  (xs)+,r$pms                  restore subject string pointer
        mov  (xs)+,pmhbs                  restore history stack base pointer
        mov  (xs)+,pmssl                  restore subject string length
        exi  1                            take failure exit
        enp                               end procedure asinp
```

```
    blkln -- determine length of block
    blkln determines the length of a block in dynamic store.
    (wa)                  first word of block
    (xr)                  pointer to block
    jsr  blkln            call to get block length
    (wa)                  length of block in bytes
    (xl)                  destroyed
    blkln is used by the garbage collector and is not
    permitted to call gbcol directly or indirectly.
    the first word stored in the block (i.e. at xr) may
    be anything, but the contents of wa must be correct.
blkln   prc   e,0                          entry point
        mov   wa,xl                        copy first word
        lei   xl                           get entry id (bl$xx)
        bsw   xl,bl$$$,bln00               switch on block type
        iff   bl$ar,bln01                  arblk
```

---

if **.cnbf**
else

```
        iff   bl$bc,bln04                  bcblk
        iff   bl$bf,bln11                  bfblk
```

fi

---

if **.csln**

```
        iff   bl$cd,bln12                  cdblk
```

else

```
        iff   bl$cd,bln01                  cdblk
```

fi

```
        iff   bl$df,bln01                  dfblk
        iff   bl$ef,bln01                  efblk
```

---

if **.csln**

```
        iff   bl$ex,bln12                  exblk
```

else

```
        iff   bl$ex,bln01                  exblk
```

fi

```
        iff   bl$pf,bln01                  pfblk
        iff   bl$tb,bln01                  tbblk
        iff   bl$vc,bln01                  vcblk
        iff   bl$ev,bln03                  evblk
        iff   bl$kv,bln03                  kvblk
        iff   bl$p0,bln02                  p0blk
        iff   bl$se,bln02                  seblk
        iff   bl$nm,bln03                  nmblk
        iff   bl$p1,bln03                  p1blk
        iff   bl$p2,bln04                  p2blk
        iff   bl$te,bln04                  teblk
        iff   bl$ff,bln05                  ffblk
        iff   bl$tr,bln05                  trblk
        iff   bl$ct,bln06                  ctblk
        iff   bl$ic,bln07                  icblk
        iff   bl$pd,bln08                  pdblk
```

---

if **.cnra**
else

```
        iff   bl$rc,bln09                  rcblk
```

fi

```
iff   bl$sc,bln10          scblk
esw                        end of jump table on block type
```

```
    blkln (continued)
    here for blocks with length in second word
bln00   mov  num01(xr),wa                          load length
        exi                                        return to blkln caller
    here for length in third word (ar,cd,df,ef,ex,pf,tb,vc)
bln01   mov  num02(xr),wa                          load length from third word
        exi                                        return to blkln caller
    here for two word blocks (p0,se)
bln02   mov  *num02,wa                             load length (two words)
        exi                                        return to blkln caller
    here for three word blocks (nm,p1,ev,kv)
bln03   mov  *num03,wa                             load length (three words)
        exi                                        return to blkln caller
    here for four word blocks (p2,te,bc)
bln04   mov  *num04,wa                             load length (four words)
        exi                                        return to blkln caller
    here for five word blocks (ff,tr)
bln05   mov  *num05,wa                             load length
        exi                                        return to blkln caller
```

```
    blkln (continued)
    here for ctblk
bln06   mov  *ctsi$,wa                  set size of ctblk
        exi                             return to blkln caller
    here for icblk
bln07   mov  *icsi$,wa                  set size of icblk
        exi                             return to blkln caller
    here for pdblk
bln08   mov  pddfp(xr),xl               point to dfblk
        mov  dfpdl(xl),wa               load pdblk length from dfblk
        exi                             return to blkln caller
```

---

if **.cnra**

else

```
    here for rcblk
bln09   mov  *rcsi$,wa                  set size of rcblk
        exi                             return to blkln caller
```

fi

```
    here for scblk
bln10   mov  sclen(xr),wa               load length in characters
        ctb  wa,scsi$                   calculate length in bytes
        exi                             return to blkln caller
```

---

if **.cnbf**

else

```
    here for bfblk
bln11   mov  bfalc(xr),wa               get allocation in bytes
        ctb  wa,bfsi$                   calculate length in bytes
        exi                             return to blkln caller
```

fi

---

if **.csln**

```
    here for length in fourth word (cd,ex)
bln12   mov  num03(xr),wa               load length from cdlen/exlen
        exi                             return to blkln caller
```

fi

```
        enp                             end procedure blkln
```

```
    copyb -- copy a block
    (xs)                 block to be copied
    jsr  copyb           call to copy block
    ppm  loc             return if block has no idval field
                         normal return if idval field
    (xr)                 copy of block
    (xs)                 popped
    (xl,wa,wb,wc)        destroyed
copyb   prc  n,1                               entry point
        mov  (xs),xr                           load argument
        beq  xr,=nulls,cop10                   return argument if it is null
        mov  (xr),wa                           else load type word
        mov  wa,wb                             copy type word
        jsr  blkln                             get length of argument block
        mov  xr,xl                             copy pointer
        jsr  alloc                             allocate block of same size
        mov  xr,(xs)                           store pointer to copy
        mvw                                    copy contents of old block to new
        zer  xl                                clear garbage xl
        mov  (xs),xr                           reload pointer to start of copy
        beq  wb,=b$tbt,cop05                   jump if table
        beq  wb,=b$vct,cop01                   jump if vector
        beq  wb,=b$pdt,cop01                   jump if program defined
```

---

if **.cnbf**
else

```
        beq  wb,=b$bct,cop11                   jump if buffer
```

fi

```
        bne  wb,=b$art,cop10                   return copy if not array
    here for array (arblk)
        add  arofs(xr),xr                      point to prototype field
        brn  cop02                             jump to merge
    here for vector, program defined
cop01   add  *pdfld,xr                         point to pdfld = vcvls
    merge here for arblk, vcblk, pdblk to delete trap
    blocks from all value fields (the copy is untrapped)
cop02   mov  (xr),xl                           load next pointer
    loop to get value at end of trblk chain
cop03   bne  (xl),=b$trt,cop04                 jump if not trapped
        mov  trval(xl),xl                      else point to next value
        brn  cop03                             and loop back
```

```
    copyb (continued)
    here with untrapped value in xl
cop04   mov  xl,(xr)+                        store real value, bump pointer
        bne  xr,dnamp,cop02                  loop back if more to go
        brn  cop09                           else jump to exit
    here to copy a table
cop05   zer  idval(xr)                       zero id to stop dump blowing up
        mov  *tesi$,wa                       set size of teblk
        mov  *tbbuk,wc                       set initial offset
    loop through buckets in table
cop06   mov  (xs),xr                         load table pointer
        beq  wc,tblen(xr),cop09              jump to exit if all done
        mov  wc,wb                           else copy offset
        sub  *tenxt,wb                       subtract link offset to merge
        add  wb,xr                           next bucket header less link offset
        ica  wc                              bump offset
    loop through teblks on one chain
cop07   mov  tenxt(xr),xl                    load pointer to next teblk
        mov  (xs),tenxt(xr)                  set end of chain pointer in case
        beq  (xl),=b$tbt,cop06               back for next bucket if chain end
        sub  wb,xr                           point to head of previous block
        mov  xr,-(xs)                        stack ptr to previous block
        mov  *tesi$,wa                       set size of teblk
        jsr  alloc                           allocate new teblk
        mov  xr,-(xs)                        stack ptr to new teblk
        mvw                                  copy old teblk to new teblk
        mov  (xs)+,xr                        restore pointer to new teblk
        mov  (xs)+,xl                        restore pointer to previous block
        add  wb,xl                           add offset back in
        mov  xr,tenxt(xl)                    link new block to previous
        mov  xr,xl                           copy pointer to new block
    loop to set real value after removing trap chain
cop08   mov  teval(xl),xl                    load value
        beq  (xl),=b$trt,cop08               loop back if trapped
        mov  xl,teval(xr)                    store untrapped value in teblk
        zer  wb                              zero offset within teblk
        brn  cop07                           back for next teblk
    common exit point
cop09   mov  (xs)+,xr                        load pointer to block
        exi                                  return
    alternative return
cop10   exi  1                               return
```

```
if .cnbf
else
    here to copy buffer
cop11   mov  bcbuf(xr),xl                                   get bfblk ptr
        mov  bfalc(xl),wa                                   get allocation
        ctb  wa,bfsi$                                       set total size
        mov  xr,xl                                          save bcblk ptr
        jsr  alloc                                          allocate bfblk
        mov  bcbuf(xl),wb                                   get old bfblk
        mov  xr,bcbuf(xl)                                   set pointer to new bfblk
        mov  wb,xl                                          point to old bfblk
        mvw                                                 copy bfblk too
        zer  xl                                             clear rubbish ptr
        brn  cop09                                          branch to exit
fi
        enp                                                 end procedure copyb
    cdgcg -- generate code for complex goto
    used by cmpil to process complex goto tree
    (wb)                  must be collectable
    (xr)                  expression pointer
    jsr  cdgcg            call to generate complex goto
    (xl,xr,wa)            destroyed
cdgcg   prc  e,0                                            entry point
        mov  cmopn(xr),xl                                   get unary goto operator
        mov  cmrop(xr),xr                                   point to goto operand
        beq  xl,=opdvd,cdgc2                                jump if direct goto
        jsr  cdgnm                                          generate opnd by name if not direct
    return point
cdgc1   mov  xl,wa                                          goto operator
        jsr  cdwrd                                          generate it
        exi                                                 return to caller
    direct goto
cdgc2   jsr  cdgvl                                          generate operand by value
        brn  cdgc1                                          merge to return
        enp                                                 end procedure cdgcg
```

```
    cdgex -- build expression block
    cdgex is passed a pointer to an expression tree (see
    expan) and returns an expression (seblk or exblk).
```

---

if **.cevb**

```
    (wa)                  0 if by value, 1 if by name
```

fi

```
    (wc)                  some collectable value
    (wb)                  integer in range 0 le x le mxlen
    (xl)                  ptr to expression tree
    jsr  cdgex            call to build expression
    (xr)                  ptr to seblk or exblk
    (xl,wa,wb)            destroyed
cdgex   prc  r,0                            entry point, recursive
        blo  (xl),=b$vr$,cdgx1              jump if not variable
    here for natural variable, build seblk
        mov  *sesi$,wa                      set size of seblk
        jsr  alloc                          allocate space for seblk
        mov  =b$sel,(xr)                    set type word
        mov  xl,sevar(xr)                   store vrblk pointer
        exi                                 return to cdgex caller
    here if not variable, build exblk
cdgx1   mov  xl,xr                          copy tree pointer
        mov  wc,-(xs)                       save wc
        mov  cwcof,xl                       save current offset
```

---

if **.cevb**

```
        bze  wa,cdgx2                       jump if by value
```

fi

```
        mov  (xr),wa                        get type word
        bne  wa,=b$cmt,cdgx2                call by value if not cmblk
        bge  cmtyp(xr),=c$$nm,cdgx2         jump if cmblk only by value
```

```
    cdgex (continued)
    here if expression can be evaluated by name
        jsr  cdgnm                                  generate code by name
        mov  =ornm$,wa                              load return by name word
        brn  cdgx3                                  merge with value case
    here if expression can only be evaluated by value
cdgx2   jsr  cdgvl                                  generate code by value
        mov  =orvl$,wa                              load return by value word
    merge here to construct exblk
cdgx3   jsr  cdwrd                                  generate return word
        jsr  exbld                                  build exblk
        mov  (xs)+,wc                               restore wc
        exi                                         return to cdgex caller
        enp                                         end procedure cdgex
```

```
    cdgnm -- generate code by name
    cdgnm is called during the compilation process to
    generate code by name for an expression. see cdblk
    description for details of code generated. the input
    to cdgnm is an expression tree as generated by expan.
    cdgnm is a recursive procedure which proceeds by making
    recursive calls to generate code for operands.
    (wb)                  integer in range 0 le n le dnamb
    (xr)                  ptr to tree generated by expan
    (wc)                  constant flag (see below)
    jsr  cdgnm            call to generate code by name
    (xr,wa)               destroyed
    (wc)                  set non-zero if non-constant
    wc is set to a non-zero (collectable) value if the
    expression for which code is generated cannot be
    evaluated at compile time, otherwise wc is unchanged.
    the code is generated in the current ccblk (see cdwrd).
cdgnm  prc  r,0                                entry point, recursive
       mov  xl,-(xs)                           save entry xl
       mov  wb,-(xs)                           save entry wb
       chk                                     check for stack overflow
       mov  (xr),wa                            load type word
       beq  wa,=b$cmt,cgn04                    jump if cmblk
       bhi  wa,=b$vr$,cgn02                    jump if simple variable
    merge here for operand yielding value (e.g. constant)
cgn01  erb  212,syntax error:                  value used where name is required
    here for natural variable reference
cgn02  mov  =olvn$,wa                          load variable load call
       jsr  cdwrd                              generate it
       mov  xr,wa                              copy vrblk pointer
       jsr  cdwrd                              generate vrblk pointer
```

```
    cdgnm (continued)
    here to exit with wc set correctly
cgn03   mov  (xs)+,wb                          restore entry wb
        mov  (xs)+,xl                          restore entry xl
        exi                                    return to cdgnm caller
    here for cmblk
cgn04   mov  xr,xl                             copy cmblk pointer
        mov  cmtyp(xr),xr                      load cmblk type
        bge  xr,=c$$nm,cgn01                   error if not name operand
        bsw  xr,c$$nm                          else switch on type
        iff  c$arr,cgn05                       array reference
        iff  c$fnc,cgn08                       function call
        iff  c$def,cgn09                       deferred expression
        iff  c$ind,cgn10                       indirect reference
        iff  c$key,cgn11                       keyword reference
        iff  c$ubo,cgn08                       undefined binary op
        iff  c$uuo,cgn08                       undefined unary op
        esw                                    end switch on cmblk type
    here to generate code for array reference
cgn05   mov  *cmopn,wb                         point to array operand
    loop to generate code for array operand and subscripts
cgn06   jsr  cmgen                             generate code for next operand
        mov  cmlen(xl),wc                      load length of cmblk
        blt  wb,wc,cgn06                       loop till all generated
    generate appropriate array call
        mov  =oaon$,wa                         load one-subscript case call
        beq  wc,*cmar1,cgn07                   jump to exit if one subscript case
        mov  =oamn$,wa                         else load multi-subscript case call
        jsr  cdwrd                             generate call
        mov  wc,wa                             copy cmblk length
        btw  wa                                convert to words
        sub  =cmvls,wa                         calculate number of subscripts
```

```
    cdgnm (continued)
    here to exit generating word (non-constant)
cgn07   mnz  wc                                  set result non-constant
        jsr  cdwrd                               generate word
        brn  cgn03                               back to exit
    here to generate code for functions and undefined oprs
cgn08   mov  xl,xr                               copy cmblk pointer
        jsr  cdgvl                               gen code by value for call
        mov  =ofne$,wa                           get extra call for by name
        brn  cgn07                               back to generate and exit
    here to generate code for defered expression
cgn09   mov  cmrop(xl),xr                        check if variable
        bhi  (xr),=b$vr$,cgn02                   treat *variable as simple var
        mov  xr,xl                               copy ptr to expression tree
```

```
if .cevb
        mov  =num01,wa                           return name
fi
        jsr  cdgex                               else build exblk
        mov  =olex$,wa                           set call to load expr by name
        jsr  cdwrd                               generate it
        mov  xr,wa                               copy exblk pointer
        jsr  cdwrd                               generate exblk pointer
        brn  cgn03                               back to exit
    here to generate code for indirect reference
cgn10   mov  cmrop(xl),xr                        get operand
        jsr  cdgvl                               generate code by value for it
        mov  =oinn$,wa                           load call for indirect by name
        brn  cgn12                               merge
    here to generate code for keyword reference
cgn11   mov  cmrop(xl),xr                        get operand
        jsr  cdgnm                               generate code by name for it
        mov  =okwn$,wa                           load call for keyword by name
    keyword, indirect merge here
cgn12   jsr  cdwrd                               generate code for operator
        brn  cgn03                               exit
        enp                                      end procedure cdgnm
```

```
    cdgvl -- generate code by value
    cdgvl is called during the compilation process to
    generate code by value for an expression. see cdblk
    description for details of the code generated. the input
    to cdgvl is an expression tree as generated by expan.
    cdgvl is a recursive procedure which proceeds by making
    recursive calls to generate code for operands.
    (wb)                  integer in range 0 le n le dnamb
    (xr)                  ptr to tree generated by expan
    (wc)                  constant flag (see below)
    jsr  cdgvl            call to generate code by value
    (xr,wa)               destroyed
    (wc)                  set non-zero if non-constant
    wc is set to a non-zero (collectable) value if the
    expression for which code is generated cannot be
    evaluated at compile time, otherwise wc is unchanged.
    if wc is non-zero on entry, then preevaluation is not
    allowed regardless of the nature of the operand.
    the code is generated in the current ccblk (see cdwrd).
cdgvl  prc  r,0                         entry point, recursive
       mov  (xr),wa                     load type word
       beq  wa,=b$cmt,cgv01             jump if cmblk
       blt  wa,=b$vra,cgv00             jump if icblk, rcblk, scblk
       bnz  vrlen(xr),cgvl0             jump if not system variable
       mov  xr,-(xs)                    stack xr
       mov  vrsvp(xr),xr                point to svblk
       mov  svbit(xr),wa                get svblk property bits
       mov  (xs)+,xr                    recover xr
       anb  btkwv,wa                    check if constant keyword value
       beq  wa,btkwv,cgv00              jump if constant keyword value
    here for variable value reference
cgvl0  mnz  wc                          indicate non-constant value
    merge here for simple constant (icblk,rcblk,scblk)
    and for variables corresponding to constant keywords.
cgv00  mov  xr,wa                       copy ptr to var or constant
       jsr  cdwrd                       generate as code word
       exi                              return to caller
```

```
    cdgvl (continued)
    here for tree node (cmblk)
cgv01   mov  wb,-(xs)                         save entry wb
        mov  xl,-(xs)                         save entry xl
        mov  wc,-(xs)                         save entry constant flag
        mov  cwcof,-(xs)                      save initial code offset
        chk                                   check for stack overflow
    prepare to generate code for cmblk. wc is set to the
    value of cswno (zero if -optimise, 1 if -noopt) to
    start with and is reset non-zero for any non-constant
    code generated. if it is still zero after generating all
    the cmblk code, then its value is computed as the result.
        mov  xr,xl                            copy cmblk pointer
        mov  cmtyp(xr),xr                     load cmblk type
        mov  cswno,wc                         reset constant flag
        ble  xr,=c$pr$,cgv02                  jump if not predicate value
        mnz  wc                               else force non-constant case
    here with wc set appropriately
cgv02   bsw  xr,c$$nv                         switch to appropriate generator
        iff  c$arr,cgv03                      array reference
        iff  c$fnc,cgv05                      function call
        iff  c$def,cgv14                      deferred expression
        iff  c$sel,cgv15                      selection
        iff  c$ind,cgv31                      indirect reference
        iff  c$key,cgv27                      keyword reference
        iff  c$ubo,cgv29                      undefined binop
        iff  c$uuo,cgv30                      undefined unop
        iff  c$bvl,cgv18                      binops with val opds
        iff  c$alt,cgv18                      alternation
        iff  c$uvl,cgv19                      unops with valu opnd
        iff  c$ass,cgv21                      assignment
        iff  c$cnc,cgv24                      concatenation
        iff  c$cnp,cgv24                      concatenation (not pattern match)
        iff  c$unm,cgv27                      unops with name opnd
        iff  c$bvn,cgv26                      binary $ and .
        iff  c$int,cgv31                      interrogation
        iff  c$neg,cgv28                      negation
        iff  c$pmt,cgv18                      pattern match
        esw                                   end switch on cmblk type
```

```
    cdgvl (continued)
    here to generate code for array reference
cgv03   mov  *cmopn,wb                          set offset to array operand
    loop to generate code for array operand and subscripts
cgv04   jsr  cmgen                              gen value code for next operand
        mov  cmlen(xl),wc                       load cmblk length
        blt  wb,wc,cgv04                        loop back if more to go
    generate call to appropriate array reference routine
        mov  =oaov$,wa                          set one subscript call in case
        beq  wc,*cmar1,cgv32                    jump to exit if 1-sub case
        mov  =oamv$,wa                          else set call for multi-subscripts
        jsr  cdwrd                              generate call
        mov  wc,wa                              copy length of cmblk
        sub  *cmvls,wa                          subtract standard length
        btw  wa                                 get number of words
        brn  cgv32                              jump to generate subscript count
    here to generate code for function call
cgv05   mov  *cmvls,wb                          set offset to first argument
    loop to generate code for arguments
cgv06   beq  wb,cmlen(xl),cgv07                 jump if all generated
        jsr  cmgen                              else gen value code for next arg
        brn  cgv06                              back to generate next argument
    here to generate actual function call
cgv07   sub  *cmvls,wb                          get number of arg ptrs (bytes)
        btw  wb                                 convert bytes to words
        mov  cmopn(xl),xr                       load function vrblk pointer
        bnz  vrlen(xr),cgv12                    jump if not system function
        mov  vrsvp(xr),xl                       load svblk ptr if system var
        mov  svbit(xl),wa                       load bit mask
        anb  btffc,wa                           test for fast function call allowed
        zrb  wa,cgv12                           jump if not
```

```
    cdgvl (continued)
    here if fast function call is allowed
        mov  svbit(xl),wa                        reload bit indicators
        anb  btpre,wa                            test for preevaluation ok
        nzb  wa,cgv08                            jump if preevaluation permitted
        mnz  wc                                  else set result non-constant
    test for correct number of args for fast call
cgv08   mov  vrfnc(xr),xl                        load ptr to svfnc field
        mov  fargs(xl),wa                        load svnar field value
        beq  wa,wb,cgv11                         jump if argument count is correct
        bhi  wa,wb,cgv09                         jump if too few arguments given
    here if too many arguments, prepare to generate o$pops
        sub  wa,wb                               get number of extra args
        lct  wb,wb                               set as count to control loop
        mov  =opop$,wa                           set pop call
        brn  cgv10                               jump to common loop
    here if too few arguments, prepare to generate nulls
cgv09   sub  wb,wa                               get number of missing arguments
        lct  wb,wa                               load as count to control loop
        mov  =nulls,wa                           load ptr to null constant
    loop to generate calls to fix argument count
cgv10   jsr  cdwrd                               generate one call
        bct  wb,cgv10                            loop till all generated
    here after adjusting arg count as required
cgv11   mov  xl,wa                               copy pointer to svfnc field
        brn  cgv36                               jump to generate call
```

```
    cdgvl (continued)
    come here if fast call is not permitted
cgv12  mov  =ofns$,wa                        set one arg call in case
       beq  wb,=num01,cgv13                  jump if one arg case
       mov  =ofnc$,wa                        else load call for more than 1 arg
       jsr  cdwrd                            generate it
       mov  wb,wa                            copy argument count
    one arg case merges here
cgv13  jsr  cdwrd                            generate =o$fns or arg count
       mov  xr,wa                            copy vrblk pointer
       brn  cgv32                            jump to generate vrblk ptr
    here for deferred expression
cgv14  mov  cmrop(xl),xl                     point to expression tree
```

---

if **.cevb**

```
       zer  wa                               return value
```

fi

```
       jsr  cdgex                            build exblk or seblk
       mov  xr,wa                            copy block ptr
       jsr  cdwrd                            generate ptr to exblk or seblk
       brn  cgv34                            jump to exit, constant test
    here to generate code for selection
cgv15  zer  -(xs)                            zero ptr to chain of forward jumps
       zer  -(xs)                            zero ptr to prev o$slc forward ptr
       mov  *cmvls,wb                        point to first alternative
       mov  =osla$,wa                        set initial code word
    0(xs)              is the offset to the previous word
                       which requires filling in with an
                       offset to the following o$slc,o$sld
    1(xs)              is the head of a chain of offset
                       pointers indicating those locations
                       to be filled with offsets past
                       the end of all the alternatives
cgv16  jsr  cdwrd                            generate o$slc (o$sla first time)
       mov  cwcof,(xs)                       set current loc as ptr to fill in
       jsr  cdwrd                            generate garbage word there for now
       jsr  cmgen                            gen value code for alternative
       mov  =oslb$,wa                        load o$slb pointer
       jsr  cdwrd                            generate o$slb call
       mov  num01(xs),wa                     load old chain ptr
       mov  cwcof,num01(xs)                  set current loc as new chain head
       jsr  cdwrd                            generate forward chain link
```

```
    cdgvl (continued)
    now to fill in the skip offset to o$slc,o$sld
          mov   (xs),xr                 load offset to word to plug
          add   r$ccb,xr                point to actual location to plug
          mov   cwcof,(xr)              plug proper offset in
          mov   =oslc$,wa               load o$slc ptr for next alternative
          mov   wb,xr                   copy offset (destroy garbage xr)
          ica   xr                      bump extra time for test
          blt   xr,cmlen(xl),cgv16      loop back if not last alternative
    here to generate code for last alternative
          mov   =osld$,wa               get header call
          jsr   cdwrd                   generate o$sld call
          jsr   cmgen                   generate code for last alternative
          ica   xs                      pop offset ptr
          mov   (xs)+,xr                load chain ptr
    loop to plug offsets past structure
cgv17     add   r$ccb,xr                make next ptr absolute
          mov   (xr),wa                 load forward ptr
          mov   cwcof,(xr)              plug required offset
          mov   wa,xr                   copy forward ptr
          bnz   wa,cgv17                loop back if more to go
          brn   cgv33                   else jump to exit (not constant)
    here for binary ops with value operands
cgv18     mov   cmlop(xl),xr            load left operand pointer
          jsr   cdgvl                   gen value code for left operand
    here for unary ops with value operand (binops merge)
cgv19     mov   cmrop(xl),xr            load right (only) operand ptr
          jsr   cdgvl                   gen code by value
```

```
    cdgvl (continued)
    merge here to generate operator call from cmopn field
cgv20   mov  cmopn(xl),wa                         load operator call pointer
        brn  cgv36                                jump to generate it with cons test
    here for assignment
cgv21   mov  cmlop(xl),xr                         load left operand pointer
        blo  (xr),=b$vr$,cgv22                    jump if not variable
    here for assignment to simple variable
        mov  cmrop(xl),xr                         load right operand ptr
        jsr  cdgvl                                generate code by value
        mov  cmlop(xl),wa                         reload left operand vrblk ptr
        add  *vrsto,wa                            point to vrsto field
        brn  cgv32                                jump to generate store ptr
    here if not simple variable assignment
cgv22   jsr  expap                                test for pattern match on left side
        ppm  cgv23                                jump if not pattern match
    here for pattern replacement
        mov  cmrop(xr),cmlop(xl)                  save pattern ptr in safe place
        mov  cmlop(xr),xr                         load subject ptr
        jsr  cdgnm                                gen code by name for subject
        mov  cmlop(xl),xr                         load pattern ptr
        jsr  cdgvl                                gen code by value for pattern
        mov  =opmn$,wa                            load match by name call
        jsr  cdwrd                                generate it
        mov  cmrop(xl),xr                         load replacement value ptr
        jsr  cdgvl                                gen code by value
        mov  =orpl$,wa                            load replace call
        brn  cgv32                                jump to gen and exit (not constant)
    here for assignment to complex variable
cgv23   mnz  wc                                   inhibit pre-evaluation
        jsr  cdgnm                                gen code by name for left side
        brn  cgv31                                merge with unop circuit
```

```
    cdgvl (continued)
    here for concatenation
cgv24   mov  cmlop(xl),xr                load left operand ptr
        bne  (xr),=b$cmt,cgv18           ordinary binop if not cmblk
        mov  cmtyp(xr),wb                load cmblk type code
        beq  wb,=c$int,cgv25             special case if interrogation
        beq  wb,=c$neg,cgv25             or negation
        bne  wb,=c$fnc,cgv18             else ordinary binop if not function
        mov  cmopn(xr),xr                else load function vrblk ptr
        bnz  vrlen(xr),cgv18             ordinary binop if not system var
        mov  vrsvp(xr),xr                else point to svblk
        mov  svbit(xr),wa                load bit indicators
        anb  btprd,wa                    test for predicate function
        zrb  wa,cgv18                    ordinary binop if not
    here if left arg of concatenation is predicate function
cgv25   mov  cmlop(xl),xr                reload left arg
        jsr  cdgvl                       gen code by value
        mov  =opop$,wa                   load pop call
        jsr  cdwrd                       generate it
        mov  cmrop(xl),xr                load right operand
        jsr  cdgvl                       gen code by value as result code
        brn  cgv33                       exit (not constant)
    here to generate code for pattern, immediate assignment
cgv26   mov  cmlop(xl),xr                load left operand
        jsr  cdgvl                       gen code by value, merge
    here for unops with arg by name (binary $ . merge)
cgv27   mov  cmrop(xl),xr                load right operand ptr
        jsr  cdgnm                       gen code by name for right arg
        mov  cmopn(xl),xr                get operator code word
        bne  (xr),=o$kwv,cgv20           gen call unless keyword value
```

```
    cdgvl (continued)
    here for keyword by value. this is constant only if
    the operand is one of the special system variables with
    the svckw bit set to indicate a constant keyword value.
    note that the only constant operand by name is a variable
       bnz   wc,cgv20              gen call if non-constant (not var)
       mnz   wc                    else set non-constant in case
       mov   cmrop(xl),xr          load ptr to operand vrblk
       bnz   vrlen(xr),cgv20       gen (non-constant) if not sys var
       mov   vrsvp(xr),xr          else load ptr to svblk
       mov   svbit(xr),wa          load bit mask
       anb   btckw,wa              test for constant keyword
       zrb   wa,cgv20              go gen if not constant
       zer   wc                    else set result constant
       brn   cgv20                 and jump back to generate call
    here to generate code for negation
cgv28  mov   =onta$,wa             get initial word
       jsr   cdwrd                 generate it
       mov   cwcof,wb              save next offset
       jsr   cdwrd                 generate gunk word for now
       mov   cmrop(xl),xr          load right operand ptr
       jsr   cdgvl                 gen code by value
       mov   =ontb$,wa             load end of evaluation call
       jsr   cdwrd                 generate it
       mov   wb,xr                 copy offset to word to plug
       add   r$ccb,xr              point to actual word to plug
       mov   cwcof,(xr)            plug word with current offset
       mov   =ontc$,wa             load final call
       brn   cgv32                 jump to generate it (not constant)
    here to generate code for undefined binary operator
cgv29  mov   cmlop(xl),xr          load left operand ptr
       jsr   cdgvl                 generate code by value
```

```
    cdgvl (continued)
    here to generate code for undefined unary operator
cgv30   mov  =c$uo$,wb                          set unop code + 1
        sub  cmtyp(xl),wb                       set number of args (1 or 2)
    merge here for undefined operators
        mov  cmrop(xl),xr                       load right (only) operand pointer
        jsr  cdgvl                              gen value code for right operand
        mov  cmopn(xl),xr                       load pointer to operator dv
        mov  dvopn(xr),xr                       load pointer offset
        wtb  xr                                 convert word offset to bytes
        add  =r$uba,xr                          point to proper function ptr
        sub  *vrfnc,xr                          set standard function offset
        brn  cgv12                              merge with function call circuit
    here to generate code for interrogation, indirection
cgv31   mnz  wc                                 set non constant
        brn  cgv19                              merge
    here to exit generating a word, result not constant
cgv32   jsr  cdwrd                              generate word, merge
    here to exit with no word generated, not constant
cgv33   mnz  wc                                 indicate result is not constant
    common exit point
cgv34   ica  xs                                 pop initial code offset
        mov  (xs)+,wa                           restore old constant flag
        mov  (xs)+,xl                           restore entry xl
        mov  (xs)+,wb                           restore entry wb
        bnz  wc,cgv35                           jump if not constant
        mov  wa,wc                              else restore entry constant flag
    here to return after dealing with wc setting
cgv35   exi                                     return to cdgvl caller
    exit here to generate word and test for constant
cgv36   jsr  cdwrd                              generate word
        bnz  wc,cgv34                           jump to exit if not constant
```

```
    cdgvl (continued)
    here to preevaluate constant sub-expression
          mov  =orvl$,wa              load call to return value
          jsr  cdwrd                  generate it
          mov  (xs),xl                load initial code offset
          jsr  exbld                  build exblk for expression
          zer  wb                     set to evaluate by value
          jsr  evalx                  evaluate expression
          ppm                         should not fail
          mov  (xr),wa                load type word of result
          blo  wa,=p$aaa,cgv37        jump if not pattern
          mov  =olpt$,wa              else load special pattern load call
          jsr  cdwrd                  generate it
    merge here to generate pointer to resulting constant
cgv37     mov  xr,wa                  copy constant pointer
          jsr  cdwrd                  generate ptr
          zer  wc                     set result constant
          brn  cgv34                  jump back to exit
          enp                         end procedure cdgvl
```

```
    cdwrd -- generate one word of code
    cdwrd writes one word into the current code block under
    construction. a new, larger, block is allocated if there
    is insufficient room in the current block. cdwrd ensures
```
---
if **.csln**
```
    that there are at least four words left in the block
```
else
```
    that there are at least three words left in the block
```
fi
```
    after entering the new word. this guarantees that any
    extra space at the end can be split off as a ccblk.
    (wa)                 word to be generated
    jsr  cdwrd           call to generate word
cdwrd   prc  e,0                                  entry point
        mov  xr,-(xs)                             save entry xr
        mov  wa,-(xs)                             save code word to be generated
    merge back here after allocating larger block
cdwd1   mov  r$ccb,xr                             load ptr to ccblk being built
        bnz  xr,cdwd2                             jump if block allocated
    here we allocate an entirely fresh block
        mov  *e$cbs,wa                            load initial length
        jsr  alloc                                allocate ccblk
        mov  =b$cct,(xr)                          store type word
        mov  *cccod,cwcof                         set initial offset
        mov  wa,cclen(xr)                         store block length
```
---
if **.csln**
```
        zer  ccsln(xr)                            zero line number
```
fi
```
        mov  xr,r$ccb                             store ptr to new block
    here we have a block we can use
cdwd2   mov  cwcof,wa                             load current offset
```
---
if **.csln**
```
        add  *num05,wa                            adjust for test (five words)
```
else
```
        add  *num04,wa                            adjust for test (four words)
```
fi
```
        blo  wa,cclen(xr),cdwd4                   jump if room in this block
    here if no room in current block
        bge  wa,mxlen,cdwd5                       jump if already at max size
        add  *e$cbs,wa                            else get new size
        mov  xl,-(xs)                             save entry xl
        mov  xr,xl                                copy pointer
        blt  wa,mxlen,cdwd3                       jump if not too large
        mov  mxlen,wa                             else reset to max allowed size
```

```
    cdwrd (continued)
    here with new block size in wa
cdwd3   jsr   alloc                          allocate new block
        mov   xr,r$ccb                       store pointer to new block
        mov   =b$cct,(xr)+                   store type word in new block
        mov   wa,(xr)+                       store block length
```

---

if **.csln**

```
        mov   ccsln(xl),(xr)+                copy source line number word
```

fi

```
        add   *ccuse,xl                      point to ccuse,cccod fields in old
        mov   (xl),wa                        load ccuse value
        mvw                                  copy useful words from old block
        mov   (xs)+,xl                       restore xl
        brn   cdwd1                          merge back to try again
    here with room in current block
cdwd4   mov   cwcof,wa                       load current offset
        ica   wa                             get new offset
        mov   wa,cwcof                       store new offset
        mov   wa,ccuse(xr)                   store in ccblk for gbcol
        dca   wa                             restore ptr to this word
        add   wa,xr                          point to current entry
        mov   (xs)+,wa                       reload word to generate
        mov   wa,(xr)                        store word in block
        mov   (xs)+,xr                       restore entry xr
        exi                                  return to caller
    here if compiled code is too long for cdblk
cdwd5   erb   213,syntax error:              statement is too complicated.
        enp                                  end procedure cdwrd
```

```
    cmgen -- generate code for cmblk ptr
    cmgen is a subsidiary procedure used to generate value
    code for a cmblk ptr from the main code generators.
    (xl)                  cmblk pointer
    (wb)                  offset to pointer in cmblk
    jsr  cmgen            call to generate code
    (xr,wa)               destroyed
    (wb)                  bumped by one word
cmgen   prc  r,0                                   entry point, recursive
        mov  xl,xr                                 copy cmblk pointer
        add  wb,xr                                 point to cmblk pointer
        mov  (xr),xr                               load cmblk pointer
        jsr  cdgvl                                 generate code by value
        ica  wb                                    bump offset
        exi                                        return to caller
        enp                                        end procedure cmgen
```

```
cmpil (compile source code)
cmpil is used to convert snobol4 source code to internal
form (see cdblk format). it is used both for the initial
compile and at run time by the code and convert functions
this procedure has control for the entire duration of
initial compilation. an error in any procedure called
during compilation will lead first to the error section
and ultimately back here for resumed compilation. the
re-entry points after an error are specially labelled -
cmpce                 resume after control card error
cmple                 resume after label error
cmpse                 resume after statement error
jsr  cmpil            call to compile code
(xr)                  ptr to cdblk for entry statement
(xl,wa,wb,wc,ra)      destroyed
the following global variables are referenced
cmpln                 line number of first line of
                      statement to be compiled
cmpsn                 number of next statement
                      to be compiled.
cswxx                 control card switch values are
                      changed when relevant control
                      cards are met.
cwcof                 offset to next word in code block
                      being built (see cdwrd).
lstsn                 number of statement most recently
                      compiled (initially set to zero).
r$cim                 current (initial) compiler image
                      (zero for initial compile call)
r$cni                 used to point to following image.
                      (see readr procedure).
scngo                 goto switch for scane procedure
scnil                 length of current image excluding
                      characters removed by -input.
scnpt                 current scan offset, see scane.
scnrs                 rescan switch for scane procedure.
scnse                 offset (in r$cim) of most recently
                      scanned element. set zero if not
                      currently scanning items
```

```
cmpil (continued)
stage              stgic  initial compile in progress
                   stgxc  code/convert compile
                   stgev  building exblk for eval
                   stgxt  execute time (outside compile)
                   stgce  initial compile after end line
                   stgxe  execute compile after end line
cmpil also uses a fixed number of locations on the
main stack as follows. (the definitions of the actual
offsets are in the definitions section).
cmstm(xs)            pointer to expan tree for body of
                     statement (see expan procedure).
cmsgo(xs)            pointer to tree representation of
                     success goto (see procedure scngo)
                     zero if no success goto is given
cmfgo(xs)            like cmsgo for failure goto.
cmcgo(xs)            set non-zero only if there is a
                     conditional goto. used for -fail,
                     -nofail code generation.
cmpcd(xs)            pointer to cdblk for previous
                     statement. zero for 1st statement.
cmffp(xs)            set non-zero if cdfal in previous
                     cdblk needs filling with forward
                     pointer, else set to zero.
cmffc(xs)            same as cmffp for current cdblk
cmsop(xs)            offset to word in previous cdblk
                     to be filled in with forward ptr
                     to next cdblk for success goto.
                     zero if no fill in is required.
cmsoc(xs)            same as cmsop for current cdblk.
cmlbl(xs)            pointer to vrblk for label of
                     current statement. zero if no label
cmtra(xs)            pointer to cdblk for entry stmnt.
```

```
    cmpil (continued)
    entry point
cmpil   prc   e,0                        entry point
        lct   wb,=cmnen                  set number of stack work locations
    loop to initialize stack working locations
cmp00   zer   -(xs)                      store a zero, make one entry
        bct   wb,cmp00                   loop back until all set
        mov   xs,cmpxs                   save stack pointer for error sec
        sss   cmpss                      save s-r stack pointer if any
    loop through statements
cmp01   mov   scnpt,wb                   set scan pointer offset
        mov   wb,scnse                   set start of element location
        mov   =ocer$,wa                  point to compile error call
        jsr   cdwrd                      generate as temporary cdfal
        blt   wb,scnil,cmp04             jump if chars left on this image
    loop here after comment or control card
    also special entry after control card error
cmpce   zer   xr                         clear possible garbage xr value
```

```
if .cinc
        bnz   cnind,cmpc2                if within include file
fi
        bne   stage,=stgic,cmp02         skip unless initial compile
cmpc2   jsr   readr                      read next input image
        bze   xr,cmp09                   jump if no input available
        jsr   nexts                      acquire next source image
        mov   cmpsn,lstsn                store stmt no for use by listr
        mov   rdcln,cmpln                store line number at start of stmt
        zer   scnpt                      reset scan pointer
        brn   cmp04                      go process image
    for execute time compile, permit embedded control cards
    and comments (by skipping to next semi-colon)
cmp02   mov   r$cim,xr                   get current image
        mov   scnpt,wb                   get current offset
        plc   xr,wb                      prepare to get chars
    skip to semi-colon
cmp03   bge   scnpt,scnil,cmp09          end loop if end of image
        lch   wc,(xr)+                   get char
        icv   scnpt                      advance offset
        bne   wc,=ch$sm,cmp03            loop if not semi-colon
```

```
    cmpil (continued)
    here with image available to scan. note that if the input
    string is null, then everything is ok since null is
    actually assembled as a word of blanks.
cmp04   mov  r$cim,xr                point to current image
        mov  scnpt,wb                load current offset
        mov  wb,wa                   copy for label scan
        plc  xr,wb                   point to first character
        lch  wc,(xr)+                load first character
        beq  wc,=ch$sm,cmp12         no label if semicolon
        beq  wc,=ch$as,cmpce         loop back if comment card
        beq  wc,=ch$mn,cmp32         jump if control card
        mov  r$cim,r$cmp             about to destroy r$cim
        mov  =cmlab,xl               point to label work string
        mov  xl,r$cim                scane is to scan work string
        psc  xl                      point to first character position
        sch  wc,(xl)+                store char just loaded
        mov  =ch$sm,wc               get a semicolon
        sch  wc,(xl)                 store after first char
        csc  xl                      finished character storing
        zer  xl                      clear pointer
        zer  scnpt                   start at first character
        mov  scnil,-(xs)             preserve image length
        mov  =num02,scnil            read 2 chars at most
        jsr  scane                   scan first char for type
        mov  (xs)+,scnil             restore image length
        mov  xl,wc                   note return code
        mov  r$cmp,xl                get old r$cim
        mov  xl,r$cim                put it back
        mov  wb,scnpt                reinstate offset
        bnz  scnbl,cmp12             blank seen - cant be label
        mov  xl,xr                   point to current image
        plc  xr,wb                   point to first char again
        beq  wc,=t$var,cmp06         ok if letter
        beq  wc,=t$con,cmp06         ok if digit
    drop in or jump from error section if scane failed
cmple   mov  r$cmp,r$cim             point to bad line
        erb  214,bad label or        misplaced continuation line
    loop to scan label
cmp05   beq  wc,=ch$sm,cmp07         skip if semicolon
        icv  wa                      bump offset
        beq  wa,scnil,cmp07          jump if end of image (label end)
```

```
    cmpil (continued)
    enter loop at this point
cmp06   lch   wc,(xr)+                     else load next character
```

---

if **.caht**

```
        beq   wc,=ch$ht,cmp07              jump if horizontal tab
```

fi

---

if **.cavt**

```
        beq   wc,=ch$vt,cmp07              jump if vertical tab
```

fi

```
        bne   wc,=ch$bl,cmp05              loop back if non-blank
    here after scanning out label
cmp07   mov   wa,scnpt                     save updated scan offset
        sub   wb,wa                        get length of label
        bze   wa,cmp12                     skip if label length zero
        zer   xr                           clear garbage xr value
        jsr   sbstr                        build scblk for label name
        jsr   gtnvr                        locate/contruct vrblk
        ppm                                dummy (impossible) error return
        mov   xr,cmlbl(xs)                 store label pointer
        bnz   vrlen(xr),cmp11              jump if not system label
        bne   vrsvp(xr),=v$end,cmp11       jump if not end label
    here for end label scanned out
        add   =stgnd,stage                 adjust stage appropriately
        jsr   scane                        scan out next element
        beq   xl,=t$smc,cmp10              jump if end of image
        bne   xl,=t$var,cmp08              else error if not variable
    here check for valid initial transfer
        beq   vrlbl(xr),=stndl,cmp08       jump if not defined (error)
        mov   vrlbl(xr),cmtra(xs)          else set initial entry pointer
        jsr   scane                        scan next element
        beq   xl,=t$smc,cmp10              jump if ok (end of image)
    here for bad transfer label
cmp08   erb   215,syntax error:            undefined or erroneous entry label
    here for end of input (no end label detected)
cmp09   zer   xr                           clear garbage xr value
        add   =stgnd,stage                 adjust stage appropriately
        beq   stage,=stgxe,cmp10           jump if code call (ok)
        erb   216,syntax error:            missing end line
    here after processing end line (merge here on end error)
cmp10   mov   =ostp$,wa                    set stop call pointer
        jsr   cdwrd                        generate as statement call
        brn   cmpse                        jump to generate as failure
```

```
    cmpil (continued)
    here after processing label other than end
cmp11  bne  stage,=stgic,cmp12          jump if code call - redef. ok
       beq  vrlbl(xr),=stndl,cmp12      else check for redefinition
       zer  cmlbl(xs)                   leave first label decln undisturbed
       erb  217,syntax error:           duplicate label
    here after dealing with label
    null statements and statements just containing a
    constant subject are optimized out by resetting the
    current ccblk to empty.
cmp12  zer  wb                          set flag for statement body
       jsr  expan                       get tree for statement body
       mov  xr,cmstm(xs)                store for later use
       zer  cmsgo(xs)                   clear success goto pointer
       zer  cmfgo(xs)                   clear failure goto pointer
       zer  cmcgo(xs)                   clear conditional goto flag
       jsr  scane                       scan next element
       beq  xl,=t$col,cmp13             jump if colon (goto)
       bnz  cswno,cmp18                 jump if not optimizing
       bnz  cmlbl(xs),cmp18             jump if label present
       mov  cmstm(xs),xr                load tree ptr for statement body
       mov  (xr),wa                     load type word
       beq  wa,=b$cmt,cmp18             jump if cmblk
       bge  wa,=b$vra,cmp18             jump if not icblk, scblk, or rcblk
       mov  r$ccb,xl                    load ptr to ccblk
       mov  *cccod,ccuse(xl)            reset use offset in ccblk
       mov  *cccod,cwcof                and in global
       icv  cmpsn                       bump statement number
       brn  cmp01                       generate no code for statement
    loop to process goto fields
cmp13  mnz  scngo                       set goto flag
       jsr  scane                       scan next element
       beq  xl,=t$smc,cmp31             jump if no fields left
       beq  xl,=t$sgo,cmp14             jump if s for success goto
       beq  xl,=t$fgo,cmp16             jump if f for failure goto
    here for unconditional goto (i.e. not f or s)
       mnz  scnrs                       set to rescan element not f,s
       jsr  scngf                       scan out goto field
       bnz  cmfgo(xs),cmp17             error if fgoto already
       mov  xr,cmfgo(xs)                else set as fgoto
       brn  cmp15                       merge with sgoto circuit
    here for success goto
cmp14  jsr  scngf                       scan success goto field
       mov  =num01,cmcgo(xs)            set conditional goto flag
    uncontional goto merges here
cmp15  bnz  cmsgo(xs),cmp17             error if sgoto already given
       mov  xr,cmsgo(xs)                else set sgoto
       brn  cmp13                       loop back for next goto field
    here for failure goto
cmp16  jsr  scngf                       scan goto field
       mov  =num01,cmcgo(xs)            set conditonal goto flag
       bnz  cmfgo(xs),cmp17             error if fgoto already given
       mov  xr,cmfgo(xs)                else store fgoto pointer
```

brn	`cmp13`	loop back for next field

```
    cmpil (continued)
    here for duplicated goto field
cmp17   erb   218,syntax error:                       duplicated goto field
    here to generate code
cmp18   zer   scnse                                   stop positional error flags
        mov   cmstm(xs),xr                            load tree ptr for statement body
        zer   wb                                      collectable value for wb for cdgvl
        zer   wc                                      reset constant flag for cdgvl
        jsr   expap                                   test for pattern match
        ppm   cmp19                                   jump if not pattern match
        mov   =opms$,cmopn(xr)                        else set pattern match pointer
        mov   =opms$,cmopn(xr)                        else set pattern match pointer
    here after dealing with special pattern match case
cmp19   jsr   cdgvl                                   generate code for body of statement
        mov   cmsgo(xs),xr                            load sgoto pointer
        mov   xr,wa                                   copy it
        bze   xr,cmp21                                jump if no success goto
        zer   cmsoc(xs)                               clear success offset fillin ptr
        bhi   xr,state,cmp20                          jump if complex goto
    here for simple success goto (label)
        add   *vrtra,wa                               point to vrtra field as required
        jsr   cdwrd                                   generate success goto
        brn   cmp22                                   jump to deal with fgoto
    here for complex success goto
cmp20   beq   xr,cmfgo(xs),cmp22                      no code if same as fgoto
        zer   wb                                      else set ok value for cdgvl in wb
        jsr   cdgcg                                   generate code for success goto
        brn   cmp22                                   jump to deal with fgoto
    here for no success goto
cmp21   mov   cwcof,cmsoc(xs)                         set success fill in offset
        mov   =ocer$,wa                               point to compile error call
        jsr   cdwrd                                   generate as temporary value
```

```
    cmpil (continued)
    here to deal with failure goto
cmp22   mov  cmfgo(xs),xr                    load failure goto pointer
        mov  xr,wa                           copy it
        zer  cmffc(xs)                       set no fill in required yet
        bze  xr,cmp23                        jump if no failure goto given
        add  *vrtra,wa                       point to vrtra field in case
        blo  xr,state,cmpse                  jump to gen if simple fgoto
    here for complex failure goto
        mov  cwcof,wb                        save offset to o$gof call
        mov  =ogof$,wa                       point to failure goto call
        jsr  cdwrd                           generate
        mov  =ofif$,wa                       point to fail in fail word
        jsr  cdwrd                           generate
        jsr  cdgcg                           generate code for failure goto
        mov  wb,wa                           copy offset to o$gof for cdfal
        mov  =b$cdc,wb                       set complex case cdtyp
        brn  cmp25                           jump to build cdblk
    here if no failure goto given
cmp23   mov  =ounf$,wa                       load unexpected failure call in cas
        mov  cswfl,wc                        get -nofail flag
        orb  cmcgo(xs),wc                    check if conditional goto
        zrb  wc,cmpse                        jump if -nofail and no cond. goto
        mnz  cmffc(xs)                       else set fill in flag
        mov  =ocer$,wa                       and set compile error for temporary
    merge here with cdfal value in wa, simple cdblk
    also special entry after statement error
cmpse   mov  =b$cds,wb                       set cdtyp for simple case
```

```
    cmpil (continued)
    merge here to build cdblk
    (wa)                 cdfal value to be generated
    (wb)                 cdtyp value to be generated
    at this stage, we chop off an appropriate chunk of the
    current ccblk and convert it into a cdblk. the remainder
    of the ccblk is reformatted to be the new ccblk.
cmp25   mov  r$ccb,xr                          point to ccblk
        mov  cmlbl(xs),xl                      get possible label pointer
        bze  xl,cmp26                          skip if no label
        zer  cmlbl(xs)                         clear flag for next statement
        mov  xr,vrlbl(xl)                      put cdblk ptr in vrblk label field
    merge after doing label
cmp26   mov  wb,(xr)                           set type word for new cdblk
        mov  wa,cdfal(xr)                      set failure word
        mov  xr,xl                             copy pointer to ccblk
        mov  ccuse(xr),wb                      load length gen (= new cdlen)
        mov  cclen(xr),wc                      load total ccblk length
        add  wb,xl                             point past cdblk
        sub  wb,wc                             get length left for chop off
        mov  =b$cct,(xl)                       set type code for new ccblk at end
        mov  *cccod,ccuse(xl)                  set initial code offset
        mov  *cccod,cwcof                      reinitialise cwcof
        mov  wc,cclen(xl)                      set new length
        mov  xl,r$ccb                          set new ccblk pointer
```

---

if **.csln**

```
        zer  ccsln(xl)                         initialize new line number
        mov  cmpln,cdsln(xr)                   set line number in old block
```

fi

```
        mov  cmpsn,cdstm(xr)                   set statement number
        icv  cmpsn                             bump statement number
    set pointers in previous code block as required
        mov  cmpcd(xs),xl                      load ptr to previous cdblk
        bze  cmffp(xs),cmp27                   jump if no failure fill in required
        mov  xr,cdfal(xl)                      else set failure ptr in previous
    here to deal with success forward pointer
cmp27   mov  cmsop(xs),wa                      load success offset
        bze  wa,cmp28                          jump if no fill in required
        add  wa,xl                             else point to fill in location
        mov  xr,(xl)                           store forward pointer
        zer  xl                                clear garbage xl value
```

```
    cmpil (continued)
    now set fill in pointers for this statement
cmp28   mov  cmffc(xs),cmffp(xs)         copy failure fill in flag
        mov  cmsoc(xs),cmsop(xs)         copy success fill in offset
        mov  xr,cmpcd(xs)                save ptr to this cdblk
        bnz  cmtra(xs),cmp29             jump if initial entry already set
        mov  xr,cmtra(xs)                else set ptr here as default
    here after compiling one statement
cmp29   blt  stage,=stgce,cmp01          jump if not end line just done
        bze  cswls,cmp30                 skip if -nolist
        jsr  listr                       list last line
    return
cmp30   mov  cmtra(xs),xr                load initial entry cdblk pointer
        add  *cmnen,xs                   pop work locations off stack
        exi                              and return to cmpil caller
    here at end of goto field
cmp31   mov  cmfgo(xs),wb                get fail goto
        orb  cmsgo(xs),wb                or in success goto
        bnz  wb,cmp18                    ok if non-null field
        erb  219,syntax error:           empty goto field
    control card found
cmp32   icv  wb                          point past ch$mn
        jsr  cncrd                       process control card
        zer  scnse                       clear start of element loc.
        brn  cmpce                       loop for next statement
        enp                              end procedure cmpil
```

```
    cncrd -- control card processor
    called to deal with control cards
    r$cim                points to current image
    (wb)                 offset to 1st char of control card
    jsr  cncrd           call to process control cards
    (xl,xr,wa,wb,wc,ia)  destroyed
cncrd  prc  e,0                          entry point
       mov  wb,scnpt                     offset for control card scan
       mov  =ccnoc,wa                    number of chars for comparison
       ctw  wa,0                         convert to word count
       mov  wa,cnswc                     save word count
    loop here if more than one control card
cnc01  bge  scnpt,scnil,cnc09            return if end of image
       mov  r$cim,xr                     point to image
       plc  xr,scnpt                     char ptr for first char
       lch  wa,(xr)+                     get first char
```

---

if **.culc**

```
       flc  wa                           fold to upper case
```

fi

```
       beq  wa,=ch$li,cnc07              special case of -inxxx
cnc0a  mnz  scncc                        set flag for scane
       jsr  scane                        scan card name
       zer  scncc                        clear scane flag
       bnz  xl,cnc06                     fail unless control card name
       mov  =ccnoc,wa                    no. of chars to be compared
```

---

if **.cicc**

```
       blt  sclen(xr),wa,cnc08           fail if too few chars
```

else

```
       blt  sclen(xr),wa,cnc06           fail if too few chars
```

fi

```
       mov  xr,xl                        point to control card name
       zer  wb                           zero offset for substring
       jsr  sbstr                        extract substring for comparison
```

---

if **.culc**

```
       mov  sclen(xr),wa                 reload length
       jsr  flstg                        fold to upper case
```

fi

```
       mov  xr,cnscc                     keep control card substring ptr
       mov  =ccnms,xr                    point to list of standard names
       zer  wb                           initialise name offset
       lct  wc,=cc$nc                    number of standard names
    try to match name
cnc02  mov  cnscc,xl                     point to name
       lct  wa,cnswc                     counter for inner loop
       brn  cnc04                        jump into loop
    inner loop to match card name chars
cnc03  ica  xr                           bump standard names ptr
       ica  xl                           bump name pointer
    here to initiate the loop
cnc04  cne  schar(xl),(xr),cnc05         comp. up to cfp$c chars at once
       bct  wa,cnc03                     loop if more words to compare
```

```
    cncrd (continued)
    matched - branch on card offset
        mov  wb,xl                                    get name offset
```
---
if **.cicc**
```
        bsw  xl,cc$nc,cnc08                           switch
```
else
```
        bsw  xl,cc$nc,cnc06                           switch
```
fi

---
if **.culc**
```
        iff  cc$ca,cnc37                              -case
```
fi

---
if **.ccmc**
```
        iff  cc$co,cnc39                              -compare
```
fi
```
        iff  cc$do,cnc10                              -double
        iff  cc$du,cnc11                              -dump
```
---
if **.cinc**
```
        iff  cc$cp,cnc41                              -copy
```
fi
```
        iff  cc$ej,cnc12                              -eject
        iff  cc$er,cnc13                              -errors
        iff  cc$ex,cnc14                              -execute
        iff  cc$fa,cnc15                              -fail
```
---
if **.cinc**
```
        iff  cc$in,cnc41                              -include
```
fi

---
if **.csln**
```
        iff  cc$ln,cnc44                              -line
```
fi
```
        iff  cc$li,cnc16                              -list
        iff  cc$nr,cnc17                              -noerrors
        iff  cc$nx,cnc18                              -noexecute
        iff  cc$nf,cnc19                              -nofail
        iff  cc$nl,cnc20                              -nolist
        iff  cc$no,cnc21                              -noopt
        iff  cc$np,cnc22                              -noprint
        iff  cc$op,cnc24                              -optimise
        iff  cc$pr,cnc25                              -print
        iff  cc$si,cnc27                              -single
        iff  cc$sp,cnc28                              -space
        iff  cc$st,cnc31                              -stitle
        iff  cc$ti,cnc32                              -title
        iff  cc$tr,cnc36                              -trace
        esw                                           end switch
    not matched yet. align std names ptr and try again
cnc05   ica  xr                                       bump standard names ptr
        bct  wa,cnc05                                 loop
        icv  wb                                       bump names offset
        bct  wc,cnc02                                 continue if more names
```
---
if **.cicc**

```
        brn   cnc08                           ignore unrecognized control card
fi
    invalid control card name
cnc06   erb   247,invalid control             statement
    special processing for -inxxx
cnc07   lch   wa,(xr)+                        get next char
```

```
if .culc
        flc   wa                              fold to upper case
fi
        bne   wa,=ch$ln,cnc0a                 if not letter n
        lch   wa,(xr)                         get third char
        blt   wa,=ch$d0,cnc0a                 if not digit
        bgt   wa,=ch$d9,cnc0a                 if not digit
        add   =num02,scnpt                    bump offset past -in
        jsr   scane                           scan integer after -in
        mov   xr,-(xs)                        stack scanned item
        jsr   gtsmi                           check if integer
        ppm   cnc06                           fail if not integer
        ppm   cnc06                           fail if negative or large
        mov   xr,cswin                        keep integer
```

```
    cncrd (continued)
    check for more control cards before returning
cnc08   mov  scnpt,wa                    preserve in case xeq time compile
        jsr  scane                       look for comma
        beq  xl,=t$cma,cnc01             loop if comma found
        mov  wa,scnpt                    restore scnpt in case xeq time
    return point
cnc09   exi                              return
    -double
cnc10   mnz  cswdb                       set switch
        brn  cnc08                       merge
    -dump
    this is used for system debugging . it has the effect of
    producing a core dump at compilation time
cnc11   jsr  sysdm                       call dumper
        brn  cnc09                       finished
    -eject
cnc12   bze  cswls,cnc09                 return if -nolist
        jsr  prtps                       eject
        jsr  listt                       list title
        brn  cnc09                       finished
    -errors
cnc13   zer  cswer                       clear switch
        brn  cnc08                       merge
    -execute
cnc14   zer  cswex                       clear switch
        brn  cnc08                       merge
    -fail
cnc15   mnz  cswfl                       set switch
        brn  cnc08                       merge
    -list
cnc16   mnz  cswls                       set switch
        beq  stage,=stgic,cnc08          done if compile time
    list code line if execute time compile
        zer  lstpf                       permit listing
        jsr  listr                       list line
        brn  cnc08                       merge
```

```
    cncrd (continued)
    -noerrors
cnc17   mnz  cswer                                     set switch
        brn  cnc08                                     merge
    -noexecute
cnc18   mnz  cswex                                     set switch
        brn  cnc08                                     merge
    -nofail
cnc19   zer  cswfl                                     clear switch
        brn  cnc08                                     merge
    -nolist
cnc20   zer  cswls                                     clear switch
        brn  cnc08                                     merge
    -nooptimise
cnc21   mnz  cswno                                     set switch
        brn  cnc08                                     merge
    -noprint
cnc22   zer  cswpr                                     clear switch
        brn  cnc08                                     merge
    -optimise
cnc24   zer  cswno                                     clear switch
        brn  cnc08                                     merge
    -print
cnc25   mnz  cswpr                                     set switch
        brn  cnc08                                     merge
```

```
    cncrd (continued)
    -single
cnc27   zer   cswdb                      clear switch
        brn   cnc08                      merge
    -space
cnc28   bze   cswls,cnc09                return if -nolist
        jsr   scane                      scan integer after -space
        mov   =num01,wc                  1 space in case
        beq   xr,=t$smc,cnc29            jump if no integer
        mov   xr,-(xs)                   stack it
        jsr   gtsmi                      check integer
        ppm   cnc06                      fail if not integer
        ppm   cnc06                      fail if negative or large
        bnz   wc,cnc29                   jump if non zero
        mov   =num01,wc                  else 1 space
    merge with count of lines to skip
cnc29   add   wc,lstlc                   bump line count
        lct   wc,wc                      convert to loop counter
        blt   lstlc,lstnp,cnc30          jump if fits on page
        jsr   prtps                      eject
        jsr   listt                      list title
        brn   cnc09                      merge
    skip lines
cnc30   jsr   prtnl                      print a blank
        bct   wc,cnc30                   loop
        brn   cnc09                      merge
```

```
    cncrd (continued)
    -stitl
cnc31   mov  =r$stl,cnr$t                ptr to r$stl
        brn  cnc33                       merge
    -title
cnc32   mov  =nulls,r$stl                clear subtitle
        mov  =r$ttl,cnr$t                ptr to r$ttl
    common processing for -title, -stitl
cnc33   mov  =nulls,xr                   null in case needed
        mnz  cnttl                       set flag for next listr call
        mov  =ccofs,wb                   offset to title/subtitle
        mov  scnil,wa                    input image length
        blo  wa,wb,cnc34                 jump if no chars left
        sub  wb,wa                       no of chars to extract
        mov  r$cim,xl                    point to image
        jsr  sbstr                       get title/subtitle
    store title/subtitle
cnc34   mov  cnr$t,xl                    point to storage location
        mov  xr,(xl)                     store title/subtitle
        beq  xl,=r$stl,cnc09             return if stitl
        bnz  precl,cnc09                 return if extended listing
        bze  prich,cnc09                 return if regular printer
        mov  sclen(xr),xl                get length of title
        mov  xl,wa                       copy it
        bze  xl,cnc35                    jump if null
        add  =num10,xl                   increment
        bhi  xl,prlen,cnc09              use default lstp0 val if too long
        add  =num04,wa                   point just past title
    store offset to page nn message for short title
cnc35   mov  wa,lstpo                    store offset
        brn  cnc09                       return
    -trace
    provided for system debugging.  toggles the system label
    trace switch at compile time
cnc36   jsr  systt                       toggle switch
        brn  cnc08                       merge
```

---

if **.culc**

```
    -case
    sets value of kvcas so that names are folded or not
    during compilation.
cnc37   jsr  scane                       scan integer after -case
        zer  wc                          get 0 in case none there
        beq  xl,=t$smc,cnc38             skip if no integer
        mov  xr,-(xs)                    stack it
        jsr  gtsmi                       check integer
        ppm  cnc06                       fail if not integer
        ppm  cnc06                       fail if negative or too large
cnc38   mov  wc,kvcas                    store new case value
        brn  cnc09                       merge
```

fi

---

if **.ccmc**

```
    -compare
```

```
    sets value of kvcom so that string comparisons may
    follow collation sequence determined by the interface.
cnc39  jsr  scane                 scan integer after -compare
       zer  wc                    get 0 in case none there
       beq  xl,=t$smc,cnc40       skip if no integer
       mov  xr,-(xs)              stack it
       jsr  gtsmi                 check integer
       ppm  cnc06                 fail if not integer
       ppm  cnc06                 fail if negative or too large
cnc40  mov  wc,kvcom              store new compare value
       brn  cnc09                 merge
fi
```

```
if .cinc
    -include
cnc41  mnz  scncc                 set flag for scane
       jsr  scane                 scan quoted file name
       zer  scncc                 clear scane flag
       bne  xl,=t$con,cnc06       if not constant
       bne  (xr),=b$scl,cnc06      if not string constant
       mov  xr,r$ifn              save file name
       mov  r$inc,xl              examine include file name table
       zer  wb                    lookup by value
       jsr  tfind                 do lookup
       ppm                        never fails
       beq  xr,=inton,cnc09       ignore if already in table
       mnz  wb                    set for trim
       mov  r$ifn,xr              file name
       jsr  trimr                 remove trailing blanks
       mov  r$inc,xl              include file name table
       mnz  wb                    lookup by name this time
       jsr  tfind                 do lookup
       ppm                        never fails
       mov  =inton,teval(xl)      make table value integer 1
       icv  cnind                 increase nesting level
       mov  cnind,wa              load new nest level
       bgt  wa,=ccinm,cnc42       fail if excessive nesting
```

```
if .csfn
    record the name and line number of the current input file
       mov  r$ifa,xl              array of nested file names
       add  =vcvlb,wa             compute offset in words
       wtb  wa                    convert to bytes
       add  wa,xl                 point to element
       mov  r$sfc,(xl)            record current file name
       mov  wa,xl                 preserve nesting byte offset
       mti  rdnln                 fetch source line number as integer
       jsr  icbld                 convert to icblk
       add  r$ifl,xl              entry in nested line number array
       mov  xr,(xl)               record in array
fi
    here to switch to include file named in r$ifn
       mov  cswin,wa              max read length
       mov  r$ifn,xl              include file name
```

```
        jsr   alocs                         get buffer for complete file name
        jsr   sysif                         open include file
        ppm   cnc43                         could not open
```

---

if **.csfn**

```
    make note of the complete file name for error messages
        zer   wb                            do not trim trailing blanks
        jsr   trimr                         adjust scblk for actual length
        mov   xr,r$sfc                      save ptr to file name
        mti   cmpsn                         current statement as integer
        jsr   icbld                         build icblk for stmt number
        mov   r$sfn,xl                      file name table
        mnz   wb                            lookup statement number by name
        jsr   tfind                         allocate new teblk
        ppm                                 always possible to allocate block
        mov   r$sfc,teval(xl)               record file name as entry value
```

else

```
        mov   xr,dnamp                      release allocated scblk
```

fi

```
        zer   rdnln                         restart line counter for new file
        beq   stage,=stgic,cnc09            if initial compile
        bne   cnind,=num01,cnc09            if not first execute-time nesting
    here for -include during execute-time compile
        mov   r$cim,r$ici                   remember code argument string
        mov   scnpt,cnspt                   save position in string
        mov   scnil,cnsil                   and length of string
        brn   cnc09                         all done, merge
    here for excessive include file nesting
cnc42   erb   284,excessively nested        include files
    here if include file could not be opened
cnc43   mov   xr,dnamp                      release allocated scblk
        erb   285,include file              cannot be opened
```

fi

---

if **.csln**

```
    -line n filename
cnc44   jsr   scane                         scan integer after -line
        bne   xl,=t$con,cnc06               jump if no line number
        bne   (xr),=b$icl,cnc06             jump if not integer
        ldi   icval(xr)                     fetch integer line number
        ile   cnc06                         error if negative or zero
        beq   stage,=stgic,cnc45            skip if initial compile
        mfi   cmpln                         set directly for other compiles
        brn   cnc46                         no need to set rdnln
cnc45   sbi   intv1                         adjust number by one
        mfi   rdnln                         save line number
```

---

if **.csfn**

```
cnc46   mnz   scncc                         set flag for scane
        jsr   scane                         scan quoted file name
        zer   scncc                         clear scane flag
        beq   xl,=t$smc,cnc47               done if no file name
        bne   xl,=t$con,cnc06               error if not constant
        bne   (xr),=b$scl,cnc06             if not string constant
        jsr   newfn                         record new file name
```

```
        brn   cnc09                                  merge
    here if file name not present
cnc47   dcv   scnpt                                  set to rescan the terminator
        brn   cnc09                                  merge
else
cnc46   brn   cnc09                                  merge
fi
fi
        enp                                          end procedure cncrd
```

if **.ceng**

```
    enevs -- evaluate string expression for engine
    enevs is used by the external interface to evaluate a
    string expression, typically for an engine wishing to
    obtain the value of a variable or expression.
```

---

if **.cevb**

```
    (wb)                  0 if by value, 1 if by name
```

fi

```
    (xr)                  scblk for string to evaluate
    jsr  enevs            call to convert and evaluate
    (xr)                  pointer to result
                          = 0 if expression evaluation failed
                          = 1 if conversion to expression failed
enevs   prc  r,0                                 entry point (recursive)
```

---

if **.cevb**

```
        mov  wb,-(xs)                            save value/name flag
```

fi

```
        jsr  gtexp                               convert to expression
        ppm  enev2                               conversion fails
```

---

if **.cevb**

```
        mov  (xs)+,wb                            recover value/name flag
```

fi

```
        jsr  evalx                               evaluate expression by value
        ppm  enev1                               evaluation fails
        exi  enev1                               evaluation fails
    here if expression evaluation failed
enev1   zer  xr                                  return zero result
        exi  xr                                  return zero result
    here if conversion to expression failed
```

---

if **.cevb**

```
enev2   ica  xs                                  discard value/name flag
        mov  =num01,xr                           return integer one result
```

else

```
enev2   mov  =num01,xr                           return integer one result
```

fi

```
        exi  =num01,xr                           return integer one result
        enp  =num01,xr                           return integer one result
```

```
    engts -- get string for engine
    engts is passed an object and returns a string with
    any necessary conversions performed.
    (xr)                  input argument
    jsr  engts            call to convert to string
    (xr)                  pointer to resulting string
                          =0 if conversion not possible
engts   prc  e,0                                    entry point
        mov  xr,-(xs)                               stack argument to convert
        jsr  gtstg                                  convert to string
        ppm  engt1                                  convert impossible
        exi  engt1                                  convert impossible
    here if unable to convert to string
engt1   zer  xr                                     return zero
        exi  xr                                     return zero
        enp  xr                                     return zero
```

fi

```
    dffnc -- define function
    dffnc is called whenever a new function is assigned to
    a variable. it deals with external function use counts.
    (xr)                pointer to vrblk
    (xl)                pointer to new function block
    jsr  dffnc          call to define function
    (wa,wb)             destroyed
dffnc   prc  e,0                                   entry point
```

---

if **.cnld**

else

```
        bne  (xl),=b$efc,dffn1                     skip if new function not external
        icv  efuse(xl)                             else increment its use count
    here after dealing with new function use count
dffn1   mov  xr,wa                                 save vrblk pointer
        mov  vrfnc(xr),xr                          load old function pointer
        bne  (xr),=b$efc,dffn2                     jump if old function not external
        mov  efuse(xr),wb                          else get use count
        dcv  wb                                    decrement
        mov  wb,efuse(xr)                          store decremented value
        bnz  wb,dffn2                              jump if use count still non-zero
        jsr  sysul                                 else call system unload function
    here after dealing with old function use count
dffn2   mov  wa,xr                                 restore vrblk pointer
```

fi

```
        mov  xl,wa                                 copy function block ptr
        blt  xr,=r$yyy,dffn3                       skip checks if opsyn op definition
        bnz  vrlen(xr),dffn3                       jump if not system variable
    for system variable, check for illegal redefinition
        mov  vrsvp(xr),xl                          point to svblk
        mov  svbit(xl),wb                          load bit indicators
        anb  btfnc,wb                              is it a system function
        zrb  wb,dffn3                              redef ok if not
        erb  248,attempted redefinition            of system function
    here if redefinition is permitted
dffn3   mov  wa,vrfnc(xr)                          store new function pointer
        mov  wa,xl                                 restore function block pointer
        exi                                        return to dffnc caller
        enp                                        end procedure dffnc
```

```
    dtach -- detach i/o associated names
    detaches trblks from i/o associated variables, removes
    entry from iochn chain attached to filearg1 vrblk and may
    remove vrblk access and store traps.
    input, output, terminal are handled specially.
    (xl)                  i/o assoc. vbl name base ptr
    (wa)                  offset to name
    jsr  dtach            call for detach operation
    (xl,xr,wa,wb,wc)      destroyed
dtach   prc   e,0                          entry point
        mov   xl,dtcnb                     store name base (gbcol not called)
        add   wa,xl                        point to name location
        mov   xl,dtcnm                     store it
    loop to search for i/o trblk
dtch1   mov   xl,xr                        copy name pointer
    continue after block deletion
dtch2   mov   (xl),xl                      point to next value
        bne   (xl),=b$trt,dtch6            jump at chain end
        mov   trtyp(xl),wa                 get trap block type
        beq   wa,=trtin,dtch3              jump if input
        beq   wa,=trtou,dtch3              jump if output
        add   *trnxt,xl                    point to next link
        brn   dtch1                        loop
    delete an old association
dtch3   mov   trval(xl),(xr)               delete trblk
        mov   xl,wa                        dump xl ...
        mov   xr,wb                        ... and xr
        mov   trtrf(xl),xl                 point to trtrf trap block
        bze   xl,dtch5                     jump if no iochn
        bne   (xl),=b$trt,dtch5            jump if input, output, terminal
    loop to search iochn chain for name ptr
dtch4   mov   xl,xr                        remember link ptr
        mov   trtrf(xl),xl                 point to next link
        bze   xl,dtch5                     jump if end of chain
        mov   ionmb(xl),wc                 get name base
        add   ionmo(xl),wc                 add offset
        bne   wc,dtcnm,dtch4               loop if no match
        mov   trtrf(xl),trtrf(xr)          remove name from chain
```

```
    dtach (continued)
    prepare to resume i/o trblk scan
dtch5   mov  wa,xl                       recover xl ...
        mov  wb,xr                       ... and xr
        add  *trval,xl                   point to value field
        brn  dtch2                       continue
    exit point
dtch6   mov  dtcnb,xr                    possible vrblk ptr
        jsr  setvr                       reset vrblk if necessary
        exi                              return
        enp                              end procedure dtach
```

```
    dtype -- get datatype name
    (xr)                  object whose datatype is required
    jsr  dtype            call to get datatype
    (xr)                  result datatype
dtype   prc  e,0                                          entry point
        beq  (xr),=b$pdt,dtyp1                            jump if prog.defined
        mov  (xr),xr                                      load type word
        lei  xr                                           get entry point id (block code)
        wtb  xr                                           convert to byte offset
        mov  scnmt(xr),xr                                 load table entry
        exi                                               exit to dtype caller
    here if program defined
dtyp1   mov  pddfp(xr),xr                                 point to dfblk
        mov  dfnam(xr),xr                                 get datatype name from dfblk
        exi                                               return to dtype caller
        enp                                               end procedure dtype
```

```
    dumpr -- print dump of storage
    (xr)                 dump argument (see below)
    jsr  dumpr           call to print dump
    (xr,xl)              destroyed
    (wa,wb,wc,ra)        destroyed
    the dump argument has the following significance
    dmarg = 0            no dump printed
    dmarg = 1            partial dump (nat vars, keywords)
    dmarg = 2            full dump (arrays, tables, etc.)
    dmarg = 3            full dump + null variables
    dmarg ge 4           core dump
    since dumpr scrambles store, it is not permissible to
    collect in mid-dump. hence a collect is done initially
    and then if store runs out an error message is produced.
dumpr   prc  e,0                                  entry point
        bze  xr,dmp28                             skip dump if argument is zero
        bgt  xr,=num03,dmp29                      jump if core dump required
        zer  xl                                   clear xl
        zer  wb                                   zero move offset
        mov  xr,dmarg                             save dump argument
```

---

if **.csed**

```
        zer  dnams                                collect sediment too
```

fi

```
        jsr  gbcol                                collect garbage
        jsr  prtpg                                eject printer
        mov  =dmhdv,xr                            point to heading for variables
        jsr  prtst                                print it
        jsr  prtnl                                terminate print line
        jsr  prtnl                                and print a blank line
    first all natural variable blocks (vrblk) whose values
    are non-null are linked in lexical order using dmvch as
    the chain head and chaining through the vrget fields.
    note that this scrambles store if the process is
    interrupted before completion e.g. by exceeding time  or
    print limits. since the subsequent core dumps and
    failures if execution is resumed are very confusing, the
    execution time error routine checks for this event and
    attempts an unscramble. similar precautions should be
    observed if translate time dumping is implemented.
        zer  dmvch                                set null chain to start
        mov  hshtb,wa                             point to hash table
    loop through headers in hash table
dmp00   mov  wa,xr                                copy hash bucket pointer
        ica  wa                                   bump pointer
        sub  *vrnxt,xr                            set offset to merge
    loop through vrblks on one chain
dmp01   mov  vrnxt(xr),xr                         point to next vrblk on chain
        bze  xr,dmp09                             jump if end of this hash chain
        mov  xr,xl                                else copy vrblk pointer
```

```
    dumpr (continued)
    loop to find value and skip if null
dmp02   mov  vrval(xl),xl                   load value
        beq  dmarg,=num03,dmp2a             skip null value check if dump(3)
        beq  xl,=nulls,dmp01                loop for next vrblk if null value
dmp2a   beq  (xl),=b$trt,dmp02              loop back if value is trapped
    non-null value, prepare to search chain
        mov  xr,wc                          save vrblk pointer
        add  *vrsof,xr                      adjust ptr to be like scblk ptr
        bnz  sclen(xr),dmp03                jump if non-system variable
        mov  vrsvo(xr),xr                   else load ptr to name in svblk
    here with name pointer for new block in xr
dmp03   mov  xr,wb                          save pointer to chars
        mov  wa,dmpsv                       save hash bucket pointer
        mov  =dmvch,wa                      point to chain head
    loop to search chain for correct insertion point
dmp04   mov  wa,dmpch                       save chain pointer
        mov  wa,xl                          copy it
        mov  (xl),xr                        load pointer to next entry
        bze  xr,dmp08                       jump if end of chain to insert
        add  *vrsof,xr                      else get name ptr for chained vrblk
        bnz  sclen(xr),dmp05                jump if not system variable
        mov  vrsvo(xr),xr                   else point to name in svblk
    here prepare to compare the names
    (wa)                  scratch
    (wb)                  pointer to string of entering vrblk
    (wc)                  pointer to entering vrblk
    (xr)                  pointer to string of current block
    (xl)                  scratch
dmp05   mov  wb,xl                          point to entering vrblk string
        mov  sclen(xl),wa                   load its length
        plc  xl                             point to chars of entering string
```

---

if **.ccmc**

```
        mov  wb,dmpsb                       save wb
        mov  sclen(xr),wb                   length of old string
        plc  xr                             point to chars of old string
        jsr  syscm                          generalized lexical compare
        ppm  dmp06                          string too long, treat like eq
        ppm  dmp06                          entering string lt old string
        ppm  dmp07                          entering string gt old string
    here when entering string le old string
dmp06   mov  dmpsb,wb                       restore wb
        brn  dmp08                          found insertion point
```

```
    dumpr (continued)
    here we move out on the chain
dmp07  mov  dmpsb,wb               restore wb
       mov  dmpch,xl               copy chain pointer
else
       bhi  wa,sclen(xr),dmp06     jump if entering length high
       plc  xr                     else point to chars of old string
       cmc  dmp08,dmp07            compare, insert if new is llt old
       brn  dmp08                  or if leq (we had shorter length)
    here when new length is longer than old length
dmp06  mov  sclen(xr),wa           load shorter length
       plc  xr                     point to chars of old string
       cmc  dmp08,dmp07            compare, insert if new one low
```

```
    dumpr (continued)
    here we move out on the chain
dmp07   mov  dmpch,xl                           copy chain pointer
fi
        mov  (xl),wa                            move to next entry on chain
        brn  dmp04                              loop back
    here after locating the proper insertion point
dmp08   mov  dmpch,xl                           copy chain pointer
        mov  dmpsv,wa                           restore hash bucket pointer
        mov  wc,xr                              restore vrblk pointer
        mov  (xl),vrget(xr)                     link vrblk to rest of chain
        mov  xr,(xl)                            link vrblk into current chain loc
        brn  dmp01                              loop back for next vrblk
    here after processing all vrblks on one chain
dmp09   bne  wa,hshte,dmp00                     loop back if more buckets to go
    loop to generate dump of natural variable values
dmp10   mov  dmvch,xr                           load pointer to next entry on chain
        bze  xr,dmp11                           jump if end of chain
        mov  (xr),dmvch                         else update chain ptr to next entry
        jsr  setvr                              restore vrget field
        mov  xr,xl                              copy vrblk pointer (name base)
        mov  *vrval,wa                          set offset for vrblk name
        jsr  prtnv                              print name = value
        brn  dmp10                              loop back till all printed
    prepare to print keywords
dmp11   jsr  prtnl                              print blank line
        jsr  prtnl                              and another
        mov  =dmhdk,xr                          point to keyword heading
        jsr  prtst                              print heading
        jsr  prtnl                              end line
        jsr  prtnl                              print one blank line
        mov  =vdmkw,xl                          point to list of keyword svblk ptrs
```

```
    dumpr (continued)
    loop to dump keyword values
dmp12   mov  (xl)+,xr                 load next svblk ptr from table
        bze  xr,dmp13                 jump if end of list
```

---

if **.ccmk**

```
        beq  xr,=num01,dmp12          &compare ignored if not implemented
```

fi

```
        mov  =ch$am,wa                load ampersand
        jsr  prtch                    print ampersand
        jsr  prtst                    print keyword name
        mov  svlen(xr),wa             load name length from svblk
        ctb  wa,svchs                 get length of name
        add  wa,xr                    point to svknm field
        mov  (xr),dmpkn               store in dummy kvblk
        mov  =tmbeb,xr                point to blank-equal-blank
        jsr  prtst                    print it
        mov  xl,dmpsv                 save table pointer
        mov  =dmpkb,xl                point to dummy kvblk
        mov  =b$kvt,(xl)              build type word
        mov  =trbkv,kvvar(xl)         build ptr to dummy trace block
        mov  *kvvar,wa                set zero offset
        jsr  acess                    get keyword value
        ppm                           failure is impossible
        jsr  prtvl                    print keyword value
        jsr  prtnl                    terminate print line
        mov  dmpsv,xl                 restore table pointer
        brn  dmp12                    loop back till all printed
    here after completing partial dump
dmp13   beq  dmarg,=num01,dmp27       exit if partial dump complete
        mov  dnamb,xr                 else point to first dynamic block
    loop through blocks in dynamic storage
dmp14   beq  xr,dnamp,dmp27           jump if end of used region
        mov  (xr),wa                  else load first word of block
        beq  wa,=b$vct,dmp16          jump if vector
        beq  wa,=b$art,dmp17          jump if array
        beq  wa,=b$pdt,dmp18          jump if program defined
        beq  wa,=b$tbt,dmp19          jump if table
```

---

if **.cnbf**

else

```
        beq  wa,=b$bct,dmp30          jump if buffer
```

fi

```
    merge here to move to next block
dmp15   jsr  blkln                    get length of block
        add  wa,xr                    point past this block
        brn  dmp14                    loop back for next block
```

```
    dumpr (continued)
    here for vector
dmp16   mov  *vcvls,wb              set offset to first value
        brn  dmp19                  jump to merge
    here for array
dmp17   mov  arofs(xr),wb           set offset to arpro field
        ica  wb                     bump to get offset to values
        brn  dmp19                  jump to merge
    here for program defined
dmp18   mov  *pdfld,wb              point to values, merge
    here for table (others merge)
dmp19   bze  idval(xr),dmp15        ignore block if zero id value
        jsr  blkln                  else get block length
        mov  xr,xl                  copy block pointer
        mov  wa,dmpsv               save length
        mov  wb,wa                  copy offset to first value
        jsr  prtnl                  print blank line
        mov  wa,dmpsa               preserve offset
        jsr  prtvl                  print block value (for title)
        mov  dmpsa,wa               recover offset
        jsr  prtnl                  end print line
        beq  (xr),=b$tbt,dmp22      jump if table
        dca  wa                     point before first word
    loop to print contents of array, vector, or program def
dmp20   mov  xl,xr                  copy block pointer
        ica  wa                     bump offset
        add  wa,xr                  point to next value
        beq  wa,dmpsv,dmp14         exit if end (xr past block)
        sub  *vrval,xr              subtract offset to merge into loop
    loop to find value and ignore nulls
dmp21   mov  vrval(xr),xr           load next value
        beq  dmarg,=num03,dmp2b     skip null value check if dump(3)
        beq  xr,=nulls,dmp20        loop back if null value
dmp2b   beq  (xr),=b$trt,dmp21      loop back if trapped
        jsr  prtnv                  else print name = value
        brn  dmp20                  loop back for next field
```

```
    dumpr (continued)
    here to dump a table
dmp22   mov  *tbbuk,wc                         set offset to first bucket
        mov  *teval,wa                         set name offset for all teblks
    loop through table buckets
dmp23   mov  xl,-(xs)                          save tbblk pointer
        add  wc,xl                             point to next bucket header
        ica  wc                                bump bucket offset
        sub  *tenxt,xl                         subtract offset to merge into loop
    loop to process teblks on one chain
dmp24   mov  tenxt(xl),xl                      point to next teblk
        beq  xl,(xs),dmp26                     jump if end of chain
        mov  xl,xr                             else copy teblk pointer
    loop to find value and ignore if null
dmp25   mov  teval(xr),xr                      load next value
        beq  xr,=nulls,dmp24                   ignore if null value
        beq  (xr),=b$trt,dmp25                 loop back if trapped
        mov  wc,dmpsv                          else save offset pointer
        jsr  prtnv                             print name = value
        mov  dmpsv,wc                          reload offset
        brn  dmp24                             loop back for next teblk
    here to move to next hash chain
dmp26   mov  (xs)+,xl                          restore tbblk pointer
        bne  wc,tblen(xl),dmp23                loop back if more buckets to go
        mov  xl,xr                             else copy table pointer
        add  wc,xr                             point to following block
        brn  dmp14                             loop back to process next block
    here after completing dump
dmp27   jsr  prtpg                             eject printer
    merge here if no dump given (dmarg=0)
dmp28   exi                                    return to dump caller
    call system core dump routine
dmp29   jsr  sysdm                             call it
        brn  dmp28                             return
```

---

if **.cnbf**
else

```
    dumpr (continued)
    here to dump buffer block
dmp30  jsr  prtnl                              print blank line
       jsr  prtvl                              print value id for title
       jsr  prtnl                              force new line
       mov  =ch$dq,wa                          load double quote
       jsr  prtch                              print it
       mov  bclen(xr),wc                       load defined length
       bze  wc,dmp32                           skip characters if none
       lct  wc,wc                              load count for loop
       mov  xr,wb                              save bcblk ptr
       mov  bcbuf(xr),xr                       point to bfblk
       plc  xr                                 get set to load characters
    loop here stuffing characters in output stream
dmp31  lch  wa,(xr)+                           get next character
       jsr  prtch                              stuff it
       bct  wc,dmp31                           branch for next one
       mov  wb,xr                              restore bcblk pointer
    merge to stuff closing quote mark
dmp32  mov  =ch$dq,wa                          stuff quote
       jsr  prtch                              print it
       jsr  prtnl                              print new line
       mov  (xr),wa                            get first wd for blkln
       brn  dmp15                              merge to get next block
fi
       enp                                     end procedure dumpr
```

```
    ermsg -- print error code and error message
    kvert                  error code
    jsr  ermsg             call to print message
    (xr,xl,wa,wb,wc,ia)    destroyed
ermsg   prc   e,0                             entry point
        mov   kvert,wa                        load error code
        mov   =ermms,xr                       point to error message /error/
        jsr   prtst                           print it
        jsr   ertex                           get error message text
        add   =thsnd,wa                       bump error code for print
        mti   wa                              fail code in int acc
        mov   profs,wb                        save current buffer position
        jsr   prtin                           print code (now have error1xxx)
        mov   prbuf,xl                        point to print buffer
        psc   xl,wb                           point to the 1
        mov   =ch$bl,wa                       load a blank
        sch   wa,(xl)                         store blank over 1 (error xxx)
        csc   xl                              complete store characters
        zer   xl                              clear garbage pointer in xl
        mov   xr,wa                           keep error text
        mov   =ermns,xr                       point to / – /
        jsr   prtst                           print it
        mov   wa,xr                           get error text again
        jsr   prtst                           print error message text
        jsr   prtis                           print line
        jsr   prtis                           print blank line
        exi                                   return to ermsg caller
        enp                                   end procedure ermsg
```

```
    ertex -- get error message text
    (wa)                   error code
    jsr  ertex             call to get error text
    (xr)                   ptr to error text in dynamic
    (r$etx)                copy of ptr to error text
    (xl,wc,ia)             destroyed
ertex   prc  e,0                                        entry point
        mov  wa,ertwa                                   save wa
        mov  wb,ertwb                                   save wb
        jsr  sysem                                      get failure message text
        mov  xr,xl                                      copy pointer to it
        mov  sclen(xr),wa                               get length of string
        bze  wa,ert02                                   jump if null
        zer  wb                                         offset of zero
        jsr  sbstr                                      copy into dynamic store
        mov  xr,r$etx                                   store for relocation
    return
ert01   mov  ertwb,wb                                   restore wb
        mov  ertwa,wa                                   restore wa
        exi                                             return to caller
    return errtext contents instead of null
ert02   mov  r$etx,xr                                   get errtext
        brn  ert01                                      return
        enp  ert01                                      return
```

```
    evali -- evaluate integer argument
    evali is used by pattern primitives len,tab,rtab,pos,rpos
    when their argument is an expression value.
    (xr)                 node pointer
    (wb)                 cursor
    jsr  evali           call to evaluate integer
    ppm  loc             transfer loc for non-integer arg
    ppm  loc             transfer loc for out of range arg
    ppm  loc             transfer loc for evaluation failure
    ppm  loc             transfer loc for successful eval
    (the normal return is never taken)
    (xr)                 ptr to node with integer argument
    (wc,xl,ra)           destroyed
    on return, the node pointed to has the integer argument
    in parm1 and the proper successor pointer in pthen.
    this allows merging with the normal (integer arg) case.
evali   prc  r,4                          entry point (recursive)
        jsr  evalp                        evaluate expression
        ppm  evli1                        jump on failure
        mov  xl,-(xs)                     stack result for gtsmi
        mov  pthen(xr),xl                 load successor pointer
        mov  xr,evlio                     save original node pointer
        mov  wc,evlif                     zero if simple argument
        jsr  gtsmi                        convert arg to small integer
        ppm  evli2                        jump if not integer
        ppm  evli3                        jump if out of range
        mov  xr,evliv                     store result in special dummy node
        mov  =evlin,xr                    point to dummy node with result
        mov  =p$len,(xr)                  dummy pattern block pcode
        mov  xl,pthen(xr)                 store successor pointer
        exi  4                            take successful exit
    here if evaluation fails
evli1   exi  3                            take failure return
    here if argument is not integer
evli2   exi  1                            take non-integer error exit
    here if argument is out of range
evli3   exi  2                            take out-of-range error exit
        enp                               end procedure evali
```

```
    evalp -- evaluate expression during pattern match
    evalp is used to evaluate an expression (by value) during
    a pattern match. the effect is like evalx, but pattern
    variables are stacked and restored if necessary.
    evalp also differs from evalx in that if the result is
    an expression it is reevaluated. this occurs repeatedly.
    to support optimization of pos and rpos, evalp uses wc
    to signal the caller for the case of a simple vrblk
    that is not an expression and is not trapped.  because
    this case cannot have any side effects, optimization is
    possible.
    (xr)                  node pointer
    (wb)                  pattern match cursor
    jsr  evalp            call to evaluate expression
    ppm  loc              transfer loc if evaluation fails
    (xl)                  result
    (wa)                  first word of result block
    (wc)                  zero if simple vrblk, else non-zero
    (xr,wb)               destroyed (failure case only)
    (ra)                  destroyed
    the expression pointer is stored in parm1 of the node
    control returns to failp on failure of evaluation
evalp   prc  r,1                                  entry point (recursive)
        mov  parm1(xr),xl                         load expression pointer
        beq  (xl),=b$exl,evlp1                    jump if exblk case
    here for case of seblk
    we can give a fast return if the value of the vrblk is
    not an expression and is not trapped.
        mov  sevar(xl),xl                         load vrblk pointer
        mov  vrval(xl),xl                         load value of vrblk
        mov  (xl),wa                              load first word of value
        bhi  wa,=b$t$$,evlp3                      jump if not seblk, trblk or exblk
    here for exblk or seblk with expr value or trapped value
evlp1   chk                                       check for stack space
        mov  xr,-(xs)                             stack node pointer
        mov  wb,-(xs)                             stack cursor
        mov  r$pms,-(xs)                          stack subject string pointer
        mov  pmssl,-(xs)                          stack subject string length
        mov  pmdfl,-(xs)                          stack dot flag
        mov  pmhbs,-(xs)                          stack history stack base pointer
        mov  parm1(xr),xr                         load expression pointer
```

```
    evalp (continued)
    loop back here to reevaluate expression result
evlp2   zer   wb                            set flag for by value
        jsr   evalx                         evaluate expression
        ppm   evlp4                         jump on failure
        mov   (xr),wa                       else load first word of value
        blo   wa,=b$e$$,evlp2               loop back to reevaluate expression
    here to restore pattern values after successful eval
        mov   xr,xl                         copy result pointer
        mov   (xs)+,pmhbs                   restore history stack base pointer
        mov   (xs)+,pmdfl                   restore dot flag
        mov   (xs)+,pmssl                   restore subject string length
        mov   (xs)+,r$pms                   restore subject string pointer
        mov   (xs)+,wb                      restore cursor
        mov   (xs)+,xr                      restore node pointer
        mov   xr,wc                         non-zero for simple vrblk
        exi                                 return to evalp caller
    here to return after simple vrblk case
evlp3   zer   wc                            simple vrblk, no side effects
        exi                                 return to evalp caller
    here for failure during evaluation
evlp4   mov   (xs)+,pmhbs                   restore history stack base pointer
        mov   (xs)+,pmdfl                   restore dot flag
        mov   (xs)+,pmssl                   restore subject string length
        mov   (xs)+,r$pms                   restore subject string pointer
        add   *num02,xs                     remove node ptr, cursor
        exi   1                             take failure exit
        enp                                 end procedure evalp
```

```
    evals -- evaluate string argument
    evals is used by span, any, notany, break, breakx when
    they are passed an expression argument.
    (xr)                 node pointer
    (wb)                 cursor
    jsr  evals           call to evaluate string
    ppm  loc             transfer loc for non-string arg
    ppm  loc             transfer loc for evaluation failure
    ppm  loc             transfer loc for successful eval
    (the normal return is never taken)
    (xr)                 ptr to node with parms set
    (xl,wc,ra)           destroyed
    on return, the node pointed to has a character table
    pointer in parm1 and a bit mask in parm2. the proper
    successor is stored in pthen of this node. thus it is
    ok for merging with the normal (multi-char string) case.
evals  prc  r,3                   entry point (recursive)
       jsr  evalp                 evaluate expression
       ppm  evls1                 jump if evaluation fails
       mov  pthen(xr),-(xs)       save successor pointer
       mov  wb,-(xs)              save cursor
       mov  xl,-(xs)              stack result ptr for patst
       zer  wb                    dummy pcode for one char string
       zer  wc                    dummy pcode for expression arg
       mov  =p$brk,xl             appropriate pcode for our use
       jsr  patst                 call routine to build node
       ppm  evls2                 jump if not string
       mov  (xs)+,wb              restore cursor
       mov  (xs)+,pthen(xr)       store successor pointer
       exi  3                     take success return
    here if evaluation fails
evls1  exi  2                     take failure return
    here if argument is not string
evls2  add  *num02,xs             pop successor and cursor
       exi  1                     take non-string error exit
       enp                        end procedure evals
```

```
    evalx -- evaluate expression
    evalx is called to evaluate an expression
    (xr)                  pointer to exblk or seblk
    (wb)                  0 if by value, 1 if by name
    jsr  evalx            call to evaluate expression
    ppm  loc              transfer loc if evaluation fails
    (xr)                  result if called by value
    (xl,wa)               result name base,offset if by name
    (xr)                  destroyed (name case only)
    (xl,wa)               destroyed (value case only)
    (wb,wc,ra)            destroyed
evalx   prc   r,1                                 entry point, recursive
        beq   (xr),=b$exl,evlx2                   jump if exblk case
    here for seblk
        mov   sevar(xr),xl                        load vrblk pointer (name base)
        mov   *vrval,wa                           set name offset
        bnz   wb,evlx1                            jump if called by name
        jsr   acess                               call routine to access value
        ppm   evlx9                               jump if failure on access
    merge here to exit for seblk case
evlx1   exi                                       return to evalx caller
```

```
    evalx (continued)
    here for full expression (exblk) case
    if an error occurs in the expression code at execution
    time, control is passed via error section to exfal
    without returning to this routine.
    the following entries are made on the stack before
    giving control to the expression code
                          evalx return point
                          saved value of r$cod
                          code pointer (-r$cod)
                          saved value of flptr
                          0 if by value, 1 if by name
    flptr --------------- *exflc, fail offset in exblk
evlx2   scp  wc                         get code pointer
        mov  r$cod,wa                   load code block pointer
        sub  wa,wc                      get code pointer as offset
        mov  wa,-(xs)                   stack old code block pointer
        mov  wc,-(xs)                   stack relative code offset
        mov  flptr,-(xs)                stack old failure pointer
        mov  wb,-(xs)                   stack name/value indicator
        mov  *exflc,-(xs)               stack new fail offset
        mov  flptr,gtcef                keep in case of error
        mov  r$cod,r$gtc                keep code block pointer similarly
        mov  xs,flptr                   set new failure pointer
        mov  xr,r$cod                   set new code block pointer
        mov  kvstn,exstm(xr)            remember stmnt number
        add  *excod,xr                  point to first code word
        lcp  xr                         set code pointer
        bne  stage,=stgxt,evlx0         jump if not execution time
        mov  =stgee,stage               evaluating expression
    here to execute first code word of expression
evlx0   zer  xl                         clear garbage xl
        lcw  xr                         load first code word
        bri  (xr)                       execute it
```

```
    evalx (continued)
    come here if successful return by value (see o$rvl)
evlx3   mov  (xs)+,xr                        load value
        bze  num01(xs),evlx5                 jump if called by value
        erb  249,expression evaluated        by name returned value
    here for expression returning by name (see o$rnm)
evlx4   mov  (xs)+,wa                        load name offset
        mov  (xs)+,xl                        load name base
        bnz  num01(xs),evlx5                 jump if called by name
        jsr  acess                           else access value first
        ppm  evlx6                           jump if failure during access
    here after loading correct result into xr or xl,wa
evlx5   zer  wb                              note successful
        brn  evlx7                           merge
    here for failure in expression evaluation (see o$fex)
evlx6   mnz  wb                              note unsuccessful
    restore environment
evlx7   bne  stage,=stgee,evlx8              skip if was not previously xt
        mov  =stgxt,stage                    execute time
    merge with stage set up
evlx8   add  *num02,xs                       pop name/value indicator, *exfal
        mov  (xs)+,flptr                     restore old failure pointer
        mov  (xs)+,wc                        load code offset
        add  (xs),wc                         make code pointer absolute
        mov  (xs)+,r$cod                     restore old code block pointer
        lcp  wc                              restore old code pointer
        bze  wb,evlx1                        jump for successful return
    merge here for failure in seblk case
evlx9   exi  1                               take failure exit
        enp                                  end of procedure evalx
```

```
    exbld -- build exblk
    exbld is used to build an expression block from the
    code compiled most recently in the current ccblk.
    (xl)                 offset in ccblk to start of code
    (wb)                 integer in range 0 le n le mxlen
    jsr  exbld           call to build exblk
    (xr)                 ptr to constructed exblk
    (wa,wb,xl)           destroyed
exbld  prc  e,0                          entry point
       mov  xl,wa                        copy offset to start of code
       sub  *excod,wa                    calc reduction in offset in exblk
       mov  wa,-(xs)                     stack for later
       mov  cwcof,wa                     load final offset
       sub  xl,wa                        compute length of code
       add  *exsi$,wa                    add space for standard fields
       jsr  alloc                        allocate space for exblk
       mov  xr,-(xs)                     save pointer to exblk
       mov  =b$exl,extyp(xr)             store type word
       zer  exstm(xr)                    zeroise stmnt number field
```

if **.csln**

```
       mov  cmpln,exsln(xr)              set line number field
```

fi

```
       mov  wa,exlen(xr)                 store length
       mov  =ofex$,exflc(xr)             store failure word
       add  *exsi$,xr                    set xr for mvw
       mov  xl,cwcof                     reset offset to start of code
       add  r$ccb,xl                     point to start of code
       sub  *exsi$,wa                    length of code to move
       mov  wa,-(xs)                     stack length of code
       mvw                               move code to exblk
       mov  (xs)+,wa                     get length of code
       btw  wa                           convert byte count to word count
       lct  wa,wa                        prepare counter for loop
       mov  (xs),xl                      copy exblk ptr, dont unstack
       add  *excod,xl                    point to code itself
       mov  num01(xs),wb                 get reduction in offset
    this loop searches for negation and selection code so
    that the offsets computed whilst code was in code block
    can be transformed to reduced values applicable in an
    exblk.
exbl1  mov  (xl)+,xr                     get next code word
       beq  xr,=osla$,exbl3              jump if selection found
       beq  xr,=onta$,exbl3              jump if negation found
       bct  wa,exbl1                     loop to end of code
    no selection found or merge to exit on termination
exbl2  mov  (xs)+,xr                     pop exblk ptr into xr
       mov  (xs)+,xl                     pop reduction constant
       exi                               return to caller
```

```
    exbld (continued)
    selection or negation found
    reduce the offsets as needed. offsets occur in words
    following code words -
         =onta$, =osla$, =oslb$, =oslc$
exbl3   sub  wb,(xl)+                    adjust offset
        bct  wa,exbl4                    decrement count
exbl4   bct  wa,exbl5                    decrement count
    continue search for more offsets
exbl5   mov  (xl)+,xr                    get next code word
        beq  xr,=osla$,exbl3             jump if offset found
        beq  xr,=oslb$,exbl3             jump if offset found
        beq  xr,=oslc$,exbl3             jump if offset found
        beq  xr,=onta$,exbl3             jump if offset found
        bct  wa,exbl5                    loop
        brn  exbl2                       merge to return
        enp                              end procedure exbld
```

```
expan -- analyze expression
the expression analyzer (expan) procedure is used to scan
an expression and convert it into a tree representation.
see the description of cmblk in the structures section
for detailed format of tree blocks.
the analyzer uses a simple precedence scheme in which
operands and operators are placed on a single stack
and condensations are made when low precedence operators
are stacked after a higher precedence operator. a global
variable (in wb) keeps track of the level as follows.
0    scanning outer level of statement or expression
1    scanning outer level of normal goto
2    scanning outer level of direct goto
3    scanning inside array brackets
4    scanning inside grouping parentheses
5    scanning inside function parentheses
this variable is saved on the stack on encountering a
grouping and restored at the end of the grouping.
another global variable (in wc) counts the number of
items at one grouping level and is incremented for each
comma encountered. it is stacked with the level indicator
the scan is controlled by a three state finite machine.
a global variable stored in wa is the current state.
wa=0                  nothing scanned at this level
wa=1                  operand expected
wa=2                  operator expected
(wb)                  call type (see below)
jsr  expan            call to analyze expression
(xr)                  pointer to resulting tree
(xl,wa,wb,wc,ra)      destroyed
the entry value of wb indicates the call type as follows.
0    scanning either the main body of a statement or the
     text of an expression (from eval call). valid
     terminators are colon, semicolon. the rescan flag is
     set to return the terminator on the next scane call.
1    scanning a normal goto. the only valid
     terminator is a right paren.
2    scanning a direct goto. the only valid
     terminator is a right bracket.
```

```
    expan (continued)
    entry point
expan   prc   e,0               entry point
        zer   -(xs)             set top of stack indicator
        zer   wa                set initial state to zero
        zer   wc                zero counter value
    loop here for successive entries
exp01   jsr   scane             scan next element
        add   wa,xl             add state to syntax code
        bsw   xl,t$nes          switch on element type/state
        iff   t$va0,exp03       variable, s=0
        iff   t$va1,exp03       variable, state one
        iff   t$va2,exp04       variable, s=2
        iff   t$co0,exp03       constant, s=0
        iff   t$co1,exp03       constant, s=1
        iff   t$co2,exp04       constant, s=2
        iff   t$lp0,exp06       left paren, s=0
        iff   t$lp1,exp06       left paren, s=1
        iff   t$lp2,exp04       left paren, s=2
        iff   t$fn0,exp10       function, s=0
        iff   t$fn1,exp10       function, s=1
        iff   t$fn2,exp04       function, s=2
        iff   t$rp0,exp02       right paren, s=0
        iff   t$rp1,exp05       right paren, s=1
        iff   t$rp2,exp12       right paren, s=2
        iff   t$lb0,exp08       left brkt, s=0
        iff   t$lb1,exp08       left brkt, s=1
        iff   t$lb2,exp09       left brkt, s=2
        iff   t$rb0,exp02       right brkt, s=0
        iff   t$rb1,exp05       right brkt, s=1
        iff   t$rb2,exp18       right brkt, s=2
        iff   t$uo0,exp27       unop, s=0
        iff   t$uo1,exp27       unop, s=1
        iff   t$uo2,exp04       unop, s=2
        iff   t$bo0,exp05       binop, s=0
        iff   t$bo1,exp05       binop, s=1
        iff   t$bo2,exp26       binop, s=2
        iff   t$cm0,exp02       comma, s=0
        iff   t$cm1,exp05       comma, s=1
        iff   t$cm2,exp11       comma, s=2
        iff   t$cl0,exp02       colon, s=0
        iff   t$cl1,exp05       colon, s=1
        iff   t$cl2,exp19       colon, s=2
        iff   t$sm0,exp02       semicolon, s=0
        iff   t$sm1,exp05       semicolon, s=1
        iff   t$sm2,exp19       semicolon, s=2
        esw                     end switch on element type/state
```

```
    expan (continued)
    here for rbr,rpr,col,smc,cma in state 0
    set to rescan the terminator encountered and create
    a null constant (case of omitted null)
exp02  mnz  scnrs                            set to rescan element
       mov  =nulls,xr                        point to null, merge
    here for var or con in states 0,1
    stack the variable/constant and set state=2
exp03  mov  xr,-(xs)                         stack pointer to operand
       mov  =num02,wa                        set state 2
       brn  exp01                            jump for next element
    here for var,con,lpr,fnc,uop in state 2
    we rescan the element and create a concatenation operator
    this is the case of the blank concatenation operator.
exp04  mnz  scnrs                            set to rescan element
       mov  =opdvc,xr                        point to concat operator dv
       bze  wb,exp4a                         ok if at top level
       mov  =opdvp,xr                        else point to unmistakable concat.
    merge here when xr set up with proper concatenation dvblk
exp4a  bnz  scnbl,exp26                      merge bop if blanks, else error
    dcv  scnse          adjust start of element location
       erb  220,syntax error:                missing operator
    here for cma,rpr,rbr,col,smc,bop(s=1) bop(s=0)
    this is an erronous contruction
    dcv  scnse          adjust start of element location
exp05  erb  221,syntax error:                missing operand
    here for lpr (s=0,1)
exp06  mov  =num04,xl                        set new level indicator
       zer  xr                               set zero value for cmopn
```

```
    expan (continued)
    merge here to store old level on stack and start new one
exp07  mov  xr,-(xs)                          stack cmopn value
       mov  wc,-(xs)                          stack old counter
       mov  wb,-(xs)                          stack old level indicator
       chk                                    check for stack overflow
       zer  wa                                set new state to zero
       mov  xl,wb                             set new level indicator
       mov  =num01,wc                         initialize new counter
       brn  exp01                             jump to scan next element
    here for lbr (s=0,1)
    this is an illegal use of left bracket
exp08  erb  222,syntax error:                 invalid use of left bracket
    here for lbr (s=2)
    set new level and start to scan subscripts
exp09  mov  (xs)+,xr                          load array ptr for cmopn
       mov  =num03,xl                         set new level indicator
       brn  exp07                             jump to stack old and start new
    here for fnc (s=0,1)
    stack old level and start to scan arguments
exp10  mov  =num05,xl                         set new lev indic (xr=vrblk=cmopn)
       brn  exp07                             jump to stack old and start new
    here for cma (s=2)
    increment argument count and continue
exp11  icv  wc                                increment counter
       jsr  expdm                             dump operators at this level
       zer  -(xs)                             set new level for parameter
       zer  wa                                set new state
       bgt  wb,=num02,exp01                   loop back unless outer level
       erb  223,syntax error:                 invalid use of comma
```

```
    expan (continued)
    here for rpr (s=2)
    at outer level in a normal goto this is a terminator
    otherwise it must terminate a function or grouping
exp12   beq   wb,=num01,exp20                 end of normal goto
        beq   wb,=num05,exp13                 end of function arguments
        beq   wb,=num04,exp14                 end of grouping / selection
        erb   224,syntax error:               unbalanced right parenthesis
    here at end of function arguments
exp13   mov   =c$fnc,xl                       set cmtyp value for function
        brn   exp15                           jump to build cmblk
    here for end of grouping
exp14   beq   wc,=num01,exp17                 jump if end of grouping
        mov   =c$sel,xl                       else set cmtyp for selection
    merge here to build cmblk for level just scanned and
    to pop up to the previous scan level before continuing.
exp15   jsr   expdm                           dump operators at this level
        mov   wc,wa                           copy count
        add   =cmvls,wa                       add for standard fields at start
        wtb   wa                              convert length to bytes
        jsr   alloc                           allocate space for cmblk
        mov   =b$cmt,(xr)                     store type code for cmblk
        mov   xl,cmtyp(xr)                    store cmblk node type indicator
        mov   wa,cmlen(xr)                    store length
        add   wa,xr                           point past end of block
        lct   wc,wc                           set loop counter
    loop to move remaining words to cmblk
exp16   mov   (xs)+,-(xr)                     move one operand ptr from stack
        mov   (xs)+,wb                        pop to old level indicator
        bct   wc,exp16                        loop till all moved
```

```
    expan (continued)
    complete cmblk and stack pointer to it on stack
        sub   *cmvls,xr                    point back to start of block
        mov   (xs)+,wc                     restore old counter
        mov   (xs),cmopn(xr)               store operand ptr in cmblk
        mov   xr,(xs)                      stack cmblk pointer
        mov   =num02,wa                    set new state
        brn   exp01                        back for next element
    here at end of a parenthesized expression
exp17   jsr   expdm                        dump operators at this level
        mov   (xs)+,xr                     restore xr
        mov   (xs)+,wb                     restore outer level
        mov   (xs)+,wc                     restore outer count
        mov   xr,(xs)                      store opnd over unused cmopn val
        mov   =num02,wa                    set new state
        brn   exp01                        back for next ele8ent
    here for rbr (s=2)
    at outer level in a direct goto, this is a terminator.
    otherwise it must terminate a subscript list.
exp18   mov   =c$arr,xl                    set cmtyp for array reference
        beq   wb,=num03,exp15              jump to build cmblk if end arrayref
        beq   wb,=num02,exp20              jump if end of direct goto
        erb   225,syntax error:            unbalanced right bracket
```

```
    expan (continued)
    here for col,smc (s=2)
    error unless terminating statement body at outer level
exp19   mnz  scnrs                                    rescan terminator
        mov  wb,xl                                    copy level indicator
        bsw  xl,6                                     switch on level indicator
        iff  0,exp20                                  normal outer level
        iff  1,exp22                                  fail if normal goto
        iff  2,exp23                                  fail if direct goto
        iff  3,exp24                                  fail array brackets
        iff  4,exp21                                  fail if in grouping
        iff  5,exp21                                  fail function args
        esw                                           end switch on level
    here at normal end of expression
exp20   jsr  expdm                                    dump remaining operators
        mov  (xs)+,xr                                 load tree pointer
        ica  xs                                       pop off bottom of stack marker
        exi                                           return to expan caller
    missing right paren
exp21   erb  226,syntax error:                        missing right paren
    missing right paren in goto field
exp22   erb  227,syntax error:                        right paren missing from goto
    missing bracket in goto
exp23   erb  228,syntax error:                        right bracket missing from goto
    missing array bracket
exp24   erb  229,syntax error:                        missing right array bracket
```

```
    expan (continued)
    loop here when an operator causes an operator dump
exp25  mov  229,syntax error::
       jsr  expop                              pop one operator
       mov  expsv,xr                           restore op dv pointer and merge
    here for bop (s=2)
    remove operators (condense) from stack until no more
    left at this level or top one has lower precedence.
    loop here till this condition is met.
exp26  mov  num01(xs),xl                       load operator dvptr from stack
       ble  xl,=num05,exp27                    jump if bottom of stack level
       blt  dvrpr(xr),dvlpr(xl),exp25          else pop if new prec is lo
    here for uop (s=0,1)
    binary operator merges after precedence check
    the operator dv is stored on the stack and the scan
    continues after setting the scan state to one.
exp27  mov  xr,-(xs)                           stack operator dvptr on stack
       chk                                     check for stack overflow
       mov  =num01,wa                          set new state
       bne  xr,=opdvs,exp01                    back for next element unless =
    here for special case of binary =. the syntax allows a
    null right argument for this operator to be left
    out. accordingly we reset to state zero to get proper
    action on a terminator (supply a null constant).
       zer  wa                                 set state zero
       brn  exp01                              jump for next element
       enp                                     end procedure expan
```

```
    expap -- test for pattern match tree
    expap is passed an expression tree to determine if it
    is a pattern match. the following are recogized as
    matches in the context of this call.
    1)   an explicit use of binary question mark
    2)   a concatenation
    3)   an alternation whose left operand is a concatenation
    (xr)                 ptr to expan tree
    jsr  expap           call to test for pattern match
    ppm  loc             transfer loc if not a pattern match
    (wa)                 destroyed
    (xr)                 unchanged (if not match)
    (xr)                 ptr to binary operator blk if match
expap   prc  e,1                                  entry point
        mov  xl,-(xs)                             save xl
        bne  (xr),=b$cmt,expp2                    no match if not complex
        mov  cmtyp(xr),wa                         else load type code
        beq  wa,=c$cnc,expp1                      concatenation is a match
        beq  wa,=c$pmt,expp1                      binary question mark is a match
        bne  wa,=c$alt,expp2                      else not match unless alternation
    here for alternation. change (a b) / c to a qm (b / c)
        mov  cmlop(xr),xl                         load left operand pointer
        bne  (xl),=b$cmt,expp2                    not match if left opnd not complex
        bne  cmtyp(xl),=c$cnc,expp2               not match if left op not conc
        mov  cmrop(xl),cmlop(xr)                  xr points to (b / c)
        mov  xr,cmrop(xl)                         set xl opnds to a, (b / c)
        mov  xl,xr                                point to this altered node
    exit here for pattern match
expp1   mov  (xs)+,xl                             restore entry xl
        exi                                       give pattern match return
    exit here if not pattern match
expp2   mov  (xs)+,xl                             restore entry xl
        exi  1                                    give non-match return
        enp                                       end procedure expap
```

```
    expdm -- dump operators at current level (for expan)
    expdm uses expop to condense all operators at this syntax
    level. the stack bottom is recognized from the level
    value which is saved on the top of the stack.
    jsr  expdm            call to dump operators
    (xs)                  popped as required
    (xr,wa)               destroyed
expdm   prc  n,0                                  entry point
        mov  xl,r$exs                             save xl value
    loop to dump operators
exdm1   ble  num01(xs),=num05,exdm2               jump if stack bottom (saved level
        jsr  expop                                else pop one operator
        brn  exdm1                                and loop back
    here after popping all operators
exdm2   mov  r$exs,xl                             restore xl
        zer  r$exs                                release save location
        exi                                       return to expdm caller
        enp                                       end procedure expdm
```

```
    expop-- pop operator (for expan)
    expop is used by the expan routine to condense one
    operator from the top of the syntax stack. an appropriate
    cmblk is built for the operator (unary or binary) and a
    pointer to this cmblk is stacked.
    expop is also used by scngf (goto field scan) procedure
    jsr  expop            call to pop operator
    (xs)                  popped appropriately
    (xr,xl,wa)            destroyed
expop   prc  n,0                              entry point
        mov  num01(xs),xr                     load operator dv pointer
        beq  dvlpr(xr),=lluno,expo2           jump if unary
    here for binary operator
        mov  *cmbs$,wa                        set size of binary operator cmblk
        jsr  alloc                            allocate space for cmblk
        mov  (xs)+,cmrop(xr)                  pop and store right operand ptr
        mov  (xs)+,xl                         pop and load operator dv ptr
        mov  (xs),cmlop(xr)                   store left operand pointer
    common exit point
expo1   mov  =b$cmt,(xr)                      store type code for cmblk
        mov  dvtyp(xl),cmtyp(xr)              store cmblk node type code
        mov  xl,cmopn(xr)                     store dvptr (=ptr to dac o$xxx)
        mov  wa,cmlen(xr)                     store cmblk length
        mov  xr,(xs)                          store resulting node ptr on stack
        exi                                   return to expop caller
    here for unary operator
expo2   mov  *cmus$,wa                        set size of unary operator cmblk
        jsr  alloc                            allocate space for cmblk
        mov  (xs)+,cmrop(xr)                  pop and store operand pointer
        mov  (xs),xl                          load operator dv pointer
        brn  expo1                            merge back to exit
        enp                                   end procedure expop
```

if **.csfn**

```
    filnm -- obtain file name from statement number
    filnm takes a statement number and examines the file name
    table pointed to by r$sfn to find the name of the file
    containing the given statement.  table entries are
    arranged in order of ascending statement number (there
    is only one hash bucket in this table).  elements are
    added to the table each time there is a change in
    file name, recording the then current statement number.
    to find the file name, the linked list of teblks is
    scanned for an element containing a subscript (statement
    number) greater than the argument statement number, or
    the end of chain.  when this condition is met, the
    previous teblk contains the desired file name as its
    value entry.
    (wc)                  statement number
    jsr  filnm            call to obtain file name
    (xl)                  file name (scblk)
    (ia)                  destroyed
filnm  prc  e,0                                   entry point
       mov  wb,-(xs)                              preserve wb
       bze  wc,filn3                              return nulls if stno is zero
       mov  r$sfn,xl                              file name table
       bze  xl,filn3                              if no table
       mov  tbbuk(xl),wb                          get bucket entry
       beq  wb,r$sfn,filn3                        jump if no teblks on chain
       mov  xr,-(xs)                              preserve xr
       mov  wb,xr                                 previous block pointer
       mov  wc,-(xs)                              preserve stmt number
    loop through teblks on hash chain
filn1  mov  xr,xl                                 next element to examine
       mov  tesub(xl),xr                          load subscript value (an icblk)
       ldi  icval(xr)                             load the statement number
       mfi  wc                                    convert to address constant
       blt  (xs),wc,filn2                         compare arg with teblk stmt number
    here if desired stmt number is ge teblk stmt number
       mov  xl,wb                                 save previous entry pointer
       mov  tenxt(xl),xr                          point to next teblk on chain
       bne  xr,r$sfn,filn1                        jump if there is one
    here if chain exhausted or desired block found.
filn2  mov  wb,xl                                 previous teblk
       mov  teval(xl),xl                          get ptr to file name scblk
       mov  (xs)+,wc                              restore stmt number
       mov  (xs)+,xr                              restore xr
       mov  (xs)+,wb                              restore wb
       exi  (xs)+,wb                              restore wb
    no table or no table entries
filn3  mov  (xs)+,wb                              restore wb
       mov  =nulls,xl                             return null string
       exi  =nulls,xl                             return null string
       enp  =nulls,xl                             return null string
```

if **.culc**

```
    flstg -- fold string to upper case
    flstg folds a character string containing lower case
    characcters to one containing upper case characters.
    folding is only done if &case (kvcas) is not zero.
    (xr)                 string argument
    (wa)                 length of string
    jsr  flstg           call to fold string
    (xr)                 result string (possibly original)
    (wc)                 destroyed
flstg  prc  e,0                                  entry point
       bze  kvcas,fst99                          skip if &case is 0
       mov  xl,-(xs)                             save xl across call
       mov  xr,-(xs)                             save original scblk ptr
       jsr  alocs                                allocate new string block
       mov  (xs),xl                              point to original scblk
       mov  xr,-(xs)                             save pointer to new scblk
       plc  xl                                   point to original chars
       psc  xr                                   point to new chars
       zer  -(xs)                                init did fold flag
       lct  wc,wc                                load loop counter
fst01  lch  wa,(xl)+                             load character
       blt  wa,=ch$$a,fst02                      skip if less than lc a
       bgt  wa,=ch$$$,fst02                      skip if greater than lc z
       flc  wa                                   fold character to upper case
       mnz  (xs)                                 set did fold character flag
fst02  sch  wa,(xr)+                             store (possibly folded) character
       bct  wc,fst01                             loop thru entire string
       csc  xr                                   complete store characters
       mov  (xs)+,xr                             see if any change
       bnz  xr,fst10                             skip if folding done (no change)
       mov  (xs)+,dnamp                          do not need new scblk
       mov  (xs)+,xr                             return original scblk
       brn  fst20                                merge below
fst10  mov  (xs)+,xr                             return new scblk
       ica  xs                                   throw away original scblk pointer
fst20  mov  sclen(xr),wa                         reload string length
       mov  (xs)+,xl                             restore xl
fst99  exi                                       return
       enp                                       return
```

fi

```
gbcol -- perform garbage collection
gbcol performs a garbage collection on the dynamic region
all blocks which are no longer in use are eliminated
by moving blocks which are in use down and resetting
dnamp, the pointer to the next available location.
(wb)                  move offset (see below)
jsr  gbcol            call to collect garbage
```

---

if **.csed**

```
(xr)                  sediment size after collection
```

else

```
(xr)                  destroyed
```

fi

```
the following conditions must be met at the time when
gbcol is called.
1)   all pointers to blocks in the dynamic area must be
     accessible to the garbage collector. this means
     that they must occur in one of the following.
     a)               main stack, with current top
                      element being indicated by xs
     b)               in relocatable fields of vrblks.
     c)               in register xl at the time of call
     e)               in the special region of working
                      storage where names begin with r$.
2)   all pointers must point to the start of blocks with
     the sole exception of the contents of the code
     pointer register which points into the r$cod block.
3)   no location which appears to contain a pointer
     into the dynamic region may occur unless it is in
     fact a pointer to the start of the block. however
     pointers outside this area may occur and will
     not be changed by the garbage collector.
     it is especially important to make sure that xl
     does not contain a garbage value from some process
     carried out before the call to the collector.
gbcol has the capability of moving the final compacted
result up in memory (with addresses adjusted accordingly)
this is used to add space to the static region. the
entry value of wb is the number of bytes to move up.
the caller must guarantee that there is enough room.
furthermore the value in wb if it is non-zero, must be at
least 256 so that the mwb instruction conditions are met.
```

gbcol (continued)
the algorithm, which is a modification of the lisp-2
garbage collector devised by r.dewar and k.belcher
takes three passes as follows.

1) all pointers in memory are scanned and blocks in use determined from this scan. note that this procedure is recursive and uses the main stack for linkage. the marking process is thus similar to that used in a standard lisp collector. however the method of actually marking the blocks is different.
   the first field of a block normally contains a code entry point pointer. such an entry pointer can be distinguished from the address of any pointer to be processed by the collector. during garbage collection, this word is used to build a back chain of pointers through fields which point to the block. the end of the chain is marked by the occurence of the word which used to be in the first word of the block. this backchain serves both as a mark indicating that the block is in use and as a list of references for the relocation phase.
2) storage is scanned sequentially to discover which blocks are currently in use as indicated by the presence of a backchain. two pointers are maintained one scans through looking at each block. the other is incremented only for blocks found to be in use. in this way, the eventual location of each block can be determined without actually moving any blocks. as each block which is in use is processed, the back chain is used to reset all pointers which point to this block to contain its new address, i.e. the address it will occupy after the blocks are moved. the first word of the block, taken from the end of the chain is restored at this point.
   during pass 2, the collector builds blocks which describe the regions of storage which are to be moved in the third pass. there is one descriptor for each contiguous set of good blocks. the descriptor is built just behind the block to be moved and contains a pointer to the next block and the number of words to be moved.
3) in the third and final pass, the move descriptor blocks built in pass two are used to actually move the blocks down to the bottom of the dynamic region. the collection is then complete and the next available location pointer is reset.

---

if **.csed**

```
the garbage collector also recognizes the concept of
sediment.  sediment is defined as long-lived objects
which percipitate to the bottom of dynamic storage.
moving these objects during repeated collections is
inefficient.  it also contributes to thrashing on
systems with virtual memory.  in a typical worst-case
situation, there may be several megabytes of live objects
in the sediment, and only a few dead objects in need of
collection.  without recognising sediment, the standard
collector would move those megabytes of objects downward
to squeeze out the dead objects.  this type of move
would result in excessive thrasing for very little memory
gain.
scanning of blocks in the sediment cannot be avoided
entirely, because these blocks may contain pointers to
live objects above the sediment.  however, sediment
blocks need not be linked to a back chain as described
in pass one above.  since these blocks will not be moved,
pointers to them do not need to be adjusted.  eliminating
unnecessary back chain links increases locality of
reference, improving virtual memory performance.
because back chains are used to mark blocks whose con-
tents have been processed, a different marking system
```

---

if **.cepp**

```
is needed for blocks in the sediment.  since block type
words point to odd-parity entry addresses, merely incre-
menting the type word serves to mark the block as pro-
cessed.  during pass three, the type words are decre-
mented to restore them to their original value.
```

else

```
is needed for blocks in the sediment.  all block type
words normally lie in the range b$aaa to p$yyy.  blocks
can be marked by adding an offset (created in gbcmk) to
move type words out of this range.  during pass three the
offset is subtracted to restore them to their original
value.
```

fi

gbcol (continued)
the variable dnams contains the number of bytes of memory currently in the sediment. setting dnams to zero will eliminate the sediment and force it to be included in a full garbage collection. gbcol returns a suggested new value for dnams (usually dnamp-dnamb) in xr which the caller can store in dnams if it wishes to maintain the sediment. that is, data remaining after a garbage collection is considered to be sediment. if one accepts the common lore that most objects are either very short- or very long-lived, then this naive setting of dnams probably includes some short-lived objects toward the end of the sediment.
knowing when to reset dnams to zero to collect the sediment is not precisely known. we force it to zero prior to producing a dump, when gbcol is invoked by collect() (so that the sediment is invisible to the user), when sysmm is unable to obtain additional memory, and when gbcol is called to relocate the dynamic area up in memory (to make room for enlarging the static area). if there are no other reset situations, this leads to the inexorable growth of the sediment, possible forcing a modest program to begin to use virtual memory that it otherwise would not.
as we scan sediment blocks in pass three, we maintain aggregate counts of the amount of dead and live storage, which is used to decide when to reset dnams. when the ratio of free storage found in the sediment to total sediment size exceeds a threshold, the sediment is marked for collection on the next gbcol call.

fi

```
    gbcol (continued)
gbcol   prc  e,0                      entry point
        bnz  dmvch,gbc14              fail if in mid-dump
        mnz  gbcfl                    note gbcol entered
        mov  wa,gbsva                 save entry wa
        mov  wb,gbsvb                 save entry wb
        mov  wc,gbsvc                 save entry wc
        mov  xl,-(xs)                 save entry xl
        scp  wa                       get code pointer value
        sub  r$cod,wa                 make relative
        lcp  wa                       and restore
```

```
if .csed
        bze  wb,gbc0a                 check there is no move offset
        zer  dnams                    collect sediment if must move it
gbc0a   mov  dnamb,wa                 start of dynamic area
        add  dnams,wa                 size of sediment
        mov  wa,gbcsd                 first location past sediment
```

```
if .cepp
else
        mov  =p$yyy,wa                last entry point
        icv  wa                       address past last entry point
        sub  =b$aaa,wa                size of entry point area
        mov  wa,gbcmk                 use to mark processed sed. blocks
fi
fi
```

```
if .cgbc
    inform sysgc that collection to commence
        mnz  xr                       non-zero flags start of collection
        mov  dnamb,wa                 start of dynamic area
        mov  dnamp,wb                 next available location
        mov  dname,wc                 last available location + 1
        jsr  sysgc                    inform of collection
fi
    process stack entries
        mov  xs,xr                    point to stack front
        mov  stbas,xl                 point past end of stack
        bge  xl,xr,gbc00              ok if d-stack
        mov  xl,xr                    reverse if ...
        mov  xs,xl                    ... u-stack
    process the stack
gbc00   jsr  gbcpf                    process pointers on stack
    process special work locations
        mov  =r$aaa,xr                point to start of relocatable locs
        mov  =r$yyy,xl                point past end of relocatable locs
        jsr  gbcpf                    process work fields
    prepare to process variable blocks
        mov  hshtb,wa                 point to first hash slot pointer
    loop through hash slots
gbc01   mov  wa,xl                    point to next slot
        ica  wa                       bump bucket pointer
        mov  wa,gbcnm                 save bucket pointer
```

```
    gbcol (continued)
    loop through variables on one hash chain
gbc02   mov  (xl),xr              load ptr to next vrblk
        bze  xr,gbc03             jump if end of chain
        mov  xr,xl                else copy vrblk pointer
        add  *vrval,xr            point to first reloc fld
        add  *vrnxt,xl            point past last (and to link ptr)
        jsr  gbcpf                process reloc fields in vrblk
        brn  gbc02                loop back for next block
    here at end of one hash chain
gbc03   mov  gbcnm,wa             restore bucket pointer
        bne  wa,hshte,gbc01       loop back if more buckets to go
```

```
gbcol (continued)
now we are ready to start pass two. registers are used
as follows in pass two.
(xr)                  scans through all blocks
(wc)                  pointer to eventual location
the move description blocks built in this pass have
the following format.
word 1                pointer to next move block,
                      zero if end of chain of blocks
word 2                length of blocks to be moved in
                      bytes. set to the address of the
                      first byte while actually scanning
                      the blocks.
the first entry on this chain is a special entry
consisting of the two words gbcnm and gbcns. after
building the chain of move descriptors, gbcnm points to
the first real move block, and gbcns is the length of
blocks in use at the start of storage which need not
be moved since they are in the correct position.
```

---

if **.csed**

	mov	`dnamb,xr`	point to first block
	zer	`wb`	accumulate size of dead blocks
`gbc04`	**beq**	`xr,gbcsd,gbc4c`	jump if end of sediment
	mov	`(xr),wa`	else get first word

---

if **.cepp**

	bod	`wa,gbc4b`	jump if entry pointer (unused)
	dcv	`wa`	restore entry pointer

else

	bhi	`wa,=p$yyy,gbc4a`	skip if not entry ptr (in use)
	bhi	`wa,=b$aaa,gbc4b`	jump if entry pointer (unused)
`gbc4a`	**sub**	`gbcmk,wa`	restore entry pointer

fi

	mov	`wa,(xr)`	restore first word
	jsr	`blkln`	get length of this block
	add	`wa,xr`	bump actual pointer
	brn	`gbc04`	continue scan through sediment

```
here for unused sediment block
```

`gbc4b`	**jsr**	`blkln`	get length of this block
	add	`wa,xr`	bump actual pointer
	add	`wa,wb`	count size of unused blocks
	brn	`gbc04`	continue scan through sediment

```
here at end of sediment.  remember size of free blocks
within the sediment.  this will be used later to decide
how to set the sediment size returned to caller.
then scan rest of dynamic area above sediment.
(wb) = aggregate size of free blocks in sediment
(xr) = first location past sediment
```

`gbc4c`	**mov**	`wb,gbcsf`	size of sediment free space

else

	mov	`dnamb,xr`	point to first block

fi

	mov	`xr,wc`	set as first eventual location

```
        add   gbsvb,wc              add offset for eventual move up
        zer   gbcnm                 clear initial forward pointer
        mov   =gbcnm,gbclm          initialize ptr to last move block
        mov   xr,gbcns              initialize first address
   loop through a series of blocks in use
gbc05   beq   xr,dnamp,gbc07        jump if end of used region
        mov   (xr),wa               else get first word
```

---

if **.cepp**

```
        bod   wa,gbc07              jump if entry pointer (unused)
```

else

```
        bhi   wa,=p$yyy,gbc06       skip if not entry ptr (in use)
        bhi   wa,=b$aaa,gbc07       jump if entry pointer (unused)
```

fi

```
   here for block in use, loop to relocate references
gbc06   mov   wa,xl                 copy pointer
        mov   (xl),wa               load forward pointer
        mov   wc,(xl)               relocate reference
```

---

if **.cepp**

```
        bev   wa,gbc06              loop back if not end of chain
```

else

```
        bhi   wa,=p$yyy,gbc06       loop back if not end of chain
        blo   wa,=b$aaa,gbc06       loop back if not end of chain
```

fi

```
    gbcol (continued)
    at end of chain, restore first word and bump past
        mov  wa,(xr)                        restore first word
        jsr  blkln                          get length of this block
        add  wa,xr                          bump actual pointer
        add  wa,wc                          bump eventual pointer
        brn  gbc05                          loop back for next block
    here at end of a series of blocks in use
gbc07   mov  xr,wa                          copy pointer past last block
        mov  gbclm,xl                       point to previous move block
        sub  num01(xl),wa                   subtract starting address
        mov  wa,num01(xl)                   store length of block to be moved
    loop through a series of blocks not in use
gbc08   beq  xr,dnamp,gbc10                 jump if end of used region
        mov  (xr),wa                        else load first word of next block
```

---

if **.cepp**

```
        bev  wa,gbc09                       jump if in use
```

else

```
        bhi  wa,=p$yyy,gbc09                jump if in use
        blo  wa,=b$aaa,gbc09                jump if in use
```

fi

```
        jsr  blkln                          else get length of next block
        add  wa,xr                          push pointer
        brn  gbc08                          and loop back
    here for a block in use after processing a series of
    blocks which were not in use, build new move block.
gbc09   sub  *num02,xr                      point 2 words behind for move block
        mov  gbclm,xl                       point to previous move block
        mov  xr,(xl)                        set forward ptr in previous block
        zer  (xr)                           zero forward ptr of new block
        mov  xr,gbclm                       remember address of this block
        mov  xr,xl                          copy ptr to move block
        add  *num02,xr                      point back to block in use
        mov  xr,num01(xl)                   store starting address
        brn  gbc06                          jump to process block in use
```

```
    gbcol (continued)
    here for pass three -- actually move the blocks down
    (xl)                 pointer to old location
    (xr)                 pointer to new location
```

if **.csed**

```
gbc10  mov  gbcsd,xr        point to storage above sediment
```

else

```
gbc10  mov  dnamb,xr        point to start of storage
```

fi

```
       add  gbcns,xr        bump past unmoved blocks at start
    loop through move descriptors
gbc11  mov  gbcnm,xl        point to next move block
       bze  xl,gbc12        jump if end of chain
       mov  (xl)+,gbcnm     move pointer down chain
       mov  (xl)+,wa        get length to move
       mvw                  perform move
       brn  gbc11           loop back
    now test for move up
gbc12  mov  xr,dnamp        set next available loc ptr
       mov  gbsvb,wb        reload move offset
       bze  wb,gbc13        jump if no move required
       mov  xr,xl           else copy old top of core
       add  wb,xr           point to new top of core
       mov  xr,dnamp        save new top of core pointer
       mov  xl,wa           copy old top
       sub  dnamb,wa        minus old bottom = length
       add  wb,dnamb        bump bottom to get new value
       mwb                  perform move (backwards)
    merge here to exit
gbc13  zer  xr              clear garbage value in xr
       mov  xr,gbcfl        note exit from gbcol
```

if **.cgbc**

```
       mov  dnamb,wa        start of dynamic area
       mov  dnamp,wb        next available location
       mov  dname,wc        last available location + 1
       jsr  sysgc           inform sysgc of completion
```

fi

if **.csed**

```
    decide whether to mark sediment for collection next time.
    this is done by examining the ratio of previous sediment
    free space to the new sediment size.
       sti  gbcia           save ia
       zer  xr              presume no sediment will remain
       mov  gbcsf,wb        free space in sediment
       btw  wb              convert bytes to words
       mti  wb              put sediment free store in ia
       mli  gbsed           multiply by sediment factor
       iov  gb13a           jump if overflowed
       mov  dnamp,wb        end of dynamic area in use
       sub  dnamb,wb        minus start is sediment remaining
       btw  wb              convert to words
       mov  wb,gbcsf        store it
```

```
        sbi   gbcsf                      subtract from scaled up free store
        igt   gb13a                      jump if large free store in sedimnt
        mov   dnamp,xr                   below threshold, return sediment
        sub   dnamb,xr                   for use by caller
gb13a   ldi   gbcia                      restore ia
fi
        mov   gbsva,wa                   restore wa
        mov   gbsvb,wb                   restore wb
        scp   wc                         get code pointer
        add   r$cod,wc                   make absolute again
        lcp   wc                         and replace absolute value
        mov   gbsvc,wc                   restore wc
        mov   (xs)+,xl                   restore entry xl
        icv   gbcnt                      increment count of collections
        exi                              exit to gbcol caller
    garbage collection not allowed whilst dumping
gbc14   icv   errft                      fatal error
        erb   250,insufficient           memory to complete dump
        enp                              end procedure gbcol
```

```
    gbcpf -- process fields for garbage collector
    this procedure is used by the garbage collector to
    process fields in pass one. see gbcol for full details.
    (xr)                 ptr to first location to process
    (xl)                 ptr past last location to process
    jsr  gbcpf           call to process fields
    (xr,wa,wb,wc,ia)     destroyed
    note that although this procedure uses a recursive
    approach, it controls its own stack and is not recursive.
gbcpf   prc  e,0                                   entry point
        zer  -(xs)                                 set zero to mark bottom of stack
        mov  xl,-(xs)                              save end pointer
    merge here to go down a level and start a new loop
    1(xs)                next lvl field ptr (0 at outer lvl)
    0(xs)                ptr past last field to process
    (xr)                 ptr to first field to process
    loop to process successive fields
gpf01   mov  (xr),xl                               load field contents
        mov  xr,wc                                 save field pointer
```

```
if .crpp
        bod  xl,gpf2a                              jump if not ptr into dynamic area
fi
        blt  xl,dnamb,gpf2a                        jump if not ptr into dynamic area
        bge  xl,dnamp,gpf2a                        jump if not ptr into dynamic area
    here we have a ptr to a block in the dynamic area.
    link this field onto the reference backchain.
        mov  (xl),wa                               load ptr to chain (or entry ptr)
```

```
if .csed
        blt  xl,gbcsd,gpf1a                        do not chain if within sediment
fi
        mov  xr,(xl)                               set this field as new head of chain
        mov  wa,(xr)                               set forward pointer
    now see if this block has been processed before
```

```
if .cepp
gpf1a   bod  wa,gpf03                              jump if not already processed
else
gpf1a   bhi  wa,=p$yyy,gpf2a                       jump if already processed
        bhi  wa,=b$aaa,gpf03                       jump if not already processed
fi
    here to restore pointer in xr to field just processed
gpf02   mov  wc,xr                                 restore field pointer
    here to move to next field
gpf2a   ica  xr                                    bump to next field
        bne  xr,(xs),gpf01                         loop back if more to go
```

```
    gbcpf (continued)
    here we pop up a level after finishing a block
        mov  (xs)+,xl                      restore pointer past end
        mov  (xs)+,xr                      restore block pointer
        bnz  xr,gpf2a                      continue loop unless outer levl
        exi                                return to caller if outer level
    here to process an active block which has not been done
```
---
if **.csed**
```
    since sediment blocks are not marked by putting them on
    the back chain, they must be explicitly marked in another
    manner.  if odd parity entry points are present, mark by
    temporarily converting to even parity.  if odd parity not
    available, the entry point is adjusted by the value in
    gbcmk.
gpf03   bge  xl,gbcsd,gpf3a                if not within sediment
```
---
if **.cepp**
```
        icv  (xl)                          mark by making entry point even
```
else
```
        add  gbcmk,(xl)                    mark by biasing entry point
```
fi
```
gpf3a   mov  xl,xr                         copy block pointer
```
else
```
gpf03   mov  xl,xr                         copy block pointer
```
fi
```
        mov  wa,xl                         copy first word of block
        lei  xl                            load entry point id (bl$xx)
    block type switch. note that blocks with no relocatable
    fields just return to gpf02 here to continue to next fld.
        bsw  xl,bl$$$                      switch on block type
        iff  bl$ar,gpf06                   arblk
```
---
if **.cnbf**
```
        iff  bl$bc,gpf02                   bcblk - dummy to fill out iffs
```
else
```
        iff  bl$bc,gpf18                   bcblk
```
fi
```
        iff  bl$bf,gpf02                   bfblk
        iff  bl$cc,gpf07                   ccblk
```
---
if **.csln**
```
        iff  bl$cd,gpf19                   cdblk
```
else
```
        iff  bl$cd,gpf08                   cdblk
```
fi
```
        iff  bl$cm,gpf04                   cmblk
        iff  bl$df,gpf02                   dfblk
        iff  bl$ev,gpf10                   evblk
        iff  bl$ex,gpf17                   exblk
        iff  bl$ff,gpf11                   ffblk
        iff  bl$nm,gpf10                   nmblk
        iff  bl$p0,gpf10                   p0blk
        iff  bl$p1,gpf12                   p1blk
        iff  bl$p2,gpf12                   p2blk
```

iff	`bl$pd,gpf13`	pdblk
iff	`bl$pf,gpf14`	pfblk
iff	`bl$tb,gpf08`	tbblk
iff	`bl$te,gpf15`	teblk
iff	`bl$tr,gpf16`	trblk
iff	`bl$vc,gpf08`	vcblk
iff	`bl$xr,gpf09`	xrblk
iff	`bl$ct,gpf02`	ctblk
iff	`bl$ef,gpf02`	efblk
iff	`bl$ic,gpf02`	icblk
iff	`bl$kv,gpf02`	kvblk
iff	`bl$rc,gpf02`	rcblk
iff	`bl$sc,gpf02`	scblk
iff	`bl$se,gpf02`	seblk
iff	`bl$xn,gpf02`	xnblk
esw		end of jump table

```
    gbcpf (continued)
    cmblk
gpf04  mov  cmlen(xr),wa                          load length
       mov  *cmtyp,wb                             set offset
    here to push down to new level
    (wc)                 field ptr at previous level
    (xr)                 ptr to new block
    (wa)                 length (reloc flds + flds at start)
    (wb)                 offset to first reloc field
gpf05  add  xr,wa                                 point past last reloc field
       add  wb,xr                                 point to first reloc field
       mov  wc,-(xs)                              stack old field pointer
       mov  wa,-(xs)                              stack new limit pointer
       chk                                        check for stack overflow
       brn  gpf01                                 if ok, back to process
    arblk
gpf06  mov  arlen(xr),wa                          load length
       mov  arofs(xr),wb                          set offset to 1st reloc fld (arpro)
       brn  gpf05                                 all set
    ccblk
gpf07  mov  ccuse(xr),wa                          set length in use
       mov  *ccuse,wb                             1st word (make sure at least one)
       brn  gpf05                                 all set
```

if **.csln**

```
    cdblk
gpf19   mov   cdlen(xr),wa            load length
        mov   *cdfal,wb               set offset
        brn   gpf05                   jump back
    tbblk, vcblk
```

else

```
    cdblk, tbblk, vcblk
```

fi

```
gpf08   mov   offs2(xr),wa            load length
        mov   *offs3,wb               set offset
        brn   gpf05                   jump back
    xrblk
gpf09   mov   xrlen(xr),wa            load length
        mov   *xrptr,wb               set offset
        brn   gpf05                   jump back
    evblk, nmblk, p0blk
gpf10   mov   *offs2,wa               point past second field
        mov   *offs1,wb               offset is one (only reloc fld is 2)
        brn   gpf05                   all set
    ffblk
gpf11   mov   *ffofs,wa               set length
        mov   *ffnxt,wb               set offset
        brn   gpf05                   all set
    p1blk, p2blk
gpf12   mov   *parm2,wa               length (parm2 is non-relocatable)
        mov   *pthen,wb               set offset
        brn   gpf05                   all set
```

```
    gbcpf (continued)
    pdblk
gpf13  mov  pddfp(xr),xl              load ptr to dfblk
       mov  dfpdl(xl),wa              get pdblk length
       mov  *pdfld,wb                 set offset
       brn  gpf05                     all set
    pfblk
gpf14  mov  *pfarg,wa                 length past last reloc
       mov  *pfcod,wb                 offset to first reloc
       brn  gpf05                     all set
    teblk
gpf15  mov  *tesi$,wa                 set length
       mov  *tesub,wb                 and offset
       brn  gpf05                     all set
    trblk
gpf16  mov  *trsi$,wa                 set length
       mov  *trval,wb                 and offset
       brn  gpf05                     all set
    exblk
gpf17  mov  exlen(xr),wa              load length
       mov  *exflc,wb                 set offset
       brn  gpf05                     jump back
```

---

if **.cnbf**
else

```
    bcblk
gpf18  mov  *bcsi$,wa                 set length
       mov  *bcbuf,wb                 and offset
       brn  gpf05                     all set
```

fi

```
       enp                            end procedure gbcpf
```

```
    gtarr -- get array
    gtarr is passed an object and returns an array if possibl
    (xr)                 value to be converted
    (wa)                 0 to place table addresses in array
                         non-zero for keys/values in array
    jsr  gtarr           call to get array
    ppm  loc             transfer loc for all null table
    ppm  loc             transfer loc if convert impossible
    (xr)                 resulting array
    (xl,wa,wb,wc)        destroyed
gtarr   prc   e,2                                   entry point
        mov   wa,gtawa                              save wa indicator
        mov   (xr),wa                               load type word
        beq   wa,=b$art,gtar8                       exit if already an array
        beq   wa,=b$vct,gtar8                       exit if already an array
        bne   wa,=b$tbt,gta9a                       else fail if not a table (sgd02)
    here we convert a table to an array
        mov   xr,-(xs)                              replace tbblk pointer on stack
        zer   xr                                    signal first pass
        zer   wb                                    zero non-null element count
    the following code is executed twice. on the first pass,
    signalled by xr=0, the number of non-null elements in
    the table is counted in wb. in the second pass, where
    xr is a pointer into the arblk, the name and value are
    entered into the current arblk location provided gtawa
    is non-zero.  if gtawa is zero, the address of the teblk
    is entered into the arblk twice (c3.762).
gtar1   mov   (xs),xl                               point to table
        add   tblen(xl),xl                          point past last bucket
        sub   *tbbuk,xl                             set first bucket offset
        mov   xl,wa                                 copy adjusted pointer
    loop through buckets in table block
    next three lines of code rely on tenxt having a value
    1 less than tbbuk.
gtar2   mov   wa,xl                                 copy bucket pointer
        dca   wa                                    decrement bucket pointer
    loop through teblks on one bucket chain
gtar3   mov   tenxt(xl),xl                          point to next teblk
        beq   xl,(xs),gtar6                         jump if chain end (tbblk ptr)
        mov   xl,cnvtp                              else save teblk pointer
    loop to find value down trblk chain
gtar4   mov   teval(xl),xl                          load value
        beq   (xl),=b$trt,gtar4                     loop till value found
        mov   xl,wc                                 copy value
        mov   cnvtp,xl                              restore teblk pointer
```

```
    gtarr (continued)
    now check for null and test cases
        beq  wc,=nulls,gtar3                     loop back to ignore null value
        bnz  xr,gtar5                            jump if second pass
        icv  wb                                  for the first pass, bump count
        brn  gtar3                               and loop back for next teblk
    here in second pass
gtar5   bze  gtawa,gta5a                         jump if address wanted
        mov  tesub(xl),(xr)+                     store subscript name
        mov  wc,(xr)+                            store value in arblk
        brn  gtar3                               loop back for next teblk
    here to record teblk address in arblk.  this allows
    a sort routine to sort by ascending address.
gta5a   mov  xl,(xr)+                            store teblk address in name
        mov  xl,(xr)+                            and value slots
        brn  gtar3                               loop back for next teblk
    here after scanning teblks on one chain
gtar6   bne  wa,(xs),gtar2                       loop back if more buckets to go
        bnz  xr,gtar7                            else jump if second pass
    here after counting non-null elements
        bze  wb,gtar9                            fail if no non-null elements
        mov  wb,wa                               else copy count
        add  wb,wa                               double (two words/element)
        add  =arvl2,wa                           add space for standard fields
        wtb  wa                                  convert length to bytes
        bgt  wa,mxlen,gta9b                      error if too long for array
        jsr  alloc                               else allocate space for arblk
        mov  =b$art,(xr)                         store type word
        zer  idval(xr)                           zero id for the moment
        mov  wa,arlen(xr)                        store length
        mov  =num02,arndm(xr)                    set dimensions = 2
        ldi  intv1                               get integer one
        sti  arlbd(xr)                           store as lbd 1
        sti  arlb2(xr)                           store as lbd 2
        ldi  intv2                               load integer two
        sti  ardm2(xr)                           store as dim 2
        mti  wb                                  get element count as integer
        sti  ardim(xr)                           store as dim 1
        zer  arpr2(xr)                           zero prototype field for now
        mov  *arpr2,arofs(xr)                    set offset field (signal pass 2)
        mov  xr,wb                               save arblk pointer
        add  *arvl2,xr                           point to first element location
        brn  gtar1                               jump back to fill in elements
```

```
    gtarr (continued)
    here after filling in element values
gtar7   mov  wb,xr                              restore arblk pointer
        mov  wb,(xs)                            store as result
    now we need the array prototype which is of the form nn,2
    this is obtained by building the string for nn02 and
    changing the zero to a comma before storing it.
        ldi  ardim(xr)                          get number of elements (nn)
        mli  intvh                              multiply by 100
        adi  intv2                              add 2 (nn02)
        jsr  icbld                              build integer
        mov  xr,-(xs)                           store ptr for gtstg
        jsr  gtstg                              convert to string
        ppm                                     convert fail is impossible
        mov  xr,xl                              copy string pointer
        mov  (xs)+,xr                           reload arblk pointer
        mov  xl,arpr2(xr)                       store prototype ptr (nn02)
        sub  =num02,wa                          adjust length to point to zero
        psc  xl,wa                              point to zero
        mov  =ch$cm,wb                          load a comma
        sch  wb,(xl)                            store a comma over the zero
        csc  xl                                 complete store characters
    normal return
gtar8   exi                                     return to caller
    null table non-conversion return
gtar9   mov  (xs)+,xr                           restore stack for conv err (sgd02)
        exi  1                                  return
    impossible conversion return
gta9a   exi  2                                  return
    array size too large
gta9b   erb  260,conversion array               size exceeds maximum permitted
        enp                                     procedure gtarr
```

```
    gtcod -- convert to code
    (xr)                  object to be converted
    jsr  gtcod            call to convert to code
    ppm  loc              transfer loc if convert impossible
    (xr)                  pointer to resulting cdblk
    (xl,wa,wb,wc,ra)      destroyed
    if a spitbol error occurs during compilation or pre-
    evaluation, control is passed via error section to exfal
    without returning to this routine.
gtcod   prc   e,1                            entry point
        beq   (xr),=b$cds,gtcd1              jump if already code
        beq   (xr),=b$cdc,gtcd1              jump if already code
    here we must generate a cdblk by compilation
        mov   xr,-(xs)                       stack argument for gtstg
        jsr   gtstg                          convert argument to string
        ppm   gtcd2                          jump if non-convertible
        mov   flptr,gtcef                    save fail ptr in case of error
        mov   r$cod,r$gtc                    also save code ptr
        mov   xr,r$cim                       else set image pointer
        mov   wa,scnil                       set image length
        zer   scnpt                          set scan pointer
        mov   =stgxc,stage                   set stage for execute compile
        mov   cmpsn,lstsn                    in case listr called
```

---

if **.csln**

```
        icv   cmpln                          bump line number
```

fi

```
        jsr   cmpil                          compile string
        mov   =stgxt,stage                   reset stage for execute time
        zer   r$cim                          clear image
    merge here if no convert required
gtcd1   exi                                  give normal gtcod return
    here if unconvertible
gtcd2   exi   1                              give error return
        enp                                  end procedure gtcod
```

```
    gtexp -- convert to expression
```

---

if **.cevb**

```
    (wb)                  0 if by value, 1 if by name
```

fi

```
    (xr)                  input value to be converted
    jsr  gtexp            call to convert to expression
    ppm  loc              transfer loc if convert impossible
    (xr)                  pointer to result exblk or seblk
    (xl,wa,wb,wc,ra)      destroyed
    if a spitbol error occurs during compilation or pre-
    evaluation, control is passed via error section to exfal
    without returning to this routine.
gtexp   prc  e,1                          entry point
        blo  (xr),=b$e$$,gtex1            jump if already an expression
        mov  xr,-(xs)                     store argument for gtstg
        jsr  gtstg                        convert argument to string
        ppm  gtex2                        jump if unconvertible
    check the last character of the string for colon or
    semicolon.  these characters can legitimately end an
    expression in open code, so expan will not detect them
    as errors, but they are invalid as terminators for a
    string that is being converted to expression form.
        mov  xr,xl                        copy input string pointer
        plc  xl,wa                        point one past the string end
        lch  xl,-(xl)                     fetch the last character
        beq  xl,=ch$cl,gtex2              error if it is a semicolon
        beq  xl,=ch$sm,gtex2              or if it is a colon
    here we convert a string by compilation
        mov  xr,r$cim                     set input image pointer
        zer  scnpt                        set scan pointer
        mov  wa,scnil                     set input image length
```

---

if **.cevb**

```
        mov  wb,-(xs)                     save value/name flag
```

fi

```
        zer  wb                           set code for normal scan
        mov  flptr,gtcef                  save fail ptr in case of error
        mov  r$cod,r$gtc                  also save code ptr
        mov  =stgev,stage                 adjust stage for compile
        mov  =t$uok,scntp                 indicate unary operator acceptable
        jsr  expan                        build tree for expression
        zer  scnrs                        reset rescan flag
```

---

if **.cevb**

```
        mov  (xs)+,wa                     restore value/name flag
```

fi

```
        bne  scnpt,scnil,gtex2            error if not end of image
        zer  wb                           set ok value for cdgex call
        mov  xr,xl                        copy tree pointer
        jsr  cdgex                        build expression block
        zer  r$cim                        clear pointer
        mov  =stgxt,stage                 restore stage for execute time
    merge here if no conversion required
gtex1   exi                               return to gtexp caller
```

```
    here if unconvertible
gtex2   exi   1                                  take error exit
        enp                                      end procedure gtexp
```

```
    gtint -- get integer value
    gtint is passed an object and returns an integer after
    performing any necessary conversions.
    (xr)                  value to be converted
    jsr  gtint            call to convert to integer
    ppm  loc              transfer loc for convert impossible
    (xr)                  resulting integer
    (wc,ra)               destroyed
    (wa,wb)               destroyed (only on conversion err)
    (xr)                  unchanged (on convert error)
gtint   prc  e,1                                 entry point
        beq  (xr),=b$icl,gtin2                   jump if already an integer
        mov  wa,gtina                            else save wa
        mov  wb,gtinb                            save wb
        jsr  gtnum                               convert to numeric
        ppm  gtin3                               jump if unconvertible
```

---

if **.cnra**
else

```
        beq  wa,=b$icl,gtin1                     jump if integer
    here we convert a real to integer
        ldr  rcval(xr)                           load real value
        rti  gtin3                               convert to integer (err if ovflow)
        jsr  icbld                               if ok build icblk
```

fi

```
    here after successful conversion to integer
gtin1   mov  gtina,wa                            restore wa
        mov  gtinb,wb                            restore wb
    common exit point
gtin2   exi                                      return to gtint caller
    here on conversion error
gtin3   exi  1                                   take convert error exit
        enp                                      end procedure gtint
```

```
    gtnum -- get numeric value
    gtnum is given an object and returns either an integer
    or a real, performing any necessary conversions.
    (xr)                  object to be converted
    jsr  gtnum            call to convert to numeric
    ppm  loc              transfer loc if convert impossible
    (xr)                  pointer to result (int or real)
    (wa)                  first word of result block
    (wb,wc,ra)            destroyed
    (xr)                  unchanged (on convert error)
gtnum   prc  e,1                                  entry point
        mov  (xr),wa                              load first word of block
        beq  wa,=b$icl,gtn34                      jump if integer (no conversion)
```

---

if **.cnra**
else

```
        beq  wa,=b$rcl,gtn34                      jump if real (no conversion)
```

fi

```
    at this point the only possibility is to convert a string
    to an integer or real as appropriate.
        mov  xr,-(xs)                             stack argument in case convert err
        mov  xr,-(xs)                             stack argument for gtstg
```

---

if **.cnbf**

```
        jsr  gtstg                                convert argument to string
```

else

```
        jsr  gtstb                                get argument as string or buffer
```

fi

```
        ppm  gtn36                                jump if unconvertible
    initialize numeric conversion
        ldi  intv0                                initialize integer result to zero
        bze  wa,gtn32                             jump to exit with zero if null
        lct  wa,wa                                set bct counter for following loops
        zer  gtnnf                                tentatively indicate result +
```

---

if **.cnra**
else

```
        sti  gtnex                                initialise exponent to zero
        zer  gtnsc                                zero scale in case real
        zer  gtndf                                reset flag for dec point found
        zer  gtnrd                                reset flag for digits found
        ldr  reav0                                zero real accum in case real
```

fi

```
        plc  xr                                   point to argument characters
    merge back here after ignoring leading blank
gtn01   lch  wb,(xr)+                             load first character
        blt  wb,=ch$d0,gtn02                      jump if not digit
        ble  wb,=ch$d9,gtn06                      jump if first char is a digit
```

```
    gtnum (continued)
    here if first digit is non-digit
gtn02  bne  wb,=ch$bl,gtn03          jump if non-blank
gtna2  bct  wa,gtn01                 else decr count and loop back
       brn  gtn07                    jump to return zero if all blanks
    here for first character non-blank, non-digit
gtn03  beq  wb,=ch$pl,gtn04          jump if plus sign
```

---

if **.caht**

```
       beq  wb,=ch$ht,gtna2          horizontal tab equiv to blank
```

fi

---

if **.cavt**

```
       beq  wb,=ch$vt,gtna2          vertical tab equiv to blank
```

fi

---

if **.cnra**

```
       bne  wb,=ch$mn,gtn36          else fail
```

else

```
       bne  wb,=ch$mn,gtn12          jump if not minus (may be real)
```

fi

```
       mnz  gtnnf                    if minus sign, set negative flag
    merge here after processing sign
gtn04  bct  wa,gtn05                 jump if chars left
       brn  gtn36                    else error
    loop to fetch characters of an integer
gtn05  lch  wb,(xr)+                 load next character
       blt  wb,=ch$d0,gtn08          jump if not a digit
       bgt  wb,=ch$d9,gtn08          jump if not a digit
    merge here for first digit
gtn06  sti  gtnsi                    save current value
```

---

if **.cnra**

```
       cvm  gtn36                    current*10-(new dig) jump if ovflow
```

else

```
       cvm  gtn35                    current*10-(new dig) jump if ovflow
       mnz  gtnrd                    set digit read flag
```

fi

```
       bct  wa,gtn05                 else loop back if more chars
    here to exit with converted integer value
gtn07  bnz  gtnnf,gtn32              jump if negative (all set)
       ngi                           else negate
       ino  gtn32                    jump if no overflow
       brn  gtn36                    else signal error
```

```
    gtnum (continued)
    here for a non-digit character while attempting to
    convert an integer, check for trailing blanks or real.
gtn08   beq   wb,=ch$bl,gtna9                  jump if a blank
-----------------------------------------------------------------
if .caht
        beq   wb,=ch$ht,gtna9                  jump if horizontal tab
fi
-----------------------------------------------------------------
if .cavt
        beq   wb,=ch$vt,gtna9                  jump if vertical tab
fi
-----------------------------------------------------------------
if .cnra
        brn   gtn36                            error
else
        itr                                    else convert integer to real
        ngr                                    negate to get positive value
        brn   gtn12                            jump to try for real
fi
    here we scan out blanks to end of string
gtn09   lch   wb,(xr)+                         get next char
-----------------------------------------------------------------
if .caht
        beq   wb,=ch$ht,gtna9                  jump if horizontal tab
fi
-----------------------------------------------------------------
if .cavt
        beq   wb,=ch$vt,gtna9                  jump if vertical tab
fi
        bne   wb,=ch$bl,gtn36                  error if non-blank
gtna9   bct   wa,gtn09                         loop back if more chars to check
        brn   gtn07                            return integer if all blanks
-----------------------------------------------------------------
if .cnra
else
    loop to collect mantissa of real
gtn10   lch   wb,(xr)+                         load next character
        blt   wb,=ch$d0,gtn12                  jump if non-numeric
        bgt   wb,=ch$d9,gtn12                  jump if non-numeric
    merge here to collect first real digit
gtn11   sub   =ch$d0,wb                        convert digit to number
        mlr   reavt                            multiply real by 10.0
        rov   gtn36                            convert error if overflow
        str   gtnsr                            save result
        mti   wb                               get new digit as integer
        itr                                    convert new digit to real
        adr   gtnsr                            add to get new total
        add   gtndf,gtnsc                      increment scale if after dec point
        mnz   gtnrd                            set digit found flag
        bct   wa,gtn10                         loop back if more chars
        brn   gtn22                            else jump to scale
```

```
    gtnum (continued)
    here if non-digit found while collecting a real
gtn12   bne   wb,=ch$dt,gtn13                     jump if not dec point
        bnz   gtndf,gtn36                         if dec point, error if one already
        mov   =num01,gtndf                        else set flag for dec point
        bct   wa,gtn10                            loop back if more chars
        brn   gtn22                               else jump to scale
    here if not decimal point
gtn13   beq   wb,=ch$le,gtn15                     jump if e for exponent
        beq   wb,=ch$ld,gtn15                     jump if d for exponent
```

---

if **.culc**

```
        beq   wb,=ch$$e,gtn15                     jump if e for exponent
        beq   wb,=ch$$d,gtn15                     jump if d for exponent
```

fi

```
    here check for trailing blanks
gtn14   beq   wb,=ch$bl,gtnb4                     jump if blank
```

---

if **.caht**

```
        beq   wb,=ch$ht,gtnb4                     jump if horizontal tab
```

fi

---

if **.cavt**

```
        beq   wb,=ch$vt,gtnb4                     jump if vertical tab
```

fi

```
        brn   gtn36                               error if non-blank
gtnb4   lch   wb,(xr)+                            get next character
        bct   wa,gtn14                            loop back to check if more
        brn   gtn22                               else jump to scale
    here to read and process an exponent
gtn15   zer   gtnes                               set exponent sign positive
        ldi   intv0                               initialize exponent to zero
        mnz   gtndf                               reset no dec point indication
        bct   wa,gtn16                            jump skipping past e or d
        brn   gtn36                               error if null exponent
    check for exponent sign
gtn16   lch   wb,(xr)+                            load first exponent character
        beq   wb,=ch$pl,gtn17                     jump if plus sign
        bne   wb,=ch$mn,gtn19                     else jump if not minus sign
        mnz   gtnes                               set sign negative if minus sign
    merge here after processing exponent sign
gtn17   bct   wa,gtn18                            jump if chars left
        brn   gtn36                               else error
    loop to convert exponent digits
gtn18   lch   wb,(xr)+                            load next character
```

```
    gtnum (continued)
    merge here for first exponent digit
gtn19   blt   wb,=ch$d0,gtn20                 jump if not digit
        bgt   wb,=ch$d9,gtn20                 jump if not digit
        cvm   gtn36                           else current*10, subtract new digit
        bct   wa,gtn18                        loop back if more chars
        brn   gtn21                           jump if exponent field is exhausted
    here to check for trailing blanks after exponent
gtn20   beq   wb,=ch$bl,gtnc0                 jump if blank
```

---

if **.caht**

```
        beq   wb,=ch$ht,gtnc0                 jump if horizontal tab
```

fi

---

if **.cavt**

```
        beq   wc,=ch$vt,gtnc0                 jump if vertical tab
```

fi

```
        brn   gtn36                           error if non-blank
gtnc0   lch   wb,(xr)+                        get next character
        bct   wa,gtn20                        loop back till all blanks scanned
    merge here after collecting exponent
gtn21   sti   gtnex                           save collected exponent
        bnz   gtnes,gtn22                     jump if it was negative
        ngi                                   else complement
        iov   gtn36                           error if overflow
        sti   gtnex                           and store positive exponent
    merge here with exponent (0 if none given)
gtn22   bze   gtnrd,gtn36                     error if not digits collected
        bze   gtndf,gtn36                     error if no exponent or dec point
        mti   gtnsc                           else load scale as integer
        sbi   gtnex                           subtract exponent
        iov   gtn36                           error if overflow
        ilt   gtn26                           jump if we must scale up
    here we have a negative exponent, so scale down
        mfi   wa,gtn36                        load scale factor, err if ovflow
    loop to scale down in steps of 10**10
gtn23   ble   wa,=num10,gtn24                 jump if 10 or less to go
        dvr   reatt                           else divide by 10**10
        sub   =num10,wa                       decrement scale
        brn   gtn23                           and loop back
```

```
    gtnum (continued)
    here scale rest of way from powers of ten table
gtn24  bze  wa,gtn30             jump if scaled
       lct  wb,=cfp$r            else get indexing factor
       mov  =reav1,xr            point to powers of ten table
       wtb  wa                   convert remaining scale to byte ofs
    loop to point to powers of ten table entry
gtn25  add  wa,xr                bump pointer
       bct  wb,gtn25             once for each value word
       dvr  (xr)                 scale down as required
       brn  gtn30                and jump
    come here to scale result up (positive exponent)
gtn26  ngi                       get absolute value of exponent
       iov  gtn36                error if overflow
       mfi  wa,gtn36             acquire scale, error if ovflow
    loop to scale up in steps of 10**10
gtn27  ble  wa,=num10,gtn28      jump if 10 or less to go
       mlr  reatt                else multiply by 10**10
       rov  gtn36                error if overflow
       sub  =num10,wa            else decrement scale
       brn  gtn27                and loop back
    here to scale up rest of way with table
gtn28  bze  wa,gtn30             jump if scaled
       lct  wb,=cfp$r            else get indexing factor
       mov  =reav1,xr            point to powers of ten table
       wtb  wa                   convert remaining scale to byte ofs
    loop to point to proper entry in powers of ten table
gtn29  add  wa,xr                bump pointer
       bct  wb,gtn29             once for each word in value
       mlr  (xr)                 scale up
       rov  gtn36                error if overflow
```

```
    gtnum (continued)
    here with real value scaled and ready except for sign
gtn30   bze   gtnnf,gtn31                              jump if positive
        ngr                                            else negate
    here with properly signed real value in (ra)
gtn31   jsr   rcbld                                    build real block
        brn   gtn33                                    merge to exit
fi
    here with properly signed integer value in (ia)
gtn32   jsr   icbld                                    build icblk
    real merges here
gtn33   mov   (xr),wa                                  load first word of result block
        ica   xs                                       pop argument off stack
    common exit point
gtn34   exi                                            return to gtnum caller
```

```
if .cnra
else
    come here if overflow occurs during collection of integer
    have to restore wb which cvm may have destroyed.
gtn35   lch   wb,-(xr)                                 reload current character
        lch   wb,(xr)+                                 bump character pointer
        ldi   gtnsi                                    reload integer so far
        itr                                            convert to real
        ngr                                            make value positive
        brn   gtn11                                    merge with real circuit
fi
    here for unconvertible to string or conversion error
gtn36   mov   (xs)+,xr                                 reload original argument
        exi   1                                        take convert-error exit
        enp                                            end procedure gtnum
```

```
    gtnvr -- convert to natural variable
    gtnvr locates a variable block (vrblk) given either an
    appropriate name (nmblk) or a non-null string (scblk).
    (xr)                  argument
    jsr  gtnvr            call to convert to natural variable
    ppm  loc              transfer loc if convert impossible
    (xr)                  pointer to vrblk
    (wa,wb)               destroyed (conversion error only)
    (wc)                  destroyed
gtnvr  prc  e,1                                   entry point
       bne  (xr),=b$nml,gnv02                     jump if not name
       mov  nmbas(xr),xr                          else load name base if name
       blo  xr,state,gnv07                        skip if vrblk (in static region)
    common error exit
gnv01  exi  1                                     take convert-error exit
    here if not name
gnv02  mov  wa,gnvsa                              save wa
       mov  wb,gnvsb                              save wb
       mov  xr,-(xs)                              stack argument for gtstg
       jsr  gtstg                                 convert argument to string
       ppm  gnv01                                 jump if conversion error
       bze  wa,gnv01                              null string is an error
```

```
if .culc
       jsr  flstg                                 fold lower case to upper case
fi
       mov  xl,-(xs)                              save xl
       mov  xr,-(xs)                              stack string ptr for later
       mov  xr,wb                                 copy string pointer
       add  *schar,wb                             point to characters of string
       mov  wb,gnvst                              save pointer to characters
       mov  wa,wb                                 copy length
       ctw  wb,0                                  get number of words in name
       mov  wb,gnvnw                              save for later
       jsr  hashs                                 compute hash index for string
       rmi  hshnb                                 compute hash offset by taking mod
       mfi  wc                                    get as offset
       wtb  wc                                    convert offset to bytes
       add  hshtb,wc                              point to proper hash chain
       sub  *vrnxt,wc                             subtract offset to merge into loop
```

```
    gtnvr (continued)
    loop to search hash chain
gnv03  mov  wc,xl                          copy hash chain pointer
       mov  vrnxt(xl),xl                   point to next vrblk on chain
       bze  xl,gnv08                       jump if end of chain
       mov  xl,wc                          save pointer to this vrblk
       bnz  vrlen(xl),gnv04                jump if not system variable
       mov  vrsvp(xl),xl                   else point to svblk
       sub  *vrsof,xl                      adjust offset for merge
    merge here with string ptr (like vrblk) in xl
gnv04  bne  wa,vrlen(xl),gnv03             back for next vrblk if lengths ne
       add  *vrchs,xl                      else point to chars of chain entry
       lct  wb,gnvnw                       get word counter to control loop
       mov  gnvst,xr                       point to chars of new name
    loop to compare characters of the two names
gnv05  cne  (xr),(xl),gnv03                jump if no match for next vrblk
       ica  xr                             bump new name pointer
       ica  xl                             bump vrblk in chain name pointer
       bct  wb,gnv05                       else loop till all compared
       mov  wc,xr                          we have found a match, get vrblk
    exit point after finding vrblk or building new one
gnv06  mov  gnvsa,wa                       restore wa
       mov  gnvsb,wb                       restore wb
       ica  xs                             pop string pointer
       mov  (xs)+,xl                       restore xl
    common exit point
gnv07  exi                                 return to gtnvr caller
    not found, prepare to search system variable table
gnv08  zer  xr                             clear garbage xr pointer
       mov  wc,gnvhe                       save ptr to end of hash chain
       bgt  wa,=num09,gnv14                cannot be system var if length gt 9
       mov  wa,xl                          else copy length
       wtb  xl                             convert to byte offset
       mov  vsrch(xl),xl                   point to first svblk of this length
```

```
    gtnvr (continued)
    loop to search entries in standard variable table
gnv09   mov  xl,gnvsp                       save table pointer
        mov  (xl)+,wc                       load svbit bit string
        mov  (xl)+,wb                       load length from table entry
        bne  wa,wb,gnv14                    jump if end of right length entries
        lct  wb,gnvnw                       get word counter to control loop
        mov  gnvst,xr                       point to chars of new name
    loop to check for matching names
gnv10   cne  (xr),(xl),gnv11                jump if name mismatch
        ica  xr                             else bump new name pointer
        ica  xl                             bump svblk pointer
        bct  wb,gnv10                       else loop until all checked
    here we have a match in the standard variable table
        zer  wc                             set vrlen value zero
        mov  *vrsi$,wa                      set standard size
        brn  gnv15                          jump to build vrblk
    here if no match with table entry in svblks table
gnv11   ica  xl                             bump past word of chars
        bct  wb,gnv11                       loop back if more to go
        rsh  wc,svnbt                       remove uninteresting bits
    loop to bump table ptr for each flagged word
gnv12   mov  bits1,wb                       load bit to test
        anb  wc,wb                          test for word present
        zrb  wb,gnv13                       jump if not present
        ica  xl                             else bump table pointer
    here after dealing with one word (one bit)
gnv13   rsh  wc,1                           remove bit already processed
        nzb  wc,gnv12                       loop back if more bits to test
        brn  gnv09                          else loop back for next svblk
    here if not system variable
gnv14   mov  wa,wc                          copy vrlen value
        mov  =vrchs,wa                      load standard size -chars
        add  gnvnw,wa                       adjust for chars of name
        wtb  wa                             convert length to bytes
```

```
    gtnvr (continued)
    merge here to build vrblk
gnv15   jsr   alost                allocate space for vrblk (static)
        mov   xr,wb                save vrblk pointer
        mov   =stnvr,xl            point to model variable block
        mov   *vrlen,wa            set length of standard fields
        mvw                        set initial fields of new block
        mov   gnvhe,xl             load pointer to end of hash chain
        mov   wb,vrnxt(xl)         add new block to end of chain
        mov   wc,(xr)+             set vrlen field, bump ptr
        mov   gnvnw,wa             get length in words
        wtb   wa                   convert to length in bytes
        bze   wc,gnv16             jump if system variable
    here for non-system variable -- set chars of name
        mov   (xs),xl              point back to string name
        add   *schar,xl            point to chars of name
        mvw                        move characters into place
        mov   wb,xr                restore vrblk pointer
        brn   gnv06                jump back to exit
    here for system variable case to fill in fields where
    necessary from the fields present in the svblk.
gnv16   mov   gnvsp,xl             load pointer to svblk
        mov   xl,(xr)              set svblk ptr in vrblk
        mov   wb,xr                restore vrblk pointer
        mov   svbit(xl),wb         load bit indicators
        add   *svchs,xl            point to characters of name
        add   wa,xl                point past characters
    skip past keyword number (svknm) if present
        mov   btknm,wc             load test bit
        anb   wb,wc                and to test
        zrb   wc,gnv17             jump if no keyword number
        ica   xl                   else bump pointer
```

```
    gtnvr (continued)
    here test for function (svfnc and svnar)
gnv17  mov  btfnc,wc              get test bit
       anb  wb,wc                 and to test
       zrb  wc,gnv18              skip if no system function
       mov  xl,vrfnc(xr)          else point vrfnc to svfnc field
       add  *num02,xl             and bump past svfnc, svnar fields
    now test for label (svlbl)
gnv18  mov  btlbl,wc              get test bit
       anb  wb,wc                 and to test
       zrb  wc,gnv19              jump if bit is off (no system labl)
       mov  xl,vrlbl(xr)          else point vrlbl to svlbl field
       ica  xl                    bump past svlbl field
    now test for value (svval)
gnv19  mov  btval,wc              load test bit
       anb  wb,wc                 and to test
       zrb  wc,gnv06              all done if no value
       mov  (xl),vrval(xr)        else set initial value
       mov  =b$vre,vrsto(xr)      set error store access
       brn  gnv06                 merge back to exit to caller
       enp                        end procedure gtnvr
```

```
    gtpat -- get pattern
    gtpat is passed an object in (xr) and returns a
    pattern after performing any necessary conversions
    (xr)                 input argument
    jsr  gtpat           call to convert to pattern
    ppm  loc             transfer loc if convert impossible
    (xr)                 resulting pattern
    (wa)                 destroyed
    (wb)                 destroyed (only on convert error)
    (xr)                 unchanged (only on convert error)
gtpat   prc  e,1                                   entry point
        bhi  (xr),=p$aaa,gtpt5                     jump if pattern already
    here if not pattern, try for string
        mov  wb,gtpsb                              save wb
        mov  xr,-(xs)                              stack argument for gtstg
        jsr  gtstg                                 convert argument to string
        ppm  gtpt2                                 jump if impossible
    here we have a string
        bnz  wa,gtpt1                              jump if non-null
    here for null string. generate pointer to null pattern.
        mov  =ndnth,xr                             point to nothen node
        brn  gtpt4                                 jump to exit
```

```
    gtpat (continued)
    here for non-null string
gtpt1   mov  =p$str,wb                         load pcode for multi-char string
        bne  wa,=num01,gtpt3                   jump if multi-char string
    here for one character string, share one character any
        plc  xr                                point to character
        lch  wa,(xr)                           load character
        mov  wa,xr                             set as parm1
        mov  =p$ans,wb                         point to pcode for 1-char any
        brn  gtpt3                             jump to build node
    here if argument is not convertible to string
gtpt2   mov  =p$exa,wb                         set pcode for expression in case
        blo  (xr),=b$e$$,gtpt3                 jump to build node if expression
    here we have an error (conversion impossible)
        exi  1                                 take convert error exit
    merge here to build node for string or expression
gtpt3   jsr  pbild                             call routine to build pattern node
    common exit after successful conversion
gtpt4   mov  gtpsb,wb                          restore wb
    merge here to exit if no conversion required
gtpt5   exi                                    return to gtpat caller
        enp                                    end procedure gtpat
```

---

if **.cnra**

else

```
    gtrea -- get real value
    gtrea is passed an object and returns a real value
    performing any necessary conversions.
    (xr)                 object to be converted
    jsr  gtrea           call to convert object to real
    ppm  loc             transfer loc if convert impossible
    (xr)                 pointer to resulting real
    (wa,wb,wc,ra)        destroyed
    (xr)                 unchanged (convert error only)
gtrea  prc  e,1                          entry point
       mov  (xr),wa                      get first word of block
       beq  wa,=b$rcl,gtre2              jump if real
       jsr  gtnum                        else convert argument to numeric
       ppm  gtre3                        jump if unconvertible
       beq  wa,=b$rcl,gtre2              jump if real was returned
    here for case of an integer to convert to real
gtre1  ldi  icval(xr)                    load integer
       itr                               convert to real
       jsr  rcbld                        build rcblk
    exit with real
gtre2  exi                               return to gtrea caller
    here on conversion error
gtre3  exi  1                            take convert error exit
       enp                               end procedure gtrea
fi
```

```
    gtsmi -- get small integer
    gtsmi is passed a snobol object and returns an address
    integer in the range (0 le n le dnamb). such a value can
    only be derived from an integer in the appropriate range.
    small integers never appear as snobol values. however,
    they are used internally for a variety of purposes.
    -(xs)                  argument to convert (on stack)
    jsr  gtsmi             call to convert to small integer
    ppm  loc               transfer loc for not integer
    ppm  loc               transfer loc for lt 0, gt dnamb
    (xr,wc)                resulting small int (two copies)
    (xs)                   popped
    (ra)                   destroyed
    (wa,wb)                destroyed (on convert error only)
    (xr)                   input arg (convert error only)
gtsmi   prc  n,2                                  entry point
        mov  (xs)+,xr                             load argument
        beq  (xr),=b$icl,gtsm1                    skip if already an integer
    here if not an integer
        jsr  gtint                                convert argument to integer
        ppm  gtsm2                                jump if convert is impossible
    merge here with integer
gtsm1   ldi  icval(xr)                            load integer value
        mfi  wc,gtsm3                             move as one word, jump if ovflow
        bgt  wc,mxlen,gtsm3                       or if too large
        mov  wc,xr                                copy result to xr
        exi                                       return to gtsmi caller
    here if unconvertible to integer
gtsm2   exi  1                                    take non-integer error exit
    here if out of range
gtsm3   exi  2                                    take out-of-range error exit
        enp                                       end procedure gtsmi
```

if **.cnbf**

else

```
    gtstb -- get string or buffer
    gtstb is passed an object and returns it unchanged if
    it is a buffer block, else it returns it as a string with
    any necessary conversions performed.
    -(xs)                 input argument (on stack)
    jsr  gtstb            call to get buffer or cnvrt to stg
    ppm  loc              transfer loc if convert impossible
    (xr)                  pointer to resulting scblk or bfblk
    (wa)                  length of string in characters
    (wb)                  zero/bcblk if string/buffer
    (xs)                  popped
    (ra)                  destroyed
    (xr)                  input arg (convert error only)
gtstb   prc  n,1                              entry point
        mov  (xs),xr                          load argument, leave on stack
        mov  (xr),wa                          load block type
        beq  wa,=b$scl,gtsb2                  jump if already a string
        beq  wa,=b$bct,gtsb3                  jump if already a buffer
        jsr  gtstg                            convert to string
        ppm  gtsb1                            conversion failed
        zer  wb                               signal string result
        exi                                   convert with string result
    here if conversion failed
gtsb1   exi  1                                take convert error exit
    here if a string already
gtsb2   ica  xs                               pop argument
        mov  sclen(xr),wa                     load string length
        zer  wb                               signal string result
        exi                                   return with string result
    here if it is already a buffer
gtsb3   ica  xs                               pop argument
        mov  bclen(xr),wa                     load length of string in buffer
        mov  xr,wb                            return bcblk pointer in wb
        mov  bcbuf(xr),xr                     return bfblk pointer in xr
        exi                                   return with buffer result
        enp                                   end procedure gtstg
```

fi

```
    gtstg -- get string
    gtstg is passed an object and returns a string with
    any necessary conversions performed.
    -(xs)                  input argument (on stack)
    jsr  gtstg             call to convert to string
    ppm  loc               transfer loc if convert impossible
    (xr)                   pointer to resulting string
    (wa)                   length of string in characters
    (xs)                   popped
    (ra)                   destroyed
    (xr)                   input arg (convert error only)
gtstg   prc  n,1                                    entry point
        mov  (xs)+,xr                               load argument, pop stack
        beq  (xr),=b$scl,gts30                      jump if already a string
    here if not a string already
gts01   mov  xr,-(xs)                               restack argument in case error
        mov  xl,-(xs)                               save xl
        mov  wb,gtsvb                               save wb
        mov  wc,gtsvc                               save wc
        mov  (xr),wa                                load first word of block
        beq  wa,=b$icl,gts05                        jump to convert integer
```

---

if **.cnra**

else

```
        beq  wa,=b$rcl,gts10                        jump to convert real
```

fi

```
        beq  wa,=b$nml,gts03                        jump to convert name
```

---

if **.cnbf**

else

```
        beq  wa,=b$bct,gts32                        jump to convert buffer
```

fi

```
    here on conversion error
gts02   mov  (xs)+,xl                               restore xl
        mov  (xs)+,xr                               reload input argument
        exi  1                                      take convert error exit
```

```
    gtstg (continued)
    here to convert a name (only possible if natural var)
gts03   mov  nmbas(xr),xl                       load name base
        bhi  xl,state,gts02                     error if not natural var (static)
        add  *vrsof,xl                          else point to possible string name
        mov  sclen(xl),wa                       load length
        bnz  wa,gts04                           jump if not system variable
        mov  vrsvo(xl),xl                       else point to svblk
        mov  svlen(xl),wa                       and load name length
    merge here with string in xr, length in wa
gts04   zer  wb                                 set offset to zero
        jsr  sbstr                              use sbstr to copy string
        brn  gts29                              jump to exit
    come here to convert an integer
gts05   ldi  icval(xr)                          load integer value
```

---

if **.cnci**

```
        jsr  sysci                              convert integer
        mov  sclen(xl),wa                       get length
        zer  wb                                 zero offset for sbstr
        jsr  sbstr                              copy in result from sysci
        brn  gts29                              exit
```

else

```
        mov  =num01,gtssf                       set sign flag negative
        ilt  gts06                              skip if integer is negative
        ngi                                     else negate integer
        zer  gtssf                              and reset negative flag
```

```
    gtstg (continued)
    here with sign flag set and sign forced negative as
    required by the cvd instruction.
gts06   mov  gtswk,xr                       point to result work area
        mov  =nstmx,wb                      initialize counter to max length
        psc  xr,wb                          prepare to store (right-left)
    loop to convert digits into work area
gts07   cvd                                 convert one digit into wa
        sch  wa,-(xr)                       store in work area
        dcv  wb                             decrement counter
        ine  gts07                          loop if more digits to go
        csc  xr                             complete store characters
fi
    merge here after converting integer or real into work
    area. wb is set to nstmx - (number of chars in result).
gts08   mov  =nstmx,wa                      get max number of characters
        sub  wb,wa                          compute length of result
        mov  wa,xl                          remember length for move later on
        add  gtssf,wa                       add one for negative sign if needed
        jsr  alocs                          allocate string for result
        mov  xr,wc                          save result pointer for the moment
        psc  xr                             point to chars of result block
        bze  gtssf,gts09                    skip if positive
        mov  =ch$mn,wa                      else load negative sign
        sch  wa,(xr)+                       and store it
        csc  xr                             complete store characters
    here after dealing with sign
gts09   mov  xl,wa                          recall length to move
        mov  gtswk,xl                       point to result work area
        plc  xl,wb                          point to first result character
        mvc                                 move chars to result string
        mov  wc,xr                          restore result pointer
```

---

if **.cnra**
else

```
        brn  gts29                          jump to exit
```

```
    gtstg (continued)
    here to convert a real
gts10  ldr   rcval(xr)                        load real
----------------------------------------------------------------------
if .cncr
       mov   =nstmr,wa                        max number of result chars
       zer   xl                               clear dud value
       jsr   alocs                            allocate result area
       mov   =cfp$s,wa                        significant digits to produce
       zer   wb                               conversion type
       jsr   syscr                            convert real to string
       mov   wa,sclen(xr)                     store result size
       zer   wb                               no trailing blanks to remove
       jsr   trimr                            discard excess memory
else
       zer   gtssf                            reset negative flag
       req   gts31                            skip if zero
       rge   gts11                            jump if real is positive
       mov   =num01,gtssf                     else set negative flag
       ngr                                    and get absolute value of real
    now scale the real to the range (0.1 le x lt 1.0)
gts11  ldi   intv0                            initialize exponent to zero
    loop to scale up in steps of 10**10
gts12  str   gtsrs                            save real value
       sbr   reap1                            subtract 0.1 to compare
       rge   gts13                            jump if scale up not required
       ldr   gtsrs                            else reload value
       mlr   reatt                            multiply by 10**10
       sbi   intvt                            decrement exponent by 10
       brn   gts12                            loop back to test again
    test for scale down required
gts13  ldr   gtsrs                            reload value
       sbr   reav1                            subtract 1.0
       rlt   gts17                            jump if no scale down required
       ldr   gtsrs                            else reload value
    loop to scale down in steps of 10**10
gts14  sbr   reatt                            subtract 10**10 to compare
       rlt   gts15                            jump if large step not required
       ldr   gtsrs                            else restore value
       dvr   reatt                            divide by 10**10
       str   gtsrs                            store new value
       adi   intvt                            increment exponent by 10
       brn   gts14                            loop back
```

```
    gtstg (continued)
    at this point we have (1.0 le x lt 10**10)
    complete scaling with powers of ten table
gts15   mov  =reav1,xr                               point to powers of ten table
    loop to locate correct entry in table
gts16   ldr   gtsrs                                  reload value
        adi   intv1                                  increment exponent
        add   *cfp$r,xr                              point to next entry in table
        sbr   (xr)                                   subtract it to compare
        rge   gts16                                  loop till we find a larger entry
        ldr   gtsrs                                  then reload the value
        dvr   (xr)                                   and complete scaling
        str   gtsrs                                  store value
    we are now scaled, so round by adding 0.5 * 10**(-cfp$s)
gts17   ldr   gtsrs                                  get value again
        adr   gtsrn                                  add rounding factor
        str   gtsrs                                  store result
    the rounding operation may have pushed us up past
    1.0 again, so check one more time.
        sbr   reav1                                  subtract 1.0 to compare
        rlt   gts18                                  skip if ok
        adi   intv1                                  else increment exponent
        ldr   gtsrs                                  reload value
        dvr   reavt                                  divide by 10.0 to rescale
        brn   gts19                                  jump to merge
    here if rounding did not muck up scaling
gts18   ldr   gtsrs                                  reload rounded value
```

```
    gtstg (continued)
    now we have completed the scaling as follows
    (ia)                  signed exponent
    (ra)                  scaled real (absolute value)
    if the exponent is negative or greater than cfp$s, then
    we convert the number in the form.
    (neg sign) 0 . (cpf$s digits) e (exp sign) (exp digits)
    if the exponent is positive and less than or equal to
    cfp$s, the number is converted in the form.
    (neg sign) (exponent digits) . (cfp$s-exponent digits)
    in both cases, the formats obtained from the above
    rules are modified by deleting trailing zeros after the
    decimal point. there are no leading zeros in the exponent
    and the exponent sign is always present.
gts19  mov  =cfp$s,xl          set num dec digits = cfp$s
       mov  =ch$mn,gtses       set exponent sign negative
       ilt  gts21              all set if exponent is negative
       mfi  wa                 else fetch exponent
       ble  wa,=cfp$s,gts20    skip if we can use special format
       mti  wa                 else restore exponent
       ngi                     set negative for cvd
       mov  =ch$pl,gtses       set plus sign for exponent sign
       brn  gts21              jump to generate exponent
    here if we can use the format without an exponent
gts20  sub  wa,xl              compute digits after decimal point
       ldi  intv0              reset exponent to zero
```

```
    gtstg (continued)
    merge here as follows
    (ia)                 exponent absolute value
    gtses                character for exponent sign
    (ra)                 positive fraction
    (xl)                 number of digits after dec point
gts21   mov  gtswk,xr                        point to work area
        mov  =nstmx,wb                       set character ctr to max length
        psc  xr,wb                           prepare to store (right to left)
        ieq  gts23                           skip exponent if it is zero
    loop to generate digits of exponent
gts22   cvd                                  convert a digit into wa
        sch  wa,-(xr)                        store in work area
        dcv  wb                              decrement counter
        ine  gts22                           loop back if more digits to go
    here generate exponent sign and e
        mov  gtses,wa                        load exponent sign
        sch  wa,-(xr)                        store in work area
        mov  =ch$le,wa                       get character letter e
        sch  wa,-(xr)                        store in work area
        sub  =num02,wb                       decrement counter for sign and e
    here to generate the fraction
gts23   mlr  gtssc                           convert real to integer (10**cfp$s)
        rti                                  get integer (overflow impossible)
        ngi                                  negate as required by cvd
    loop to suppress trailing zeros
gts24   bze  xl,gts27                        jump if no digits left to do
        cvd                                  else convert one digit
        bne  wa,=ch$d0,gts26                 jump if not a zero
        dcv  xl                              decrement counter
        brn  gts24                           loop back for next digit
```

```
    gtstg (continued)
    loop to generate digits after decimal point
gts25   cvd                                     convert a digit into wa
    merge here first time
gts26   sch   wa,-(xr)                          store digit
        dcv   wb                                decrement counter
        dcv   xl                                decrement counter
        bnz   xl,gts25                          loop back if more to go
    here generate the decimal point
gts27   mov   =ch$dt,wa                         load decimal point
        sch   wa,-(xr)                          store in work area
        dcv   wb                                decrement counter
    here generate the digits before the decimal point
gts28   cvd                                     convert a digit into wa
        sch   wa,-(xr)                          store in work area
        dcv   wb                                decrement counter
        ine   gts28                             loop back if more to go
        csc   xr                                complete store characters
        brn   gts08                             else jump back to exit
fi
fi
    exit point after successful conversion
gts29   mov   (xs)+,xl                          restore xl
        ica   xs                                pop argument
        mov   gtsvb,wb                          restore wb
        mov   gtsvc,wc                          restore wc
    merge here if no conversion required
gts30   mov   sclen(xr),wa                      load string length
        exi                                     return to caller
```

---

if **.cnra**
else

```
    here to return string for real zero
gts31   mov   =scre0,xl                         point to string
        mov   =num02,wa                         2 chars
        zer   wb                                zero offset
        jsr   sbstr                             copy string
        brn   gts29                             return
fi
```

---

if **.cnbf**
else

```
    here to convert a buffer block
gts32  mov  xr,xl              copy arg ptr
       mov  bclen(xl),wa       get size to allocate
       bze  wa,gts33           if null then return null
       jsr  alocs              allocate string frame
       mov  xr,wb              save string ptr
       mov  sclen(xr),wa       get length to move
       ctb  wa,0               get as multiple of word size
       mov  bcbuf(xl),xl       point to bfblk
       add  *scsi$,xr          point to start of character area
       add  *bfsi$,xl          point to start of buffer chars
       mvw                     copy words
       mov  wb,xr              restore scblk ptr
       brn  gts29              exit with scblk
    here when null buffer is being converted
gts33  mov  =nulls,xr          point to null
       brn  gts29              exit with null
fi
       enp                     end procedure gtstg
```

```
    gtvar -- get variable for i/o/trace association
    gtvar is used to point to an actual variable location
    for the detach,input,output,trace,stoptr system functions
    (xr)                  argument to function
    jsr  gtvar            call to locate variable pointer
    ppm  loc              transfer loc if not ok variable
    (xl,wa)               name base,offset of variable
    (xr,ra)               destroyed
    (wb,wc)               destroyed (convert error only)
    (xr)                  input arg (convert error only)
gtvar   prc  e,1                                  entry point
        bne  (xr),=b$nml,gtvr2                    jump if not a name
        mov  nmofs(xr),wa                         else load name offset
        mov  nmbas(xr),xl                         load name base
        beq  (xl),=b$evt,gtvr1                    error if expression variable
        bne  (xl),=b$kvt,gtvr3                    all ok if not keyword variable
    here on conversion error
gtvr1   exi  1                                    take convert error exit
    here if not a name, try convert to natural variable
gtvr2   mov  wc,gtvrc                             save wc
        jsr  gtnvr                                locate vrblk if possible
        ppm  gtvr1                                jump if convert error
        mov  xr,xl                                else copy vrblk name base
        mov  *vrval,wa                            and set offset
        mov  gtvrc,wc                             restore wc
    here for name obtained
gtvr3   bhi  xl,state,gtvr4                       all ok if not natural variable
        beq  vrsto(xl),=b$vre,gtvr1               error if protected variable
    common exit point
gtvr4   exi                                       return to caller
        enp                                       end procedure gtvar
```

```
    hashs -- compute hash index for string
    hashs is used to convert a string to a unique integer
    value. the resulting hash value is a positive integer
    in the range 0 to cfp$m
    (xr)                  string to be hashed
    jsr  hashs            call to hash string
    (ia)                  hash value
    (xr,wb,wc)            destroyed
    the hash function used is as follows.
    start with the length of the string (sgd07)
    take the first e$hnw words of the characters from
    the string or all the words if fewer than e$hnw.
    compute the exclusive or of all these words treating
    them as one word bit string values.
    move the result as an integer with the mti instruction.
hashs  prc  e,0                    entry point
       mov  sclen(xr),wc           load string length in characters
       mov  wc,wb                  initialize with length
       bze  wc,hshs3               jump if null string
       zgb  wb                     correct byte ordering if necessary
       ctw  wc,0                   get number of words of chars
       add  *schar,xr              point to characters of string
       blo  wc,=e$hnw,hshs1        use whole string if short
       mov  =e$hnw,wc              else set to involve first e$hnw wds
    here with count of words to check in wc
hshs1  lct  wc,wc                  set counter to control loop
    loop to compute exclusive or
hshs2  xob  (xr)+,wb               exclusive or next word of chars
       bct  wc,hshs2               loop till all processed
    merge here with exclusive or in wb
hshs3  zgb  wb                     zeroise undefined bits
       anb  bitsm,wb               ensure in range 0 to cfp$m
       mti  wb                     move result as integer
       zer  xr                     clear garbage value in xr
       exi                         return to hashs caller
       enp                         end procedure hashs
```

```
    icbld -- build integer block
    (ia)                 integer value for icblk
    jsr  icbld           call to build integer block
    (xr)                 pointer to result icblk
    (wa)                 destroyed
icbld   prc  e,0                                  entry point
        mfi  xr,icbl1                             copy small integers
        ble  xr,=num02,icbl3                      jump if 0,1 or 2
    construct icblk
icbl1   mov  dnamp,xr                             load pointer to next available loc
        add  *icsi$,xr                            point past new icblk
        blo  xr,dname,icbl2                       jump if there is room
        mov  *icsi$,wa                            else load length of icblk
        jsr  alloc                                use standard allocator to get block
        add  wa,xr                                point past block to merge
    merge here with xr pointing past the block obtained
icbl2   mov  xr,dnamp                             set new pointer
        sub  *icsi$,xr                            point back to start of block
        mov  =b$icl,(xr)                          store type word
        sti  icval(xr)                            store integer value in icblk
        exi                                       return to icbld caller
    optimise by not building icblks for small integers
icbl3   wtb  xr                                   convert integer to offset
        mov  intab(xr),xr                         point to pre-built icblk
        exi                                       return
        enp                                       end procedure icbld
```

```
    ident -- compare two values
    ident compares two values in the sense of the ident
    differ functions available at the snobol level.
    (xr)                 first argument
    (xl)                 second argument
    jsr  ident           call to compare arguments
    ppm  loc             transfer loc if ident
    (normal return if differ)
    (xr,xl,wc,ra)        destroyed
ident   prc  e,1                                entry point
        beq  xr,xl,iden7                        jump if same pointer (ident)
        mov  (xr),wc                            else load arg 1 type word
```

---

if **.cnbf**

```
        bne  wc,(xl),iden1                      differ if arg 2 type word differ
```

else

```
        bne  wc,(xl),iden0                      differ if arg 2 type word differ
```

fi

```
        beq  wc,=b$scl,iden2                    jump if strings
        beq  wc,=b$icl,iden4                    jump if integers
```

---

if **.cnra**

else

```
        beq  wc,=b$rcl,iden5                    jump if reals
```

fi

```
        beq  wc,=b$nml,iden6                    jump if names
```

---

if **.cnbf**

else

```
        bne  wc,=b$bct,iden1                    jump if not buffers
    here for buffers, ident only if lengths and chars same
        mov  bclen(xr),wc                       load arg 1 length
        bne  wc,bclen(xl),iden1                 differ if lengths differ
        bze  wc,iden7                           identical if length 0
        mov  bcbuf(xr),xr                       arg 1 buffer block
        mov  bcbuf(xl),xl                       arg 2 buffer block
        brn  idn2a                              compare characters
    here if the type words differ.
    check if string/buffer comparison
iden0   beq  wc,=b$scl,idn0a                    jump if arg 1 is a string
        bne  wc,=b$bct,iden1                    jump if arg 1 not string or buffer
    here if arg 1 is a buffer
        bne  (xl),=b$scl,iden1                  jump if arg 2 is not string
        mov  bclen(xr),wc                       load arg 1 length
        bne  wc,sclen(xl),iden1                 differ if lengths differ
        bze  wc,iden7                           identical if length 0
        mov  bcbuf(xr),xr                       arg 1 buffer block
        brn  idn2a                              compare characters
    here if arg 1 is a string
idn0a   bne  (xl),=b$bct,iden1                  jump if arg 2 is not buffer
        mov  sclen(xr),wc                       load arg 1 length
        bne  wc,bclen(xl),iden1                 differ if lengths differ
        bze  wc,iden7                           identical if length 0
        mov  bcbuf(xl),xl                       arg 2 buffer block
        brn  idn2a                              compare characters
```

fi

```
    for all other datatypes, must be differ if xr ne xl
    merge here for differ
iden1   exi                                     take differ exit
    here for strings, ident only if lengths and chars same
iden2   mov  sclen(xr),wc                       load arg 1 length
        bne  wc,sclen(xl),iden1                 differ if lengths differ
    buffer and string comparisons merge here
idn2a   add  *schar,xr                          point to chars of arg 1
        add  *schar,xl                          point to chars of arg 2
        ctw  wc,0                               get number of words in strings
        lct  wc,wc                              set loop counter
    loop to compare characters. note that wc cannot be zero
    since all null strings point to nulls and give xl=xr.
iden3   cne  (xr),(xl),iden8                    differ if chars do not match
        ica  xr                                 else bump arg one pointer
        ica  xl                                 bump arg two pointer
        bct  wc,iden3                           loop back till all checked
```

```
    ident (continued)
    here to exit for case of two ident strings
        zer   xl                                  clear garbage value in xl
        zer   xr                                  clear garbage value in xr
        exi   1                                   take ident exit
    here for integers, ident if same values
iden4   ldi   icval(xr)                           load arg 1
        sbi   icval(xl)                           subtract arg 2 to compare
        iov   iden1                               differ if overflow
        ine   iden1                               differ if result is not zero
        exi   1                                   take ident exit
```

---

if **.cnra**
else

```
    here for reals, ident if same values
iden5   ldr   rcval(xr)                           load arg 1
        sbr   rcval(xl)                           subtract arg 2 to compare
        rov   iden1                               differ if overflow
        rne   iden1                               differ if result is not zero
        exi   1                                   take ident exit
```

fi

```
    here for names, ident if bases and offsets same
iden6   bne   nmofs(xr),nmofs(xl),iden1           differ if different offset
        bne   nmbas(xr),nmbas(xl),iden1           differ if different base
    merge here to signal ident for identical pointers
iden7   exi   1                                   take ident exit
    here for differ strings
iden8   zer   xr                                  clear garbage ptr in xr
        zer   xl                                  clear garbage ptr in xl
        exi                                       return to caller (differ)
        enp                                       end procedure ident
```

```
    inout - used to initialise input and output variables
    (xl)                 pointer to vbl name string
    (wb)                 trblk type
    jsr  inout           call to perform initialisation
    (xl)                 vrblk ptr
    (xr)                 trblk ptr
    (wa,wc)              destroyed
    note that trter (= trtrf) field of standard i/o variables
    points to corresponding svblk not to a trblk as is the
    case for ordinary variables.
inout  prc  e,0                                  entry point
       mov  wb,-(xs)                             stack trblk type
       mov  sclen(xl),wa                         get name length
       zer  wb                                   point to start of name
       jsr  sbstr                                build a proper scblk
       jsr  gtnvr                                build vrblk
       ppm                                       no error return
       mov  xr,wc                                save vrblk pointer
       mov  (xs)+,wb                             get trter field
       zer  xl                                   zero trfpt
       jsr  trbld                                build trblk
       mov  wc,xl                                recall vrblk pointer
       mov  vrsvp(xl),trter(xr)                  store svblk pointer
       mov  xr,vrval(xl)                         store trblk ptr in vrblk
       mov  =b$vra,vrget(xl)                     set trapped access
       mov  =b$vrv,vrsto(xl)                     set trapped store
       exi                                       return to caller
       enp                                       end procedure inout
```

if **.cnbf**

else

```
    insbf -- insert string in buffer
    this routine will replace a section of a buffer with the
    contents of a given string.  if the length of the
    section to be replaced is different than the length of
    the given string, and the replacement is not an append,
    then the upper section of the buffer is shifted up or
    down to create the proper space for the insert.
    (xr)                 pointer to bcblk
    (xl)                 object which is string convertable
    (wa)                 offset of start of insert in buffer
    (wb)                 length of section to replace
    jsr  insbf           call to insert characters in buffer
    ppm  loc             thread if (xl) not convertable
    ppm  loc             thread if insert not possible
    the second alternate exit is taken if the insert would
    overflow the buffer, or if the insert is out past the
    defined end of the buffer as given.
insbf   prc  e,2                        entry point
        mov  wa,inssa                   save entry wa
        mov  wb,inssb                   save entry wb
        mov  wc,inssc                   save entry wc
        add  wb,wa                      add to get offset past replace part
        mov  wa,insab                   save wa+wb
        mov  bclen(xr),wc               get current defined length
        bgt  inssa,wc,ins07             fail if start offset too big
        bgt  wa,wc,ins07                fail if final offset too big
        mov  xl,-(xs)                   save entry xl
        mov  xr,-(xs)                   save bcblk ptr
        mov  xl,-(xs)                   stack again for gtstg or gtstb
        beq  xr,xl,ins08                b if inserting same buffer
        jsr  gtstb                      call to get string or buffer
        ppm  ins05                      take string convert err exit
    merge here with xr pointing to the scblk or bfblk of
    the object being inserted, and wa containing the
    number of characters in that object.
ins09   mov  xr,xl                      save string ptr
        mov  wa,insln                   save its length
        mov  (xs),xr                    restore bcblk ptr
        add  wc,wa                      add buffer len to string len
        sub  inssb,wa                   bias out component being replaced
        mov  bcbuf(xr),xr               point to bfblk
        bgt  wa,bfalc(xr),ins06         fail if result exceeds allocation
        mov  (xs),xr                    restore bcblk ptr
        mov  wc,wa                      get buffer length
        sub  insab,wa                   subtract to get shift length
        add  insln,wc                   add length of new
        sub  inssb,wc                   subtract old to get total new len
        mov  bclen(xr),wb               get old bclen
        mov  wc,bclen(xr)               stuff new length
        bze  wa,ins04                   skip shift if nothing to do
```

```
beq   inssb,insln,ins04        skip shift if lengths match
mov   bcbuf(xr),xr             point to bfblk
mov   xl,-(xs)                 save scblk ptr
blo   inssb,insln,ins01        brn if shift is for more room
```

```
    insbf (continued)
    we are shifting the upper segment down to compact
    the buffer.  (the string length is smaller than the
    segment being replaced.)  registers are set as
    (wa)                 move (shift down) length
    (wb)                 old bclen
    (wc)                 new bclen
    (xr)                 bfblk ptr
    (xl),(xs)            scblk or bfblk ptr
        mov  inssa,wb                       get offset to insert
        add  insln,wb                       add insert length to get dest off
        mov  xr,xl                          make copy
        plc  xl,insab                       prepare source for move
        psc  xr,wb                          prepare destination reg for move
        mvc                                 move em out
        brn  ins02                          branch to pad
    we are shifting the upper segment up to expand
    the buffer.  (the string length is larger than the
    segment being replaced.)
ins01   mov  xr,xl                          copy bfblk ptr
        plc  xl,wb                          set source reg for move backwards
        psc  xr,wc                          set destination ptr for move
        mcb                                 move backwards (possible overlap)
    merge here after move to adjust padding at new buffer end
ins02   mov  (xs)+,xl                       restore scblk or bfblk ptr
        mov  wc,wa                          copy new buffer end
        ctb  wa,0                           round out
        sub  wc,wa                          subtract to get remainder
        bze  wa,ins04                       no pad if already even boundary
        mov  (xs),xr                        get bcblk ptr
        mov  bcbuf(xr),xr                   get bfblk ptr
        psc  xr,wc                          prepare to pad
        zer  wb                             clear wb
        lct  wa,wa                          load loop count
    loop here to stuff pad characters
ins03   sch  wb,(xr)+                       stuff zero pad
        bct  wa,ins03                       branch for more
        csc  xr                             complete store character
```

```
    insbf (continued)
    merge here when padding ok.  now copy in the insert
    string to the hole.
ins04   mov  insln,wa                          get insert length
        bze  wa,ins4b                          if nothing to insert
        mov  (xs),xr                           get bcblk ptr
        mov  bcbuf(xr),xr                      get bfblk ptr
        plc  xl                                prepare to copy from first char
        psc  xr,inssa                          prepare to store in hole
        mvc                                    copy the characters
    continue here after possible insertion copy
ins4b   mov  (xs)+,xr                          restore entry xr
        mov  (xs)+,xl                          restore entry xl
        mov  inssa,wa                          restore entry wa
        mov  inssb,wb                          restore entry wb
        mov  inssc,wc                          restore entry wc
        exi                                    return to caller
    here to take string convert error exit
ins05   mov  (xs)+,xr                          restore entry xr
        mov  (xs)+,xl                          restore entry xl
        mov  inssa,wa                          restore entry wa
        mov  inssb,wb                          restore entry wb
        mov  inssc,wc                          restore entry wc
        exi  1                                 alternate exit
    here for invalid offset or length
ins06   mov  (xs)+,xr                          restore entry xr
        mov  (xs)+,xl                          restore entry xl
    merge for length failure exit with stack set
ins07   mov  inssa,wa                          restore entry wa
        mov  inssb,wb                          restore entry wb
        mov  inssc,wc                          restore entry wc
        exi  2                                 alternate exit
    here if inserting the same buffer into itself.  have
    to convert the inserted buffer to an intermediate
    string to prevent garbled data.
ins08   jsr  gtstg                             call to get string
        ppm  ins05                             take string convert err exit
        brn  ins09                             merge back to perform insertion
        enp                                    end procedure insbf
```

fi

```
    insta - used to initialize structures in static region
    (xr)                  pointer to starting static location
    jsr  insta            call to initialize static structure
    (xr)                  ptr to next free static location
    (wa,wb,wc)            destroyed
    note that this procedure establishes the pointers
    prbuf, gtswk, and kvalp.
insta   prc   e,0                          entry point
    initialize print buffer with blank words
        mov   prlen,wc                     no. of chars in print bfr
        mov   xr,prbuf                     print bfr is put at static start
        mov   =b$scl,(xr)+                 store string type code
        mov   wc,(xr)+                     and string length
        ctw   wc,0                         get number of words in buffer
        mov   wc,prlnw                     store for buffer clear
        lct   wc,wc                        words to clear
    loop to clear buffer
inst1   mov   nullw,(xr)+                  store blank
        bct   wc,inst1                     loop
    allocate work area for gtstg conversion procedure
        mov   =nstmx,wa                    get max num chars in output number
        ctb   wa,scsi$                     no of bytes needed
        mov   xr,gtswk                     store bfr adrs
        add   wa,xr                        bump for work bfr
    build alphabet string for alphabet keyword and replace
        mov   xr,kvalp                     save alphabet pointer
        mov   =b$scl,(xr)                  string blk type
        mov   =cfp$a,wc                    no of chars in alphabet
        mov   wc,sclen(xr)                 store as string length
        mov   wc,wb                        copy char count
        ctb   wb,scsi$                     no. of bytes needed
        add   xr,wb                        current end address for static
        mov   wb,wa                        save adrs past alphabet string
        lct   wc,wc                        loop counter
        psc   xr                           point to chars of string
        zer   wb                           set initial character value
    loop to enter character codes in order
inst2   sch   wb,(xr)+                     store next code
        icv   wb                           bump code value
        bct   wc,inst2                     loop till all stored
        csc   xr                           complete store characters
        mov   wa,xr                        return current static ptr
        exi                                return to caller
        enp                                end procedure insta
```

```
    iofcb -- get input/output fcblk pointer
    used by endfile, eject and rewind to find the fcblk
    (if any) corresponding to their argument.
    -(xs)                 argument
    jsr  iofcb            call to find fcblk
    ppm  loc              arg is an unsuitable name
    ppm  loc              arg is null string
    ppm  loc              arg file not found
    (xs)                  popped
    (xl)                  ptr to filearg1 vrblk
    (xr)                  argument
    (wa)                  fcblk ptr or 0
    (wb,wc)               destroyed
iofcb   prc  n,3                                  entry point
        jsr  gtstg                                get arg as string
        ppm  iofc2                                fail
        mov  xr,xl                                copy string ptr
        jsr  gtnvr                                get as natural variable
        ppm  iofc3                                fail if null
        mov  xl,wb                                copy string pointer again
        mov  xr,xl                                copy vrblk ptr for return
        zer  wa                                   in case no trblk found
    loop to find file arg1 trblk
iofc1   mov  vrval(xr),xr                         get possible trblk ptr
        bne  (xr),=b$trt,iofc4                    fail if end of chain
        bne  trtyp(xr),=trtfc,iofc1               loop if not file arg trblk
        mov  trfpt(xr),wa                         get fcblk ptr
        mov  wb,xr                                copy arg
        exi                                       return
    fail return
iofc2   exi  1                                    fail
    null arg
iofc3   exi  2                                    null arg return
    file not found
iofc4   exi  3                                    file not found return
        enp                                       end procedure iofcb
```

```
    ioppf -- process filearg2 for ioput
    (r$xsc)              filearg2 ptr
    jsr  ioppf           call to process filearg2
    (xl)                 filearg1 ptr
    (xr)                 file arg2 ptr
    -(xs)...-(xs)        fields extracted from filearg2
    (wc)                 no. of fields extracted
    (wb)                 input/output flag
    (wa)                 fcblk ptr or 0
ioppf   prc  n,0                                   entry point
        zer  wb                                    to count fields extracted
    loop to extract fields
iopp1   mov  =iodel,xl                             get delimiter
        mov  xl,wc                                 copy it
        zer  wa                                    retain leading blanks in filearg2
        jsr  xscan                                 get next field
        mov  xr,-(xs)                              stack it
        icv  wb                                    increment count
        bnz  wa,iopp1                              loop
        mov  wb,wc                                 count of fields
        mov  ioptt,wb                              i/o marker
        mov  r$iof,wa                              fcblk ptr or 0
        mov  r$io2,xr                              file arg2 ptr
        mov  r$io1,xl                              filearg1
        exi                                        return
        enp                                        end procedure ioppf
```

ioput -- routine used by input and output
ioput sets up input/output associations. it builds
such trace and file control blocks as are necessary and
calls sysfc,sysio to perform checks on the
arguments and to open the files.

```
   +-----------+   +---------------+       +-----------+
+-.i           i   i               i------.i   =b$xrt  i
i  +-----------+   +---------------+       +-----------+
i  /           /       (r$fcb)             i    *4     i
i  /           /                           +-----------+
i  +-----------+   +---------------+       i           i-
i  i   name    +--.i    =b$trt     i       +-----------+
i  /           /   +---------------+       i           i
i   (first arg)    i =trtin/=trtou i       +-----------+
i                  +---------------+            i
i                  i     value     i            i
i                  +---------------+            i
i                  i(trtrf) 0   or i--+         i
i                  +---------------+  i         i
i                  i(trfpt) 0   or i----+       i
i                  +---------------+  i i       i
i                     (i/o trblk)     i i       i
i  +-----------+                      i i       i
i  i           i                      i i       i
i  +-----------+                      i i       i
i  i           i                      i i       i
i  +-----------+   +---------------+  i i       i
i  i           +--.i    =b$trt     i.-+ i       i
i  +-----------+   +---------------+    i       i
i  /           /   i    =trtfc     i    i       i
i  /           /   +---------------+    i       i
i    (filearg1     i     value     i    i       i
i         vrblk)   +---------------+    i       i
i                  i(trtrf) 0   or i--+ i       .
i                  +---------------+  i .  +-----------+
i                  i(trfpt) 0   or i------./   fcblk   /
i                  +---------------+  i    +-----------+
i                      (trtrf)        i
i                                     i
i                                     i
i                  +---------------+  i
i                  i    =b$xrt     i.-+
i                  +---------------+
i                  i      *5       i
i                  +---------------+
+------------------i               i
                   +---------------+       +-----------+
                   i(trtrf) o   or i------.i  =b$xrt   i
                   +---------------+       +-----------+
                   i  name offset  i       i    etc    i
                   +---------------+
                     (iochn - chain of name pointers)
```

```
    ioput (continued)
    no additional trap blocks are used for standard input/out
    files. otherwise an i/o trap block is attached to second
    arg (filearg1) vrblk. see diagram above for details of
    the structure built.
    -(xs)                 1st arg (vbl to be associated)
    -(xs)                 2nd arg (file arg1)
    -(xs)                 3rd arg (file arg2)
    (wb)                  0 for input, 3 for output assoc.
    jsr  ioput            call for input/output association
    ppm  loc              3rd arg not a string
    ppm  loc              2nd arg not a suitable name
    ppm  loc              1st arg not a suitable name
    ppm  loc              inappropriate file spec for i/o
    ppm  loc              i/o file does not exist
    ppm  loc              i/o file cannot be read/written
    ppm  loc              i/o fcblk currently in use
    (xs)                  popped
    (xl,xr,wa,wb,wc)      destroyed
ioput   prc  n,7                                  entry point
        zer  r$iot                                in case no trtrf block used
        zer  r$iof                                in case no fcblk alocated
        zer  r$iop                                in case sysio fails
        mov  wb,ioptt                             store i/o trace type
        jsr  xscni                                prepare to scan filearg2
        ppm  iop13                                fail
        ppm  iopa0                                null file arg2
iopa0   mov  xr,r$io2                             keep file arg2
        mov  wa,xl                                copy length
        jsr  gtstg                                convert filearg1 to string
        ppm  iop14                                fail
        mov  xr,r$io1                             keep filearg1 ptr
        jsr  gtnvr                                convert to natural variable
        ppm  iop00                                jump if null
        brn  iop04                                jump to process non-null args
    null filearg1
iop00   bze  xl,iop01                             skip if both args null
        jsr  ioppf                                process filearg2
        jsr  sysfc                                call for filearg2 check
        ppm  iop16                                fail
        ppm  iop26                                fail
        brn  iop11                                complete file association
```

```
    ioput (continued)
    here with 0 or fcblk ptr in (xl)
iop01   mov  ioptt,wb                         get trace type
        mov  r$iot,xr                         get 0 or trtrf ptr
        jsr  trbld                            build trblk
        mov  xr,wc                            copy trblk pointer
        mov  (xs)+,xr                         get variable from stack
        mov  wc,-(xs)                         make trblk collectable
        jsr  gtvar                            point to variable
        ppm  iop15                            fail
        mov  (xs)+,wc                         recover trblk pointer
        mov  xl,r$ion                         save name pointer
        mov  xl,xr                            copy name pointer
        add  wa,xr                            point to variable
        sub  *vrval,xr                        subtract offset,merge into loop
    loop to end of trblk chain if any
iop02   mov  xr,xl                            copy blk ptr
        mov  vrval(xr),xr                     load ptr to next trblk
        bne  (xr),=b$trt,iop03                jump if not trapped
        bne  trtyp(xr),ioptt,iop02            loop if not same assocn
        mov  trnxt(xr),xr                     get value and delete old trblk
    ioput (continued)
    store new association
iop03   mov  wc,vrval(xl)                     link to this trblk
        mov  wc,xl                            copy pointer
        mov  xr,trnxt(xl)                     store value in trblk
        mov  r$ion,xr                         restore possible vrblk pointer
        mov  wa,wb                            keep offset to name
        jsr  setvr                            if vrblk, set vrget,vrsto
        mov  r$iot,xr                         get 0 or trtrf ptr
        bnz  xr,iop19                         jump if trtrf block exists
        exi                                   return to caller
    non standard file
    see if an fcblk has already been allocated.
iop04   zer  wa                               in case no fcblk found
```

```
    ioput (continued)
    search possible trblk chain to pick up the fcblk
iop05   mov  xr,wb                          remember blk ptr
        mov  vrval(xr),xr                   chain along
        bne  (xr),=b$trt,iop06              jump if end of trblk chain
        bne  trtyp(xr),=trtfc,iop05         loop if more to go
        mov  xr,r$iot                       point to file arg1 trblk
        mov  trfpt(xr),wa                   get fcblk ptr from trblk
    wa = 0 or fcblk ptr
    wb = ptr to preceding blk to which any trtrf block
         for file arg1 must be chained.
iop06   mov  wa,r$iof                       keep possible fcblk ptr
        mov  wb,r$iop                       keep preceding blk ptr
        jsr  ioppf                          process filearg2
        jsr  sysfc                          see if fcblk required
        ppm  iop16                          fail
        ppm  iop26                          fail
        bze  wa,iop12                       skip if no new fcblk wanted
        blt  wc,=num02,iop6a                jump if fcblk in dynamic
        jsr  alost                          get it in static
        brn  iop6b                          skip
    obtain fcblk in dynamic
iop6a   jsr  alloc                          get space for fcblk
    merge
iop6b   mov  xr,xl                          point to fcblk
        mov  wa,wb                          copy its length
        btw  wb                             get count as words (sgd apr80)
        lct  wb,wb                          loop counter
    clear fcblk
iop07   zer  (xr)+                          clear a word
        bct  wb,iop07                       loop
        beq  wc,=num02,iop09                skip if in static - dont set fields
        mov  =b$xnt,(xl)                    store xnblk code in case
        mov  wa,num01(xl)                   store length
        bnz  wc,iop09                       jump if xnblk wanted
        mov  =b$xrt,(xl)                    xrblk code requested
```

```
    ioput (continued)
    complete fcblk initialisation
iop09   mov  r$iot,xr                         get possible trblk ptr
        mov  xl,r$iof                         store fcblk ptr
        bnz  xr,iop10                         jump if trblk already found
    a new trblk is needed
        mov  =trtfc,wb                        trtyp for fcblk trap blk
        jsr  trbld                            make the block
        mov  xr,r$iot                         copy trtrf ptr
        mov  r$iop,xl                         point to preceding blk
        mov  vrval(xl),vrval(xr)              copy value field to trblk
        mov  xr,vrval(xl)                     link new trblk into chain
        mov  xl,xr                            point to predecessor blk
        jsr  setvr                            set trace intercepts
        mov  vrval(xr),xr                     recover trblk ptr
        brn  iop1a                            store fcblk ptr
    here if existing trblk
iop10   zer  r$iop                            do not release if sysio fails
    xr is ptr to trblk, xl is fcblk ptr or 0
iop1a   mov  r$iof,trfpt(xr)                  store fcblk ptr
    call sysio to complete file accessing
iop11   mov  r$iof,wa                         copy fcblk ptr or 0
        mov  ioptt,wb                         get input/output flag
        mov  r$io2,xr                         get file arg2
        mov  r$io1,xl                         get file arg1
        jsr  sysio                            associate to the file
        ppm  iop17                            fail
        ppm  iop18                            fail
        bnz  r$iot,iop01                      not std input if non-null trtrf blk
        bnz  ioptt,iop01                      jump if output
        bze  wc,iop01                         no change to standard read length
        mov  wc,cswin                         store new read length for std file
        brn  iop01                            merge to finish the task
    sysfc may have returned a pointer to a private fcblk
iop12   bnz  xl,iop09                         jump if private fcblk
        brn  iop11                            finish the association
    failure returns
iop13   exi  1                                3rd arg not a string
iop14   exi  2                                2nd arg unsuitable
iop15   ica  xs                               discard trblk pointer
        exi  3                                1st arg unsuitable
iop16   exi  4                                file spec wrong
iop26   exi  7                                fcblk in use
    i/o file does not exist
iop17   mov  r$iop,xr                         is there a trblk to release
        bze  xr,iopa7                         if not
        mov  vrval(xr),xl                     point to trblk
        mov  vrval(xl),vrval(xr)              unsplice it
        jsr  setvr                            adjust trace intercepts
iopa7   exi  5                                i/o file does not exist
    i/o file cannot be read/written
iop18   mov  r$iop,xr                         is there a trblk to release
        bze  xr,iopa7                         if not
```

```
        mov  vrval(xr),xl              point to trblk
        mov  vrval(xl),vrval(xr)       unsplice it
        jsr  setvr                     adjust trace intercepts
iopa8   exi  6                         i/o file cannot be read/written
```

```
    ioput (continued)
    add to iochn chain of associated variables unless
    already present.
iop19   mov  r$ion,wc                         wc = name base, wb = name offset
    search loop
iop20   mov  trtrf(xr),xr                     next link of chain
        bze  xr,iop21                         not found
        bne  wc,ionmb(xr),iop20               no match
        beq  wb,ionmo(xr),iop22               exit if matched
        brn  iop20                            loop
    not found
iop21   mov  *num05,wa                        space needed
        jsr  alloc                            get it
        mov  =b$xrt,(xr)                      store xrblk code
        mov  wa,num01(xr)                     store length
        mov  wc,ionmb(xr)                     store name base
        mov  wb,ionmo(xr)                     store name offset
        mov  r$iot,xl                         point to trtrf blk
        mov  trtrf(xl),wa                     get ptr field contents
        mov  xr,trtrf(xl)                     store ptr to new block
        mov  wa,trtrf(xr)                     complete the linking
    insert fcblk on fcblk chain for sysej, sysxi
iop22   bze  r$iof,iop25                      skip if no fcblk
        mov  r$fcb,xl                         ptr to head of existing chain
    see if fcblk already on chain
iop23   bze  xl,iop24                         not on if end of chain
        beq  num03(xl),r$iof,iop25            dont duplicate if find it
        mov  num02(xl),xl                     get next link
        brn  iop23                            loop
    not found so add an entry for this fcblk
iop24   mov  *num04,wa                        space needed
        jsr  alloc                            get it
        mov  =b$xrt,(xr)                      store block code
        mov  wa,num01(xr)                     store length
        mov  r$fcb,num02(xr)                  store previous link in this node
        mov  r$iof,num03(xr)                  store fcblk ptr
        mov  xr,r$fcb                         insert node into fcblk chain
    return
iop25   exi                                   return to caller
        enp                                   end procedure ioput
```

```
    ktrex -- execute keyword trace
    ktrex is used to execute a possible keyword trace. it
    includes the test on trace and tests for trace active.
    (xl)                  ptr to trblk (or 0 if untraced)
    jsr  ktrex            call to execute keyword trace
    (xl,wa,wb,wc)         destroyed
    (ra)                  destroyed
ktrex   prc  r,0                          entry point (recursive)
        bze  xl,ktrx3                     immediate exit if keyword untraced
        bze  kvtra,ktrx3                  immediate exit if trace = 0
        dcv  kvtra                        else decrement trace
        mov  xr,-(xs)                     save xr
        mov  xl,xr                        copy trblk pointer
        mov  trkvr(xr),xl                 load vrblk pointer (nmbas)
        mov  *vrval,wa                    set name offset
        bze  trfnc(xr),ktrx1              jump if print trace
        jsr  trxeq                        else execute full trace
        brn  ktrx2                        and jump to exit
    here for print trace
ktrx1   mov  xl,-(xs)                     stack vrblk ptr for kwnam
        mov  wa,-(xs)                     stack offset for kwnam
        jsr  prtsn                        print statement number
        mov  =ch$am,wa                    load ampersand
        jsr  prtch                        print ampersand
        jsr  prtnm                        print keyword name
        mov  =tmbeb,xr                    point to blank-equal-blank
        jsr  prtst                        print blank-equal-blank
        jsr  kwnam                        get keyword pseudo-variable name
        mov  xr,dnamp                     reset ptr to delete kvblk
        jsr  acess                        get keyword value
        ppm                               failure is impossible
        jsr  prtvl                        print keyword value
        jsr  prtnl                        terminate print line
    here to exit after completing trace
ktrx2   mov  (xs)+,xr                     restore entry xr
    merge here to exit if no trace required
ktrx3   exi                               return to ktrex caller
        enp                               end procedure ktrex
```

```
    kwnam -- get pseudo-variable name for keyword
    1(xs)                 name base for vrblk
    0(xs)                 offset (should be *vrval)
    jsr  kwnam            call to get pseudo-variable name
    (xs)                  popped twice
    (xl,wa)               resulting pseudo-variable name
    (xr,wa,wb)            destroyed
kwnam  prc  n,0                               entry point
       ica  xs                                ignore name offset
       mov  (xs)+,xr                          load name base
       bge  xr,state,kwnm1                    jump if not natural variable name
       bnz  vrlen(xr),kwnm1                   error if not system variable
       mov  vrsvp(xr),xr                      else point to svblk
       mov  svbit(xr),wa                      load bit mask
       anb  btknm,wa                          and with keyword bit
       zrb  wa,kwnm1                          error if no keyword association
       mov  svlen(xr),wa                      else load name length in characters
       ctb  wa,svchs                          compute offset to field we want
       add  wa,xr                             point to svknm field
       mov  (xr),wb                           load svknm value
       mov  *kvsi$,wa                         set size of kvblk
       jsr  alloc                             allocate kvblk
       mov  =b$kvt,(xr)                       store type word
       mov  wb,kvnum(xr)                      store keyword number
       mov  =trbkv,kvvar(xr)                  set dummy trblk pointer
       mov  xr,xl                             copy kvblk pointer
       mov  *kvvar,wa                         set proper offset
       exi                                    return to kvnam caller
    here if not keyword name
kwnm1  erb  251,keyword operand               is not name of defined keyword
       enp                                    end procedure kwnam
```

```
    lcomp-- compare two strings lexically
    1(xs)                 first argument
    0(xs)                 second argument
    jsr  lcomp            call to compare aruments
    ppm  loc              transfer loc for arg1 not string
    ppm  loc              transfer loc for arg2 not string
    ppm  loc              transfer loc if arg1 llt arg2
    ppm  loc              transfer loc if arg1 leq arg2
    ppm  loc              transfer loc if arg1 lgt arg2
    (the normal return is never taken)
    (xs)                  popped twice
    (xr,xl)               destroyed
    (wa,wb,wc,ra)         destroyed
```

lcomp	**prc**	n,5	entry point

---

if **.cnbf**

	jsr	gtstg	convert second arg to string
else			
	jsr	gtstb	get second arg as string or buffer
fi			
	ppm	lcmp6	jump if second arg not string
	mov	xr,xl	else save pointer
	mov	wa,wc	and length

---

if **.cnbf**

	jsr	gtstg	convert first argument to string
else			
	jsr	gtstb	get first arg as string or buffer
fi			
	ppm	lcmp5	jump if not string
	mov	wa,wb	save arg 1 length
	plc	xr	point to chars of arg 1
	plc	xl	point to chars of arg 2

---

if **.ccmc**

	mov	wc,wa	arg 2 length to wa
	jsr	syscm	compare (xl,wa=arg2 xr,wb=arg1)
	err	283,string length	exceeded for generalized lexical comparison
	ppm	lcmp4	arg 2 lt arg 1, lgt exit
	ppm	lcmp3	arg 2 gt arg 1, llt exit
	exi	4	else identical strings, leq exit

```
    lcomp (continued)
else
        blo   wa,wc,lcmp1        jump if arg 1 length is smaller
        mov   wc,wa              else set arg 2 length as smaller
    here with smaller length in (wa)
lcmp1   bze   wa,lcmp7           if null string, compare lengths
        cmc   lcmp4,lcmp3        compare strings, jump if unequal
lcmp7   bne   wb,wc,lcmp2        if equal, jump if lengths unequal
        exi   4                  else identical strings, leq exit
```

```
    lcomp (continued)
    here if initial strings identical, but lengths unequal
lcmp2   bhi   wb,wc,lcmp4                 jump if arg 1 length gt arg 2 leng
fi
    here if first arg llt second arg
lcmp3   exi   3                           take llt exit
    here if first arg lgt second arg
lcmp4   exi   5                           take lgt exit
    here if first arg is not a string
lcmp5   exi   1                           take bad first arg exit
    here for second arg not a string
lcmp6   exi   2                           take bad second arg error exit
        enp                               end procedure lcomp
```

```
    listr -- list source line
    listr is used to list a source line during the initial
    compilation. it is called from scane and scanl.
    jsr  listr           call to list line
    (xr,xl,wa,wb,wc)     destroyed
    global locations used by listr
    cnttl                flag for -title, -stitl
    erlst                if listing on account of an error
```

---

if **.cinc**

```
    lstid                include depth of current image
```

fi

```
    lstlc                count lines on current page
    lstnp                max number of lines/page
    lstpf                set non-zero if the current source
                         line has been listed, else zero.
    lstpg                compiler listing page number
    lstsn                set if stmnt num to be listed
    r$cim                pointer to current input line.
    r$ttl                title for source listing
    r$stl                ptr to sub-title string
    entry point
listr  prc  e,0                      entry point
       bnz  cnttl,list5              jump if -title or -stitl
       bnz  lstpf,list4              immediate exit if already listed
       bge  lstlc,lstnp,list6        jump if no room
    here after printing title (if needed)
list0  mov  r$cim,xr                 load pointer to current image
       bze  xr,list4                 jump if no image to print
       plc  xr                       point to characters
       lch  wa,(xr)                  load first character
       mov  lstsn,xr                 load statement number
       bze  xr,list2                 jump if no statement number
       mti  xr                       else get stmnt number as integer
       bne  stage,=stgic,list1       skip if execute time
       beq  wa,=ch$as,list2          no stmnt number list if comment
       beq  wa,=ch$mn,list2          no stmnt no. if control card
    print statement number
list1  jsr  prtin                    else print statement number
       zer  lstsn                    and clear for next time in
```

---

if **.cinc**

```
    here to test for printing include depth
list2  mov  lstid,xr                 include depth of image
       bze  xr,list8                 if not from an include file
       mov  =stnpd,wa                position for start of statement
       sub  =num03,wa                position to place include depth
       mov  wa,profs                 set as starting position
       mti  xr                       include depth as integer
       jsr  prtin                    print include depth
```

listr (continued)

here after printing statement number and include depth

```
list8   mov  =stnpd,profs                     point past statement number
```

else

```
    listr (continued)
    merge here after printing statement number (if required)
list2   mov  =stnpd,profs                     point past statement number
fi
        mov  r$cim,xr                         load pointer to current image
        jsr  prtst                            print it
        icv  lstlc                            bump line counter
        bnz  erlst,list3                      jump if error copy to int.ch.
        jsr  prtnl                            terminate line
        bze  cswdb,list3                      jump if -single mode
        jsr  prtnl                            else add a blank line
        icv  lstlc                            and bump line counter
    here after printing source image
list3   mnz  lstpf                            set flag for line printed
    merge here to exit
list4   exi                                   return to listr caller
    print title after -title or -stitl card
list5   zer  cnttl                            clear flag
    eject to new page and list title
list6   jsr  prtps                            eject
        bze  prich,list7                      skip if listing to regular printer
        beq  r$ttl,=nulls,list0               terminal listing omits null title
    list title
list7   jsr  listt                            list title
        brn  list0                            merge
        enp                                   end procedure listr
```

```
    listt -- list title and subtitle
    used during compilation to print page heading
    jsr  listt            call to list title
    (xr,wa)               destroyed
listt   prc   e,0                                     entry point
        mov   r$ttl,xr                                point to source listing title
        jsr   prtst                                   print title
        mov   lstpo,profs                             set offset
        mov   =lstms,xr                               set page message
        jsr   prtst                                   print page message
        icv   lstpg                                   bump page number
        mti   lstpg                                   load page number as integer
        jsr   prtin                                   print page number
        jsr   prtnl                                   terminate title line
        add   =num02,lstlc                            count title line and blank line
    print sub-title (if any)
        mov   r$stl,xr                                load pointer to sub-title
        bze   xr,lstt1                                jump if no sub-title
        jsr   prtst                                   else print sub-title
        jsr   prtnl                                   terminate line
        icv   lstlc                                   bump line count
    return point
lstt1   jsr   prtnl                                   print a blank line
        exi                                           return to caller
        enp                                           end procedure listt
```

if **.csfn**

```
newfn -- record new source file name
newfn is used after switching to a new include file, or
after a -line statement which contains a file name.
(xr)                 file name scblk
jsr  newfn
(wa,wb,wc,xl,xr,ra)   destroyed
on return, the table that maps statement numbers to file
names has been updated to include this new file name and
the current statement number.  the entry is made only if
the file name had changed from its previous value.
newfn  prc  e,0                 entry point
       mov  xr,-(xs)            save new name
       mov  r$sfc,xl            load previous name
       jsr  ident               check for equality
       ppm  nwfn1               jump if identical
       mov  (xs)+,xr            different, restore name
       mov  xr,r$sfc            record current file name
       mov  cmpsn,wb            get current statement
       mti  wb                  convert to integer
       jsr  icbld               build icblk for stmt number
       mov  r$sfn,xl            file name table
       mnz  wb                  lookup statement number by name
       jsr  tfind               allocate new teblk
       ppm                      always possible to allocate block
       mov  r$sfc,teval(xl)     record file name as entry value
       exi  r$sfc,teval(xl)     record file name as entry value
ere if new name and old name identical
nwfn1  ica  xs                  pop stack
       exi  xs                  pop stack
```

fi

```
nexts -- acquire next source image
nexts is used to acquire the next source image at compile
time. it assumes that a prior call to readr has input
a line image (see procedure readr). before the current
image is finally lost it may be listed here.
jsr  nexts            call to acquire next input line
(xr,xl,wa,wb,wc)      destroyed
global values affected
```

---

if **.cinc**

```
lstid                 include depth of next image
```

fi

```
r$cni                 on input, next image. on
                      exit reset to zero
r$cim                 on exit, set to point to image
rdcln                 current ln set from next line num
scnil                 input image length on exit
scnse                 reset to zero on exit
lstpf                 set on exit if line is listed
nexts  prc  e,0                      entry point
       bze  cswls,nxts2              jump if -nolist
       mov  r$cim,xr                 point to image
       bze  xr,nxts2                 jump if no image
       plc  xr                       get char ptr
       lch  wa,(xr)                  get first char
       bne  wa,=ch$mn,nxts1          jump if not ctrl card
       bze  cswpr,nxts2              jump if -noprint
   here to call lister
nxts1  jsr  listr                    list line
   here after possible listing
nxts2  mov  r$cni,xr                 point to next image
       mov  xr,r$cim                 set as next image
       mov  rdnln,rdcln              set as current line number
```

---

if **.cinc**

```
       mov  cnind,lstid              set as current include depth
```

fi

```
       zer  r$cni                    clear next image pointer
       mov  sclen(xr),wa             get input image length
       mov  cswin,wb                 get max allowable length
       blo  wa,wb,nxts3              skip if not too long
       mov  wb,wa                    else truncate
   here with length in (wa)
nxts3  mov  wa,scnil                 use as record length
       zer  scnse                    reset scnse
       zer  lstpf                    set line not listed yet
       exi                           return to nexts caller
       enp                           end procedure nexts
```

```
    patin -- pattern construction for len,pos,rpos,tab,rtab
    these pattern types all generate a similar node type. so
    the construction code is shared. see functions section
    for actual entry points for these five functions.
    (wa)                  pcode for expression arg case
    (wb)                  pcode for integer arg case
    jsr  patin            call to build pattern node
    ppm  loc              transfer loc for not integer or exp
    ppm  loc              transfer loc for int out of range
    (xr)                  pointer to constructed node
    (xl,wa,wb,wc,ia)      destroyed
patin   prc  n,2                                   entry point
        mov  wa,xl                                 preserve expression arg pcode
        jsr  gtsmi                                 try to convert arg as small integer
        ppm  ptin2                                 jump if not integer
        ppm  ptin3                                 jump if out of range
    common successful exit point
ptin1   jsr  pbild                                 build pattern node
        exi                                        return to caller
    here if argument is not an integer
ptin2   mov  xl,wb                                 copy expr arg case pcode
        blo  (xr),=b$e$$,ptin1                     all ok if expression arg
        exi  1                                     else take error exit for wrong type
    here for error of out of range integer argument
ptin3   exi  2                                     take out-of-range error exit
        enp                                        end procedure patin
```

```
    patst -- pattern construction for any,notany,
             break,span and breakx pattern functions.
    these pattern functions build similar types of nodes and
    the construction code is shared. see functions section
    for actual entry points for these five pattern functions.
    0(xs)                 string argument
    (wb)                  pcode for one char argument
    (xl)                  pcode for multi-char argument
    (wc)                  pcode for expression argument
    jsr  patst            call to build node
    ppm  loc              if not string or expr (or null)
    (xs)                  popped past string argument
    (xr)                  pointer to constructed node
    (xl)                  destroyed
    (wa,wb,wc,ra)         destroyed
    note that there is a special call to patst in the evals
    procedure with a slightly different form. see evals
    for details of the form of this call.
patst   prc   n,1                               entry point
        jsr   gtstg                             convert argument as string
        ppm   pats7                             jump if not string
        bze   wa,pats7                          jump if null string (catspaw)
        bne   wa,=num01,pats2                   jump if not one char string
    here for one char string case
        bze   wb,pats2                          treat as multi-char if evals call
        plc   xr                                point to character
        lch   xr,(xr)                           load character
    common exit point after successful construction
pats1   jsr   pbild                             call routine to build node
        exi                                     return to patst caller
```

```
    patst (continued)
    here for multi-character string case
pats2  mov  xl,-(xs)            save multi-char pcode
       mov  ctmsk,wc            load current mask bit
       beq  xr,r$cts,pats6      jump if same as last string c3.738
       mov  xr,-(xs)            save string pointer
       lsh  wc,1                shift to next position
       nzb  wc,pats4            skip if position left in this tbl
    here we must allocate a new character table
       mov  *ctsi$,wa           set size of ctblk
       jsr  alloc               allocate ctblk
       mov  xr,r$ctp            store ptr to new ctblk
       mov  =b$ctt,(xr)+        store type code, bump ptr
       lct  wb,=cfp$a           set number of words to clear
       mov  bits0,wc            load all zero bits
    loop to clear all bits in table to zeros
pats3  mov  wc,(xr)+            move word of zero bits
       bct  wb,pats3            loop till all cleared
       mov  bits1,wc            set initial bit position
    merge here with bit position available
pats4  mov  wc,ctmsk            save parm2 (new bit position)
       mov  (xs)+,xl            restore pointer to argument string
       mov  xl,r$cts            save for next time c3.738
       mov  sclen(xl),wb        load string length
       bze  wb,pats6            jump if null string case
       lct  wb,wb               else set loop counter
       plc  xl                  point to characters in argument
```

```
    patst (continued)
    loop to set bits in column of table
pats5   lch  wa,(xl)+                 load next character
        wtb  wa                       convert to byte offset
        mov  r$ctp,xr                 point to ctblk
        add  wa,xr                    point to ctblk entry
        mov  wc,wa                    copy bit mask
        orb  ctchs(xr),wa             or in bits already set
        mov  wa,ctchs(xr)             store resulting bit string
        bct  wb,pats5                 loop till all bits set
    complete processing for multi-char string case
pats6   mov  r$ctp,xr                 load ctblk ptr as parm1 for pbild
        zer  xl                       clear garbage ptr in xl
        mov  (xs)+,wb                 load pcode for multi-char str case
        brn  pats1                    back to exit (wc=bitstring=parm2)
    here if argument is not a string
    note that the call from evals cannot pass an expression
    since evalp always reevaluates expressions.
pats7   mov  wc,wb                    set pcode for expression argument
        blo  (xr),=b$e$$,pats1        jump to exit if expression arg
        exi  1                        else take wrong type error exit
        enp                           end procedure patst
```

```
    pbild -- build pattern node
    (xr)                  parm1 (only if required)
    (wb)                  pcode for node
    (wc)                  parm2 (only if required)
    jsr  pbild            call to build node
    (xr)                  pointer to constructed node
    (wa)                  destroyed
pbild   prc  e,0                                  entry point
        mov  xr,-(xs)                             stack possible parm1
        mov  wb,xr                                copy pcode
        lei  xr                                   load entry point id (bl$px)
        beq  xr,=bl$p1,pbld1                      jump if one parameter
        beq  xr,=bl$p0,pbld3                      jump if no parameters
    here for two parameter case
        mov  *pcsi$,wa                            set size of p2blk
        jsr  alloc                                allocate block
        mov  wc,parm2(xr)                         store second parameter
        brn  pbld2                                merge with one parm case
    here for one parameter case
pbld1   mov  *pbsi$,wa                            set size of p1blk
        jsr  alloc                                allocate node
    merge here from two parm case
pbld2   mov  (xs),parm1(xr)                       store first parameter
        brn  pbld4                                merge with no parameter case
    here for case of no parameters
pbld3   mov  *pasi$,wa                            set size of p0blk
        jsr  alloc                                allocate node
    merge here from other cases
pbld4   mov  wb,(xr)                              store pcode
        ica  xs                                   pop first parameter
        mov  =ndnth,pthen(xr)                     set nothen successor pointer
        exi                                       return to pbild caller
        enp                                       end procedure pbild
```

```
    pconc -- concatenate two patterns
    (xl)                 ptr to right pattern
    (xr)                 ptr to left pattern
    jsr  pconc           call to concatenate patterns
    (xr)                 ptr to concatenated pattern
    (xl,wa,wb,wc)        destroyed
    to concatenate two patterns, all successors in the left
    pattern which point to the nothen node must be changed to
    point to the right pattern. however, this modification
    must be performed on a copy of the left argument rather
    than the left argument itself, since the left argument
    may be pointed to by some other variable value.
    accordingly, it is necessary to copy the left argument.
    this is not a trivial process since we must avoid copying
    nodes more than once and the pattern is a graph structure
    the following algorithm is employed.
    the stack is used to store a list of nodes which
    have already been copied. the format of the entries on
    this list consists of a two word block. the first word
    is the old address and the second word is the address
    of the copy. this list is searched by the pcopy
    routine to avoid making duplicate copies. a trick is
    used to accomplish the concatenation at the same time.
    a special entry is made to start with on the stack. this
    entry records that the nothen node has been copied
    already and the address of its copy is the right pattern.
    this automatically performs the correct replacements.
pconc  prc  e,0                    entry point
       zer  -(xs)                  make room for one entry at bottom
       mov  xs,wc                  store pointer to start of list
       mov  =ndnth,-(xs)           stack nothen node as old node
       mov  xl,-(xs)               store right arg as copy of nothen
       mov  xs,xt                  initialize pointer to stack entries
       jsr  pcopy                  copy first node of left arg
       mov  wa,num02(xt)           store as result under list
```

```
    pconc (continued)
    the following loop scans entries in the list and makes
    sure that their successors have been copied.
pcnc1   beq  xt,xs,pcnc2                        jump if all entries processed
        mov  -(xt),xr                           else load next old address
        mov  pthen(xr),xr                       load pointer to successor
        jsr  pcopy                              copy successor node
        mov  -(xt),xr                           load pointer to new node (copy)
        mov  wa,pthen(xr)                       store ptr to new successor
    now check for special case of alternation node where
    parm1 points to a node and must be copied like pthen.
        bne  (xr),=p$alt,pcnc1                  loop back if not
        mov  parm1(xr),xr                       else load pointer to alternative
        jsr  pcopy                              copy it
        mov  (xt),xr                            restore ptr to new node
        mov  wa,parm1(xr)                       store ptr to copied alternative
        brn  pcnc1                              loop back for next entry
    here at end of copy process
pcnc2   mov  wc,xs                              restore stack pointer
        mov  (xs)+,xr                           load pointer to copy
        exi                                     return to pconc caller
        enp                                     end procedure pconc
```

```
    pcopy -- copy a pattern node
    pcopy is called from the pconc procedure to copy a single
    pattern node. the copy is only carried out if the node
    has not been copied already.
    (xr)                   pointer to node to be copied
    (xt)                   ptr to current loc in copy list
    (wc)                   pointer to list of copied nodes
    jsr  pcopy             call to copy a node
    (wa)                   pointer to copy
    (wb,xr)                destroyed
pcopy   prc  n,0                                   entry point
        mov  xt,wb                                 save xt
        mov  wc,xt                                 point to start of list
    loop to search list of nodes copied already
pcop1   dca  xt                                    point to next entry on list
        beq  xr,(xt),pcop2                         jump if match
        dca  xt                                    else skip over copied address
        bne  xt,xs,pcop1                           loop back if more to test
    here if not in list, perform copy
        mov  (xr),wa                               load first word of block
        jsr  blkln                                 get length of block
        mov  xr,xl                                 save pointer to old node
        jsr  alloc                                 allocate space for copy
        mov  xl,-(xs)                              store old address on list
        mov  xr,-(xs)                              store new address on list
        chk                                        check for stack overflow
        mvw                                        move words from old block to copy
        mov  (xs),wa                               load pointer to copy
        brn  pcop3                                 jump to exit
    here if we find entry in list
pcop2   mov  -(xt),wa                              load address of copy from list
    common exit point
pcop3   mov  wb,xt                                 restore xt
        exi                                        return to pcopy caller
        enp                                        end procedure pcopy
```

if **.cnpf**

else

```
    prflr -- print profile
    prflr is called to print the contents of the profile
    table in a fairly readable tabular format.
    jsr  prflr            call to print profile
    (wa,ia)               destroyed
prflr   prc
        bze  pfdmp,prfl4                    no printing if no profiling done
        mov  xr,-(xs)                       preserve entry xr
        mov  wb,pfsvw                       and also wb
        jsr  prtpg                          eject
        mov  =pfms1,xr                      load msg /program profile/
        jsr  prtst                          and print it
        jsr  prtnl                          followed by newline
        jsr  prtnl                          and another
        mov  =pfms2,xr                      point to first hdr
        jsr  prtst                          print it
        jsr  prtnl                          new line
        mov  =pfms3,xr                      second hdr
        jsr  prtst                          print it
        jsr  prtnl                          new line
        jsr  prtnl                          and another blank line
        zer  wb                             initial stmt count
        mov  pftbl,xr                       point to table origin
        add  *xndta,xr                      bias past xnblk header (sgd07)
    loop here to print successive entries
prfl1   icv  wb                             bump stmt nr
        ldi  (xr)                           load nr of executions
        ieq  prfl3                          no printing if zero
        mov  =pfpd1,profs                   point where to print
        jsr  prtin                          and print it
        zer  profs                          back to start of line
        mti  wb                             load stmt nr
        jsr  prtin                          print it there
        mov  =pfpd2,profs                   and pad past count
        ldi  cfp$i(xr)                      load total exec time
        jsr  prtin                          print that too
        ldi  cfp$i(xr)                      reload time
        mli  intth                          convert to microsec
        iov  prfl2                          omit next bit if overflow
        dvi  (xr)                           divide by executions
        mov  =pfpd3,profs                   pad last print
        jsr  prtin                          and print mcsec/execn
    merge after printing time
prfl2   jsr  prtnl                          thats another line
    here to go to next entry
prfl3   add  *pf$i2,xr                      bump index ptr (sgd07)
        blt  wb,pfnte,prfl1                 loop if more stmts
        mov  (xs)+,xr                       restore callers xr
        mov  pfsvw,wb                       and wb too
    here to exit
```

`prfl4`	**exi**	return
	enp	end of prflr

```
    prflu -- update an entry in the profile table
    on entry, kvstn contains nr of stmt to profile
    jsr  prflu            call to update entry
    (ia)                  destroyed
prflu   prc
        bnz  pffnc,pflu4                          skip if just entered function
        mov  xr,-(xs)                             preserve entry xr
        mov  wa,pfsvw                             save wa (sgd07)
        bnz  pftbl,pflu2                          branch if table allocated
    here if space for profile table not yet allocated.
    calculate size needed, allocate a static xnblk, and
    initialize it all to zero.
    the time taken for this will be attributed to the current
    statement (assignment to keywd profile), but since the
    timing for this statement is up the pole anyway, this
    doesnt really matter...
        sub  =num01,pfnte                         adjust for extra count (sgd07)
        mti  pfi2a                                convrt entry size to int
        sti  pfste                                and store safely for later
        mti  pfnte                                load table length as integer
        mli  pfste                                multiply by entry size
        mfi  wa                                   get back address-style
        add  =num02,wa                            add on 2 word overhead
        wtb  wa                                   convert the whole lot to bytes
        jsr  alost                                gimme the space
        mov  xr,pftbl                             save block pointer
        mov  =b$xnt,(xr)+                         put block type and ...
        mov  wa,(xr)+                             ... length into header
        mfi  wa                                   get back nr of wds in data area
        lct  wa,wa                                load the counter
    loop here to zero the block data
pflu1   zer  (xr)+                                blank a word
        bct  wa,pflu1                             and alllllll the rest
    end of allocation. merge back into routine
pflu2   mti  kvstn                                load nr of stmt just ended
        sbi  intv1                                make into index offset
        mli  pfste                                make offset of table entry
        mfi  wa                                   convert to address
        wtb  wa                                   get as baus
        add  *num02,wa                            offset includes table header
        mov  pftbl,xr                             get table start
        bge  wa,num01(xr),pflu3                   if out of table, skip it
        add  wa,xr                                else point to entry
        ldi  (xr)                                 get nr of executions so far
        adi  intv1                                nudge up one
        sti  (xr)                                 and put back
        jsr  systm                                get time now
        sti  pfetm                                stash ending time
        sbi  pfstm                                subtract start time
        adi  cfp$i(xr)                            add cumulative time so far
        sti  cfp$i(xr)                            and put back new total
        ldi  pfetm                                load end time of this stmt ...
        sti  pfstm                                ... which is start time of next
```

```
    merge here to exit
pflu3   mov  (xs)+,xr                                 restore callers xr
        mov  pfsvw,wa                                 restore saved reg
        exi                                           and return
    here if profile is suppressed because a program defined
    function is about to be entered, and so the current stmt
    has not yet finished
pflu4   zer  pffnc                                    reset the condition flag
        exi                                           and immediate return
        enp                                           end of procedure prflu
```

ﬁ

```
    prpar - process print parameters
    (wc)                  if nonzero associate terminal only
    jsr  prpar            call to process print parameters
    (xl,xr,wa,wb,wc)      destroyed
    since memory allocation is undecided on initial call,
    terminal cannot be associated. the entry with wc non-zero
    is provided so a later call can be made to complete this.
prpar  prc  e,0                               entry point
       bnz  wc,prpa8                          jump to associate terminal
       jsr  syspp                             get print parameters
       bnz  wb,prpa1                          jump if lines/page specified
       mov  =cfp$m,wb                         else use a large value
       rsh  wb,1                              but not too large
    store line count/page
prpa1  mov  wb,lstnp                          store number of lines/page
       mov  wb,lstlc                          pretend page is full initially
       zer  lstpg                             clear page number
       mov  prlen,wb                          get prior length if any
       bze  wb,prpa2                          skip if no length
       bgt  wa,wb,prpa3                       skip storing if too big
    store print buffer length
prpa2  mov  wa,prlen                          store value
    process bits options
prpa3  mov  bits3,wb                          bit 3 mask
       anb  wc,wb                             get -nolist bit
       zrb  wb,prpa4                          skip if clear
       zer  cswls                             set -nolist
    check if fail reports goto interactive channel
prpa4  mov  bits1,wb                          bit 1 mask
       anb  wc,wb                             get bit
       mov  wb,erich                          store int. chan. error flag
       mov  bits2,wb                          bit 2 mask
       anb  wc,wb                             get bit
       mov  wb,prich                          flag for std printer on int. chan.
       mov  bits4,wb                          bit 4 mask
       anb  wc,wb                             get bit
       mov  wb,cpsts                          flag for compile stats suppressn.
       mov  bits5,wb                          bit 5 mask
       anb  wc,wb                             get bit
       mov  wb,exsts                          flag for exec stats suppression
```

```
    prpar (continued)
        mov  bits6,wb                 bit 6 mask
        anb  wc,wb                    get bit
        mov  wb,precl                 extended/compact listing flag
        sub  =num08,wa                point 8 chars from line end
        zrb  wb,prpa5                 jump if not extended
        mov  wa,lstpo                 store for listing page headings
     continue option processing
prpa5   mov  bits7,wb                 bit 7 mask
        anb  wc,wb                    get bit 7
        mov  wb,cswex                 set -noexecute if non-zero
        mov  bit10,wb                 bit 10 mask
        anb  wc,wb                    get bit 10
        mov  wb,headp                 pretend printed to omit headers
        mov  bits9,wb                 bit 9 mask
        anb  wc,wb                    get bit 9
        mov  wb,prsto                 keep it as std listing option
```

---

if **.culc**

```
        mov  wc,wb                    copy flags
        rsh  wb,12                    right justify bit 13
        anb  bits1,wb                 get bit
        mov  wb,kvcas                 set -case
```

fi

```
        mov  bit12,wb                 bit 12 mask
        anb  wc,wb                    get bit 12
        mov  wb,cswer                 keep it as errors/noerrors option
        zrb  wb,prpa6                 skip if clear
        mov  prlen,wa                 get print buffer length
        sub  =num08,wa                point 8 chars from line end
        mov  wa,lstpo                 store page offset
    check for -print/-noprint
prpa6   mov  bit11,wb                 bit 11 mask
        anb  wc,wb                    get bit 11
        mov  wb,cswpr                 set -print if non-zero
    check for terminal
        anb  bits8,wc                 see if terminal to be activated
        bnz  wc,prpa8                 jump if terminal required
        bze  initr,prpa9              jump if no terminal to detach
        mov  =v$ter,xl                ptr to /terminal/
        jsr  gtnvr                    get vrblk pointer
        ppm                           cant fail
        mov  =nulls,vrval(xr)         clear value of terminal
        jsr  setvr                    remove association
        brn  prpa9                    return
    associate terminal
prpa8   mnz  initr                    note terminal associated
        bze  dnamb,prpa9              cant if memory not organised
        mov  =v$ter,xl                point to terminal string
        mov  =trtou,wb                output trace type
        jsr  inout                    attach output trblk to vrblk
        mov  xr,-(xs)                 stack trblk ptr
        mov  =v$ter,xl                point to terminal string
        mov  =trtin,wb                input trace type
```

```
        jsr   inout                          attach input trace blk
        mov   (xs)+,vrval(xr)                add output trblk to chain
    return point
prpa9   exi                                  return
        enp                                  end procedure prpar
```

```
    prtch -- print a character
    prtch is used to print a single character
    (wa)                  character to be printed
    jsr  prtch            call to print character
prtch   prc   e,0                                   entry point
        mov   xr,-(xs)                              save xr
        bne   profs,prlen,prch1                     jump if room in buffer
        jsr   prtnl                                 else print this line
    here after making sure we have room
prch1   mov   prbuf,xr                              point to print buffer
        psc   xr,profs                              point to next character location
        sch   wa,(xr)                               store new character
        csc   xr                                    complete store characters
        icv   profs                                 bump pointer
        mov   (xs)+,xr                              restore entry xr
        exi                                         return to prtch caller
        enp                                         end procedure prtch
```

```
    prtic -- print to interactive channel
    prtic is called to print the contents of the standard
    print buffer to the interactive channel. it is only
    called after prtst has set up the string for printing.
    it does not clear the buffer.
    jsr  prtic            call for print
    (wa,wb)               destroyed
prtic   prc   e,0                                    entry point
        mov   xr,-(xs)                               save xr
        mov   prbuf,xr                               point to buffer
        mov   profs,wa                               no of chars
        jsr   syspi                                  print
        ppm   prtc2                                  fail return
    return
prtc1   mov   (xs)+,xr                               restore xr
        exi                                          return
    error occured
prtc2   zer   erich                                  prevent looping
        erb   252,error on printing                  to interactive channel
        brn   prtc1                                  return
        enp                                          procedure prtic
```

```
    prtis -- print to interactive and standard printer
    prtis puts a line from the print buffer onto the
    interactive channel (if any) and the standard printer.
    it always prints to the standard printer but does
    not duplicate lines if the standard printer is
    interactive.  it clears down the print buffer.
    jsr  prtis           call for printing
    (wa,wb)              destroyed
prtis   prc   e,0                        entry point
        bnz   prich,prts1                jump if standard printer is int.ch.
        bze   erich,prts1                skip if not doing int. error reps.
        jsr   prtic                      print to interactive channel
    merge and exit
prts1   jsr   prtnl                      print to standard printer
        exi                              return
        enp                              end procedure prtis
```

```
    prtin -- print an integer
    prtin prints the integer value which is in the integer
    accumulator. blocks built in dynamic storage
    during this process are immediately deleted.
    (ia)                  integer value to be printed
    jsr  prtin            call to print integer
    (ia,ra)               destroyed
prtin   prc   e,0                                 entry point
        mov   xr,-(xs)                            save xr
        jsr   icbld                               build integer block
        blo   xr,dnamb,prti1                      jump if icblk below dynamic
        bhi   xr,dnamp,prti1                      jump if above dynamic
        mov   xr,dnamp                            immediately delete it
    delete icblk from dynamic store
prti1   mov   xr,-(xs)                            stack ptr for gtstg
        jsr   gtstg                               convert to string
        ppm                                       convert error is impossible
        mov   xr,dnamp                            reset pointer to delete scblk
        jsr   prtst                               print integer string
        mov   (xs)+,xr                            restore entry xr
        exi                                       return to prtin caller
        enp                                       end procedure prtin
```

```
    prtmi -- print message and integer
    prtmi is used to print messages together with an integer
    value starting in column 15 (used by the routines at
    the end of compilation).
    jsr  prtmi            call to print message and integer
prtmi   prc  e,0                                  entry point
        jsr  prtst                                print string message
        mov  =prtmf,profs                         set column offset
        jsr  prtin                                print integer
        jsr  prtnl                                print line
        exi                                       return to prtmi caller
        enp                                       end procedure prtmi
```

```
    prtmm -- print memory used and available
    prtmm is used to provide memory usage information in
    both the end-of-compile and end-of-run statistics.
    jsr  prtmm            call to print memory stats
```

```
prtmm  prc
       mov  dnamp,wa                      next available loc
       sub  statb,wa                      minus start
```

---

if **.cbyt**
else

```
       btw  wa                            convert to words
```

fi

```
       mti  wa                            convert to integer
       mov  =encm1,xr                     point to /memory used (words)/
       jsr  prtmi                         print message
       mov  dname,wa                      end of memory
       sub  dnamp,wa                      minus next available loc
```

---

if **.cbyt**
else

```
       btw  wa                            convert to words
```

fi

```
       mti  wa                            convert to integer
       mov  =encm2,xr                     point to /memory available (words)/
       jsr  prtmi                         print line
       exi                                return to prtmm caller
       enp                                end of procedure prtmm
```

```
    prtmx  -- as prtmi with extra copy to interactive chan.
    jsr  prtmx            call for printing
    (wa,wb)               destroyed
prtmx   prc   e,0                                     entry point
        jsr   prtst                                   print string message
        mov   =prtmf,profs                            set column offset
        jsr   prtin                                   print integer
        jsr   prtis                                   print line
        exi                                           return
        enp                                           end procedure prtmx
```

```
    prtnl -- print new line (end print line)
    prtnl prints the contents of the print buffer, resets
    the buffer to all blanks and resets the print pointer.
    jsr  prtnl            call to print line
prtnl   prc   r,0                        entry point
        bnz   headp,prnl0                were headers printed
        jsr   prtps                      no - print them
    call syspr
prnl0   mov   xr,-(xs)                   save entry xr
        mov   wa,prtsa                   save wa
        mov   wb,prtsb                   save wb
        mov   prbuf,xr                   load pointer to buffer
        mov   profs,wa                   load number of chars in buffer
        jsr   syspr                      call system print routine
        ppm   prnl2                      jump if failed
        lct   wa,prlnw                   load length of buffer in words
        add   *schar,xr                  point to chars of buffer
        mov   nullw,wb                   get word of blanks
    loop to blank buffer
prnl1   mov   wb,(xr)+                   store word of blanks, bump ptr
        bct   wa,prnl1                   loop till all blanked
    exit point
        mov   prtsb,wb                   restore wb
        mov   prtsa,wa                   restore wa
        mov   (xs)+,xr                   restore entry xr
        zer   profs                      reset print buffer pointer
        exi                              return to prtnl caller
    file full or no output file for load module
prnl2   bnz   prtef,prnl3                jump if not first time
        mnz   prtef                      mark first occurrence
        erb   253,print limit exceeded   on standard output channel
    stop at once
prnl3   mov   =nini8,wb                  ending code
        mov   kvstn,wa                   statement number
        mov   r$fcb,xl                   get fcblk chain head
        jsr   sysej                      stop
        enp                              end procedure prtnl
```

```
    prtnm -- print variable name
    prtnm is used to print a character representation of the
    name of a variable (not a value of datatype name)
    names of pseudo-variables may not be passed to prtnm.
    (xl)                  name base
    (wa)                  name offset
    jsr  prtnm            call to print name
    (wb,wc,ra)            destroyed
prtnm  prc  r,0                            entry point (recursive, see prtvl)
       mov  wa,-(xs)                       save wa (offset is collectable)
       mov  xr,-(xs)                       save entry xr
       mov  xl,-(xs)                       save name base
       bhi  xl,state,prn02                 jump if not natural variable
    here for natural variable name, recognized by the fact
    that the name base points into the static area.
       mov  xl,xr                          point to vrblk
       jsr  prtvn                          print name of variable
    common exit point
prn01  mov  (xs)+,xl                       restore name base
       mov  (xs)+,xr                       restore entry value of xr
       mov  (xs)+,wa                       restore wa
       exi                                 return to prtnm caller
    here for case of non-natural variable
prn02  mov  wa,wb                          copy name offset
       bne  (xl),=b$pdt,prn03              jump if array or table
    for program defined datatype, prt fld name, left paren
       mov  pddfp(xl),xr                   load pointer to dfblk
       add  wa,xr                          add name offset
       mov  pdfof(xr),xr                   load vrblk pointer for field
       jsr  prtvn                          print field name
       mov  =ch$pp,wa                      load left paren
       jsr  prtch                          print character
```

```
    prtnm (continued)
    now we print an identifying name for the object if one
    can be found. the following code searches for a natural
    variable which contains this object as value. if such a
    variable is found, its name is printed, else the value
    of the object (as printed by prtvl) is used instead.
    first we point to the parent tbblk if this is the case of
    a table element. to do this, chase down the trnxt chain.
prn03  bne  (xl),=b$tet,prn04      jump if we got there (or not te)
       mov  tenxt(xl),xl           else move out on chain
       brn  prn03                  and loop back
    now we are ready for the search. to speed things up in
    the case of calls from dump where the same name base
    will occur repeatedly while dumping an array or table,
    we remember the last vrblk pointer found in prnmv. so
    first check to see if we have this one again.
prn04  mov  prnmv,xr               point to vrblk we found last time
       mov  hshtb,wa               point to hash table in case not
       brn  prn07                  jump into search for special check
    loop through hash slots
prn05  mov  wa,xr                  copy slot pointer
       ica  wa                     bump slot pointer
       sub  *vrnxt,xr              introduce standard vrblk offset
    loop through vrblks on one hash chain
prn06  mov  vrnxt(xr),xr           point to next vrblk on hash chain
    merge here first time to check block we found last time
prn07  mov  xr,wc                  copy vrblk pointer
       bze  wc,prn09               jump if chain end (or prnmv zero)
```

```
    prtnm (continued)
    loop to find value (chase down possible trblk chain)
prn08   mov  vrval(xr),xr                          load value
        beq  (xr),=b$trt,prn08                     loop if that was a trblk
    now we have the value, is this the block we want
        beq  xr,xl,prn10                           jump if this matches the name base
        mov  wc,xr                                 else point back to that vrblk
        brn  prn06                                 and loop back
    here to move to next hash slot
prn09   blt  wa,hshte,prn05                        loop back if more to go
        mov  xl,xr                                 else not found, copy value pointer
        jsr  prtvl                                 print value
        brn  prn11                                 and merge ahead
    here when we find a matching entry
prn10   mov  wc,xr                                 copy vrblk pointer
        mov  xr,prnmv                              save for next time in
        jsr  prtvn                                 print variable name
    merge here if no entry found
prn11   mov  (xl),wc                               load first word of name base
        bne  wc,=b$pdt,prn13                       jump if not program defined
    for program defined datatype, add right paren and exit
        mov  =ch$rp,wa                             load right paren, merge
    merge here to print final right paren or bracket
prn12   jsr  prtch                                 print final character
        mov  wb,wa                                 restore name offset
        brn  prn01                                 merge back to exit
```

```
    prtnm (continued)
    here for array or table
prn13   mov  =ch$bb,wa                          load left bracket
        jsr  prtch                              and print it
        mov  (xs),xl                            restore block pointer
        mov  (xl),wc                            load type word again
        bne  wc,=b$tet,prn15                    jump if not table
    here for table, print subscript value
        mov  tesub(xl),xr                       load subscript value
        mov  wb,xl                              save name offset
        jsr  prtvl                              print subscript value
        mov  xl,wb                              restore name offset
    merge here from array case to print right bracket
prn14   mov  =ch$rb,wa                          load right bracket
        brn  prn12                              merge back to print it
    here for array or vector, to print subscript(s)
prn15   mov  wb,wa                              copy name offset
        btw  wa                                 convert to words
        beq  wc,=b$art,prn16                    jump if arblk
    here for vector
        sub  =vcvlb,wa                          adjust for standard fields
        mti  wa                                 move to integer accum
        jsr  prtin                              print linear subscript
        brn  prn14                              merge back for right bracket
```

```
    prtnm (continued)
    here for array. first calculate absolute subscript
    offsets by successive divisions by the dimension values.
    this must be done right to left since the elements are
    stored row-wise. the subscripts are stacked as integers.
prn16   mov  arofs(xl),wc                        load length of bounds info
        ica  wc                                  adjust for arpro field
        btw  wc                                  convert to words
        sub  wc,wa                               get linear zero-origin subscript
        mti  wa                                  get integer value
        lct  wa,arndm(xl)                        set num of dimensions as loop count
        add  arofs(xl),xl                        point past bounds information
        sub  *arlbd,xl                           set ok offset for proper ptr later
    loop to stack subscript offsets
prn17   sub  *ardms,xl                           point to next set of bounds
        sti  prnsi                               save current offset
        rmi  ardim(xl)                           get remainder on dividing by dimens
        mfi  -(xs)                               store on stack (one word)
        ldi  prnsi                               reload argument
        dvi  ardim(xl)                           divide to get quotient
        bct  wa,prn17                            loop till all stacked
        zer  xr                                  set offset to first set of bounds
        lct  wb,arndm(xl)                        load count of dims to control loop
        brn  prn19                               jump into print loop
    loop to print subscripts from stack adjusting by adding
    the appropriate low bound value from the arblk
prn18   mov  =ch$cm,wa                           load a comma
        jsr  prtch                               print it
    merge here first time in (no comma required)
prn19   mti  (xs)+                               load subscript offset as integer
        add  xr,xl                               point to current lbd
        adi  arlbd(xl)                           add lbd to get signed subscript
        sub  xr,xl                               point back to start of arblk
        jsr  prtin                               print subscript
        add  *ardms,xr                           bump offset to next bounds
        bct  wb,prn18                            loop back till all printed
        brn  prn14                               merge back to print right bracket
        enp                                      end procedure prtnm
```

```
    prtnv -- print name value
    prtnv is used by the trace and dump routines to print
    a line of the form
    name = value
    note that the name involved can never be a pseudo-var
    (xl)                  name base
    (wa)                  name offset
    jsr  prtnv            call to print name = value
    (wb,wc,ra)            destroyed
prtnv  prc  e,0                                  entry point
       jsr  prtnm                                print argument name
       mov  xr,-(xs)                             save entry xr
       mov  wa,-(xs)                             save name offset (collectable)
       mov  =tmbeb,xr                            point to blank equal blank
       jsr  prtst                                print it
       mov  xl,xr                                copy name base
       add  wa,xr                                point to value
       mov  (xr),xr                              load value pointer
       jsr  prtvl                                print value
       jsr  prtnl                                terminate line
       mov  (xs)+,wa                             restore name offset
       mov  (xs)+,xr                             restore entry xr
       exi                                       return to caller
       enp                                       end procedure prtnv
```

```
    prtpg  -- print a page throw
    prints a page throw or a few blank lines on the standard
    listing channel depending on the listing options chosen.
    jsr  prtpg            call for page eject
prtpg   prc   e,0                          entry point
        beq   stage,=stgxt,prp01           jump if execution time
        bze   lstlc,prp06                  return if top of page already
        zer   lstlc                        clear line count
    check type of listing
prp01   mov   xr,-(xs)                     preserve xr
        bnz   prstd,prp02                  eject if flag set
        bnz   prich,prp03                  jump if interactive listing channel
        bze   precl,prp03                  jump if compact listing
    perform an eject
prp02   jsr   sysep                        eject
        brn   prp04                        merge
    compact or interactive channel listing. cant print
    blanks until check made for headers printed and flag set.
prp03   mov   headp,xr                     remember headp
        mnz   headp                        set to avoid repeated prtpg calls
        jsr   prtnl                        print blank line
        jsr   prtnl                        print blank line
        jsr   prtnl                        print blank line
        mov   =num03,lstlc                 count blank lines
        mov   xr,headp                     restore header flag
```

```
    prptg (continued)
    print the heading
prp04   bnz   headp,prp05            jump if header listed
        mnz   headp                  mark headers printed
        mov   xl,-(xs)               keep xl
        mov   =headr,xr              point to listing header
        jsr   prtst                  place it
        jsr   sysid                  get system identification
        jsr   prtst                  append extra chars
        jsr   prtnl                  print it
        mov   xl,xr                  extra header line
        jsr   prtst                  place it
        jsr   prtnl                  print it
        jsr   prtnl                  print a blank
        jsr   prtnl                  and another
        add   =num04,lstlc           four header lines printed
        mov   (xs)+,xl               restore xl
    merge if header not printed
prp05   mov   (xs)+,xr               restore xr
    return
prp06   exi                          return
        enp                          end procedure prtpg
```

```
    prtps - print page with test for standard listing option
    if the standard listing option is selected, insist that
    an eject be done
    jsr  prtps            call for eject
prtps   prc  e,0                                   entry point
        mov  prsto,prstd                           copy option flag
        jsr  prtpg                                 print page
        zer  prstd                                 clear flag
        exi                                        return
        enp                                        end procedure prtps
```

```
    prtsn -- print statement number
    prtsn is used to initiate a print trace line by printing
    asterisks and the current statement number. the actual
    format of the output generated is.
    ***nnnnn**** iii.....iiii
    nnnnn is the statement number with leading zeros replaced
    by asterisks (e.g. *******9****)
    iii...iii represents a variable length output consisting
    of a number of letter i characters equal to fnclevel.
    jsr  prtsn            call to print statement number
    (wc)                  destroyed
prtsn   prc  e,0                                   entry point
        mov  xr,-(xs)                              save entry xr
        mov  wa,prsna                              save entry wa
        mov  =tmasb,xr                             point to asterisks
        jsr  prtst                                 print asterisks
        mov  =num04,profs                          point into middle of asterisks
        mti  kvstn                                 load statement number as integer
        jsr  prtin                                 print integer statement number
        mov  =prsnf,profs                          point past asterisks plus blank
        mov  kvfnc,xr                              get fnclevel
        mov  =ch$li,wa                             set letter i
    loop to generate letter i fnclevel times
prsn1   bze  xr,prsn2                              jump if all set
        jsr  prtch                                 else print an i
        dcv  xr                                    decrement counter
        brn  prsn1                                 loop back
    merge with all letter i characters generated
prsn2   mov  =ch$bl,wa                             get blank
        jsr  prtch                                 print blank
        mov  prsna,wa                              restore entry wa
        mov  (xs)+,xr                              restore entry xr
        exi                                        return to prtsn caller
        enp                                        end procedure prtsn
```

```
    prtst -- print string
    prtst places a string of characters in the print buffer
    see prtnl for global locations used
    note that the first word of the block (normally b$scl)
    is not used and need not be set correctly (see prtvn)
    (xr)                  string to be printed
    jsr  prtst            call to print string
    (profs)               updated past chars placed
prtst   prc   r,0                                       entry point
        bnz   headp,prst0                               were headers printed
        jsr   prtps                                     no - print them
    call syspr
prst0   mov   wa,prsva                                  save wa
        mov   wb,prsvb                                  save wb
        zer   wb                                        set chars printed count to zero
    loop to print successive lines for long string
prst1   mov   sclen(xr),wa                              load string length
        sub   wb,wa                                     subtract count of chars already out
        bze   wa,prst4                                  jump to exit if none left
        mov   xl,-(xs)                                  else stack entry xl
        mov   xr,-(xs)                                  save argument
        mov   xr,xl                                     copy for eventual move
        mov   prlen,xr                                  load print buffer length
        sub   profs,xr                                  get chars left in print buffer
        bnz   xr,prst2                                  skip if room left on this line
        jsr   prtnl                                     else print this line
        mov   prlen,xr                                  and set full width available
```

```
    prtst (continued)
    here with chars to print and some room in buffer
prst2   blo  wa,xr,prst3                          jump if room for rest of string
        mov  xr,wa                                else set to fill line
    merge here with character count in wa
prst3   mov  prbuf,xr                             point to print buffer
        plc  xl,wb                                point to location in string
        psc  xr,profs                             point to location in buffer
        add  wa,wb                                bump string chars count
        add  wa,profs                             bump buffer pointer
        mov  wb,prsvc                             preserve char counter
        mvc                                       move characters to buffer
        mov  prsvc,wb                             recover char counter
        mov  (xs)+,xr                             restore argument pointer
        mov  (xs)+,xl                             restore entry xl
        brn  prst1                                loop back to test for more
    here to exit after printing string
prst4   mov  prsvb,wb                             restore entry wb
        mov  prsva,wa                             restore entry wa
        exi                                       return to prtst caller
        enp                                       end procedure prtst
```

```
    prttr -- print to terminal
    called to print contents of standard print buffer to
    online terminal. clears buffer down and resets profs.
    jsr  prttr            call for print
    (wa,wb)               destroyed
prttr   prc   e,0                                   entry point
        mov   xr,-(xs)                              save xr
        jsr   prtic                                 print buffer contents
        mov   prbuf,xr                              point to print bfr to clear it
        lct   wa,prlnw                              get buffer length
        add   *schar,xr                             point past scblk header
        mov   nullw,wb                              get blanks
    loop to clear buffer
prtt1   mov   wb,(xr)+                              clear a word
        bct   wa,prtt1                              loop
        zer   profs                                 reset profs
        mov   (xs)+,xr                              restore xr
        exi                                         return
        enp                                         end procedure prttr
```

```
    prtvl -- print a value
    prtvl places an appropriate character representation of
    a data value in the print buffer for dump/trace use.
    (xr)                  value to be printed
    jsr  prtvl            call to print value
    (wa,wb,wc,ra)         destroyed
prtvl   prc  r,0                                  entry point, recursive
        mov  xl,-(xs)                             save entry xl
        mov  xr,-(xs)                             save argument
        chk                                       check for stack overflow
    loop back here after finding a trap block (trblk)
prv01   mov  idval(xr),prvsi                      copy idval (if any)
        mov  (xr),xl                              load first word of block
        lei  xl                                   load entry point id
        bsw  xl,bl$$t,prv02                       switch on block type
        iff  bl$tr,prv04                          trblk
        iff  bl$ar,prv05                          arblk
        iff  bl$ic,prv08                          icblk
        iff  bl$nm,prv09                          nmblk
        iff  bl$pd,prv10                          pdblk
```

---

if **.cnra**
else

```
        iff  bl$rc,prv08                          rcblk
```

fi

```
        iff  bl$sc,prv11                          scblk
        iff  bl$se,prv12                          seblk
        iff  bl$tb,prv13                          tbblk
        iff  bl$vc,prv13                          vcblk
```

---

if **.cnbf**
else

```
        iff  bl$bc,prv15                          bcblk
```

fi

```
        esw                                       end of switch on block type
    here for blocks for which we just print datatype name
prv02   jsr  dtype                                get datatype name
        jsr  prtst                                print datatype name
    common exit point
prv03   mov  (xs)+,xr                             reload argument
        mov  (xs)+,xl                             restore xl
        exi                                       return to prtvl caller
    here for trblk
prv04   mov  trval(xr),xr                         load real value
        brn  prv01                                and loop back
```

```
    prtvl (continued)
    here for array (arblk)
    print array ( prototype ) blank number idval
prv05   mov  xr,xl                          preserve argument
        mov  =scarr,xr                      point to datatype name (array)
        jsr  prtst                          print it
        mov  =ch$pp,wa                      load left paren
        jsr  prtch                          print left paren
        add  arofs(xl),xl                   point to prototype
        mov  (xl),xr                        load prototype
        jsr  prtst                          print prototype
    vcblk, tbblk, bcblk merge here for ) blank number idval
prv06   mov  =ch$rp,wa                      load right paren
        jsr  prtch                          print right paren
    pdblk merges here to print blank number idval
prv07   mov  =ch$bl,wa                      load blank
        jsr  prtch                          print it
        mov  =ch$nm,wa                      load number sign
        jsr  prtch                          print it
        mti  prvsi                          get idval
        jsr  prtin                          print id number
        brn  prv03                          back to exit
    here for integer (icblk), real (rcblk)
    print character representation of value
prv08   mov  xr,-(xs)                       stack argument for gtstg
        jsr  gtstg                          convert to string
        ppm                                 error return is impossible
        jsr  prtst                          print the string
        mov  xr,dnamp                       delete garbage string from storage
        brn  prv03                          back to exit
```

```
    prtvl (continued)
    name (nmblk)
    for pseudo-variable, just print datatype name (name)
    for all other names, print dot followed by name rep
prv09   mov  nmbas(xr),xl                   load name base
        mov  (xl),wa                        load first word of block
        beq  wa,=b$kvt,prv02                just print name if keyword
        beq  wa,=b$evt,prv02                just print name if expression var
        mov  =ch$dt,wa                      else get dot
        jsr  prtch                          and print it
        mov  nmofs(xr),wa                   load name offset
        jsr  prtnm                          print name
        brn  prv03                          back to exit
    program datatype (pdblk)
    print datatype name ch$bl ch$nm idval
prv10   jsr  dtype                          get datatype name
        jsr  prtst                          print datatype name
        brn  prv07                          merge back to print id
    here for string (scblk)
    print quote string-characters quote
prv11   mov  =ch$sq,wa                      load single quote
        jsr  prtch                          print quote
        jsr  prtst                          print string value
        jsr  prtch                          print another quote
        brn  prv03                          back to exit
```

```
    prtvl (continued)
    here for simple expression (seblk)
    print asterisk variable-name
prv12   mov  =ch$as,wa                        load asterisk
        jsr  prtch                            print asterisk
        mov  sevar(xr),xr                     load variable pointer
        jsr  prtvn                            print variable name
        brn  prv03                            jump back to exit
    here for table (tbblk) and array (vcblk)
    print datatype ( prototype ) blank number idval
prv13   mov  xr,xl                            preserve argument
        jsr  dtype                            get datatype name
        jsr  prtst                            print datatype name
        mov  =ch$pp,wa                        load left paren
        jsr  prtch                            print left paren
        mov  tblen(xl),wa                     load length of block (=vclen)
        btw  wa                               convert to word count
        sub  =tbsi$,wa                        allow for standard fields
        beq  (xl),=b$tbt,prv14                jump if table
        add  =vctbd,wa                        for vcblk, adjust size
    print prototype
prv14   mti  wa                               move as integer
        jsr  prtin                            print integer prototype
        brn  prv06                            merge back for rest
```

---

if **.cnbf**
else

```
    prtvl (continued)
    here for buffer (bcblk)
prv15   mov  xr,xl              preserve argument
        mov  =scbuf,xr          point to datatype name (buffer)
        jsr  prtst              print it
        mov  =ch$pp,wa          load left paren
        jsr  prtch              print left paren
        mov  bcbuf(xl),xr       point to bfblk
        mti  bfalc(xr)          load allocation size
        jsr  prtin              print it
        mov  =ch$cm,wa          load comma
        jsr  prtch              print it
        mti  bclen(xl)          load defined length
        jsr  prtin              print it
        brn  prv06              merge to finish up
fi
        enp                     end procedure prtvl
```

```
    prtvn -- print natural variable name
    prtvn prints the name of a natural variable
    (xr)                 pointer to vrblk
    jsr  prtvn           call to print variable name
prtvn   prc  e,0                          entry point
        mov  xr,-(xs)                     stack vrblk pointer
        add  *vrsof,xr                    point to possible string name
        bnz  sclen(xr),prvn1              jump if not system variable
        mov  vrsvo(xr),xr                 point to svblk with name
    merge here with dummy scblk pointer in xr
prvn1   jsr  prtst                        print string name of variable
        mov  (xs)+,xr                     restore vrblk pointer
        exi                               return to prtvn caller
        enp                               end procedure prtvn
```

---

if **.cnra**

else

```
    rcbld -- build a real block
    (ra)                  real value for rcblk
    jsr  rcbld            call to build real block
    (xr)                  pointer to result rcblk
    (wa)                  destroyed
rcbld   prc  e,0                                         entry point
        mov  dnamp,xr                                    load pointer to next available loc
        add  *rcsi$,xr                                   point past new rcblk
        blo  xr,dname,rcbl1                              jump if there is room
        mov  *rcsi$,wa                                   else load rcblk length
        jsr  alloc                                       use standard allocator to get block
        add  wa,xr                                       point past block to merge
    merge here with xr pointing past the block obtained
rcbl1   mov  xr,dnamp                                    set new pointer
        sub  *rcsi$,xr                                   point back to start of block
        mov  =b$rcl,(xr)                                 store type word
        str  rcval(xr)                                   store real value in rcblk
        exi                                              return to rcbld caller
        enp                                              end procedure rcbld
fi
```

```
    readr -- read next source image at compile time
    readr is used to read the next source image. to process
    continuation cards properly, the compiler must read one
    line ahead. thus readr does not destroy the current image
    see also the nexts routine which actually gets the image.
    jsr  readr           call to read next image
    (xr)                 ptr to next image (0 if none)
    (r$cni)              copy of pointer
    (wa,wb,wc,xl)        destroyed
readr   prc  e,0                                entry point
        mov  r$cni,xr                           get ptr to next image
        bnz  xr,read3                           exit if already read
if .cinc
        bnz  cnind,reada                        if within include file
fi
        bne  stage,=stgic,read3                 exit if not initial compile
reada   mov  cswin,wa                           max read length
        zer  xl                                 clear any dud value in xl
        jsr  alocs                              allocate buffer
        jsr  sysrd                              read input image
        ppm  read4                              jump if eof or new file name
        icv  rdnln                              increment next line number
if .cpol
        dcv  polct                              test if time to poll interface
        bnz  polct,read0                        not yet
        zer  wa                                 =0 for poll
        mov  rdnln,wb                           line number
        jsr  syspl                              allow interactive access
        err  syspl                              allow interactive access
        ppm                                     single step
        ppm                                     expression evaluation
        mov  wa,polcs                           new countdown start value
        mov  wa,polct                           new counter value
fi
read0   ble  sclen(xr),cswin,read1              use smaller of string lnth ...
        mov  cswin,sclen(xr)                    ... and xxx of -inxxx
    perform the trim
read1   mnz  wb                                 set trimr to perform trim
        jsr  trimr                              trim trailing blanks
    merge here after read
read2   mov  xr,r$cni                           store copy of pointer
    merge here if no read attempted
read3   exi                                     return to readr caller
if .csfn
    here on end of file or new source file name.
    if this is a new source file name, the r$sfn table will
    be augmented with a new table entry consisting of the
    current compiler statement number as subscript, and the
    file name as value.
read4   bze  sclen(xr),read5                    jump if true end of file
        zer  wb                                 new source file name
        mov  wb,rdnln                           restart line counter for new file
```

```
        jsr  trimr                          remove unused space in block
        jsr  newfn                          record new file name
        brn  reada                          now reissue read for record data
    here on end of file
read5   mov  xr,dnamp                       pop unused scblk
```

---

if **.cinc**

```
        bze  cnind,read6                    jump if not within an include file
        zer  xl                             eof within include file
        jsr  sysif                          switch stream back to previous file
        ppm  sysif                          switch stream back to previous file
        mov  cnind,wa                       restore prev line number, file name
        add  =vcvlb,wa                      vector offset in words
        wtb  wa                             convert to bytes
        mov  r$ifa,xr                       file name array
        add  wa,xr                          ptr to element
        mov  (xr),r$sfc                     change source file name
        mov  =nulls,(xr)                    release scblk
        mov  r$ifl,xr                       line number array
        add  wa,xr                          ptr to element
        mov  (xr),xl                        icblk containing saved line number
        ldi  icval(xl)                      line number integer
        mfi  rdnln                          change source line number
        mov  =inton,(xr)                    release icblk
        dcv  cnind                          decrement nesting level
        mov  cmpsn,wb                       current statement number
        icv  wb                             anticipate end of previous stmt
        mti  wb                             convert to integer
        jsr  icbld                          build icblk for stmt number
        mov  r$sfn,xl                       file name table
        mnz  wb                             lookup statement number by name
        jsr  tfind                          allocate new teblk
        ppm                                 always possible to allocate block
        mov  r$sfc,teval(xl)                record file name as entry value
        beq  stage,=stgic,reada             if initial compile, reissue read
        bnz  cnind,reada                    still reading from include file
    outer nesting of execute-time compile of -include
    resume with any string remaining prior to -include.
        mov  r$ici,xl                       restore code argument string
        zer  r$ici                          release original string
        mov  cnsil,wa                       get length of string
        mov  cnspt,wb                       offset of characters left
        sub  wb,wa                          number of characters left
        mov  wa,scnil                       set new scan length
        zer  scnpt                          scan from start of substring
        jsr  sbstr                          create substring of remainder
        mov  xr,r$cim                       set scan image
        brn  read2                          return
```

fi

else

```
    here on end of file
read4   mov  xr,dnamp                       pop unused scblk
```

---

if **.cinc**

```
        bze   cnind,read6          jump if not within an include file
        zer   xl                   eof within include file
        jsr   sysif                switch stream back to previous file
        ppm   sysif                switch stream back to previous file
        dcv   cnind                decrement nesting level
        brn   reada                reissue read from previous stream
fi
fi
read6   zer   xr                   zero ptr as result
        brn   read2                merge
        enp                        end procedure readr
```

if **.c370**

```
sbool-- setup for boolean operations on strings
1(xs)                 first argument (if two)
0(xs)                 second argument
(wb)                  number of arguments
                       zero = one arguments
                       non-zero = two arguments
jsr  sbool            call to perform operation
ppm  loc              transfer loc for arg1 not string
ppm  loc              transfer loc for arg2 not string
ppm  loc              transfer loc arg lengths not equal
ppm  loc              transfer loc if null string args
(xs)                  arguments popped, result stacked
(xl)                  arg 1 chars to operate upon
(xr)                  copy of arg 2 if two arguments
(wa)                  no. of characters to process
(wc)                  no. of words to process (bct ready)
(wb)                  destroyed
the second argument string block is copied to a result
block, and pointers returned to allow the caller to
proceed with the desired operation if two arguments.
operations like and/or that do not alter the trailing
zeros in the last word of the string block can be
performed a word at a time.  operations such as compl
may either be performed a character at a time or will
have to adjust the last word if done a word at a time.
```

```
sbool  prc  n,3                entry point
       jsr  gtstg              convert second arg to string
       ppm  sbl05              jump if second arg not string
       mov  xr,xl              else save pointer
       mov  wa,wc              and length
       bze  wb,sbl01           only one argument if compl
       jsr  gtstg              convert first argument to string
       ppm  sbl04              jump if not string
       bne  wa,wc,sbl03        jump if lengths unequal
   merge here if only one argument
sbl01  mov  xr,-(xs)           stack first argument
       bze  wc,sbl02           return null if null argument
       jsr  alocs              allocate space for copy
       bze  wb,sbl06           only one argument if compl
       mov  wc,wa              string length
       mov  xr,wb              save address of copy
       ctb  wa,schar           get scblk length
       mvw                     move arg2 contents to copy
       mov  wb,xr              reload result ptr
sbl06  mov  (xs)+,xl           reload first argument
       mov  xr,-(xs)           stack result
       add  *schar,xl          point to characters in arg 1 block
       add  *schar,xr          point to characters in result block
       mov  wc,wa              character count
       ctw  wc,0               number of words of characters
       lct  wc,wc              prepare counter
```

```
       exi   wc,wc                              prepare counter
    here if null arguments
sbl02  exi   4                                  take null string exit
    here if argument lengths unequal
sbl03  exi   3                                  take unequal length error exit
    here if first arg is not a string
sbl04  exi   1                                  take bad first arg error exit
    here for second arg not a string
sbl05  exi   2                                  take bad second arg error exit
       enp                                      end procedure sbool
```

fi

```
    sbstr -- build a substring
    (xl)                  ptr to scblk/bfblk with chars
    (wa)                  number of chars in substring
    (wb)                  offset to first char in scblk
    jsr  sbstr            call to build substring
    (xr)                  ptr to new scblk with substring
    (xl)                  zero
    (wa,wb,wc,xl,ia)      destroyed
    note that sbstr is called with a dummy string pointer
    (pointing into a vrblk or svblk) to copy the name of a
    variable as a standard string value.
sbstr   prc   e,0                     entry point
        bze   wa,sbst2                jump if null substring
        jsr   alocs                   else allocate scblk
        mov   wc,wa                   move number of characters
        mov   xr,wc                   save ptr to new scblk
        plc   xl,wb                   prepare to load chars from old blk
        psc   xr                      prepare to store chars in new blk
        mvc                           move characters to new string
        mov   wc,xr                   then restore scblk pointer
    return point
sbst1   zer   xl                      clear garbage pointer in xl
        exi                           return to sbstr caller
    here for null substring
sbst2   mov   =nulls,xr               set null string as result
        brn   sbst1                   return
        enp                           end procedure sbstr
```

```
    stgcc -- compute counters for stmt startup testing
    jsr  stgcc            call to recompute counters
    (wa,wb)               destroyed
    on exit, stmcs and stmct contain the counter value to
    tested in stmgo.
stgcc   prc
```

---

if **.cpol**

```
        mov  polcs,wa          assume no profiling or stcount tracing
        mov  =num01,wb         poll each time polcs expires
```

else

```
        mov  cfp$m,wa          assume no profiling or stcount tracing
```

fi

```
        ldi  kvstl             get stmt limit
        bnz  kvpfl,stgc1       jump if profiling enabled
        ilt  stgc3             no stcount tracing if negative
        bze  r$stc,stgc2       jump if not stcount tracing
    here if profiling or if stcount tracing enabled
```

---

if **.cpol**

```
stgc1   mov  wa,wb             count polcs times within stmg
        mov  =num01,wa         break out of stmgo on each stmt
```

else

```
stgc1   mov  =num01,wa         break out of stmgo on each stmt
```

fi

```
        brn  =num01,wa         break out of stmgo on each stmt
    check that stmcs does not exceed kvstl
stgc2   mti  wa                breakout count start value
        sbi  kvstl             proposed stmcs minus stmt limit
        ile  stgc3             jump if stmt count does not limit
        ldi  kvstl             stlimit limits breakcount count
        mfi  wa                use it instead
    re-initialize counter
stgc3   mov  wa,stmcs          update breakout count start value
        mov  wa,stmct          reset breakout counter
```

---

if **.cpol**

```
        mov  wa,stmct          reset breakout counter
```

fi

```
        exi  wa,stmct          reset breakout counter
```

```
    tfind -- locate table element
    (xr)                 subscript value for element
    (xl)                 pointer to table
    (wb)                 zero by value, non-zero by name
    jsr  tfind           call to locate element
    ppm  loc             transfer location if access fails
    (xr)                 element value (if by value)
    (xr)                 destroyed (if by name)
    (xl,wa)              teblk name (if by name)
    (xl,wa)              destroyed (if by value)
    (wc,ra)              destroyed
    note that if a call by value specifies a non-existent
    subscript, the default value is returned without building
    a new teblk.
tfind   prc  e,1                          entry point
        mov  wb,-(xs)                     save name/value indicator
        mov  xr,-(xs)                     save subscript value
        mov  xl,-(xs)                     save table pointer
        mov  tblen(xl),wa                 load length of tbblk
        btw  wa                           convert to word count
        sub  =tbbuk,wa                    get number of buckets
        mti  wa                           convert to integer value
        sti  tfnsi                        save for later
        mov  (xr),xl                      load first word of subscript
        lei  xl                           load block entry id (bl$xx)
        bsw  xl,bl$$d,tfn00               switch on block type
        iff  bl$ic,tfn02                  jump if integer
```

```
if .cnra
else
        iff  bl$rc,tfn02                  real
fi
        iff  bl$p0,tfn03                  jump if pattern
        iff  bl$p1,tfn03                  jump if pattern
        iff  bl$p2,tfn03                  jump if pattern
        iff  bl$nm,tfn04                  jump if name
        iff  bl$sc,tfn05                  jump if string
        esw                               end switch on block type
    here for blocks for which we use the second word of the
    block as the hash source (see block formats for details).
tfn00   mov  1(xr),wa                     load second word
    merge here with one word hash source in wa
tfn01   mti  wa                           convert to integer
        brn  tfn06                        jump to merge
```

```
    tfind (continued)
    here for integer or real
    possibility of overflow exist on twos complement
    machine if hash source is most negative integer or is
    a real having the same bit pattern.
tfn02   ldi  1(xr)              load value as hash source
        ige  tfn06              ok if positive or zero
        ngi                     make positive
        iov  tfn06              clear possible overflow
        brn  tfn06              merge
    for pattern, use first word (pcode) as source
tfn03   mov  (xr),wa            load first word as hash source
        brn  tfn01              merge back
    for name, use offset as hash source
tfn04   mov  nmofs(xr),wa       load offset as hash source
        brn  tfn01              merge back
    here for string
tfn05   jsr  hashs              call routine to compute hash
    merge here with hash source in (ia)
tfn06   rmi  tfnsi              compute hash index by remaindering
        mfi  wc                 get as one word integer
        wtb  wc                 convert to byte offset
        mov  (xs),xl            get table ptr again
        add  wc,xl              point to proper bucket
        mov  tbbuk(xl),xr       load first teblk pointer
        beq  xr,(xs),tfn10      jump if no teblks on chain
    loop through teblks on hash chain
tfn07   mov  xr,wb              save teblk pointer
        mov  tesub(xr),xr       load subscript value
        mov  1(xs),xl           load input argument subscript val
        jsr  ident              compare them
        ppm  tfn08              jump if equal (ident)
    here if no match with that teblk
        mov  wb,xl              restore teblk pointer
        mov  tenxt(xl),xr       point to next teblk on chain
        bne  xr,(xs),tfn07      jump if there is one
    here if no match with any teblk on chain
        mov  *tenxt,wc          set offset to link field (xl base)
        brn  tfn11              jump to merge
```

```
    tfind (continued)
    here we have found a matching element
tfn08   mov  wb,xl                          restore teblk pointer
        mov  *teval,wa                      set teblk name offset
        mov  2(xs),wb                       restore name/value indicator
        bnz  wb,tfn09                       jump if called by name
        jsr  acess                          else get value
        ppm  tfn12                          jump if reference fails
        zer  wb                             restore name/value indicator
    common exit for entry found
tfn09   add  *num03,xs                      pop stack entries
        exi                                 return to tfind caller
    here if no teblks on the hash chain
tfn10   add  *tbbuk,wc                      get offset to bucket ptr
        mov  (xs),xl                        set tbblk ptr as base
    merge here with (xl,wc) base,offset of final link
tfn11   mov  (xs),xr                        tbblk pointer
        mov  tbinv(xr),xr                   load default value in case
        mov  2(xs),wb                       load name/value indicator
        bze  wb,tfn09                       exit with default if value call
        mov  xr,wb                          copy default value
    here we must build a new teblk
        mov  *tesi$,wa                      set size of teblk
        jsr  alloc                          allocate teblk
        add  wc,xl                          point to hash link
        mov  xr,(xl)                        link new teblk at end of chain
        mov  =b$tet,(xr)                    store type word
        mov  wb,teval(xr)                   set default as initial value
        mov  (xs)+,tenxt(xr)                set tbblk ptr to mark end of chain
        mov  (xs)+,tesub(xr)                store subscript value
        mov  (xs)+,wb                       restore name/value indicator
        mov  xr,xl                          copy teblk pointer (name base)
        mov  *teval,wa                      set offset
        exi                                 return to caller with new teblk
    acess fail return
tfn12   exi  1                              alternative return
        enp                                 end procedure tfind
```

```
    tmake -- make new table
    (xl)                 initial lookup value
    (wc)                 number of buckets desired
    jsr  tmake           call to make new table
    (xr)                 new table
    (wa,wb)              destroyed
tmake  prc
       mov  wc,wa                           copy number of headers
       add  =tbsi$,wa                       adjust for standard fields
       wtb  wa                              convert length to bytes
       jsr  alloc                           allocate space for tbblk
       mov  xr,wb                           copy pointer to tbblk
       mov  =b$tbt,(xr)+                    store type word
       zer  (xr)+                           zero id for the moment
       mov  wa,(xr)+                        store length (tblen)
       mov  xl,(xr)+                        store initial lookup value
       lct  wc,wc                           set loop counter (num headers)
    loop to initialize all bucket pointers
tma01  mov  wb,(xr)+                        store tbblk ptr in bucket header
       bct  wc,tma01                        loop till all stored
       mov  wb,xr                           recall pointer to tbblk
       exi  wb,xr                           recall pointer to tbblk
       enp  wb,xr                           recall pointer to tbblk
```

```
    vmake -- create a vector
    (wa)                   number of elements in vector
    (xl)                   default value for vector elements
    jsr  vmake             call to create vector
    ppm  loc               if vector too large
    (xr)                   pointer to vcblk
    (wa,wb,wc,xl)          destroyed
vmake   prc   e,1                                  entry point
        lct   wb,wa                                copy elements for loop later on
        add   =vcsi$,wa                            add space for standard fields
        wtb   wa                                   convert length to bytes
        bgt   wa,mxlen,vmak2                       fail if too large
        jsr   alloc                                allocate space for vcblk
        mov   =b$vct,(xr)                          store type word
        zer   idval(xr)                            initialize idval
        mov   wa,vclen(xr)                         set length
        mov   xl,wc                                copy default value
        mov   xr,xl                                copy vcblk pointer
        add   *vcvls,xl                            point to first element value
    loop to set vector elements to default value
vmak1   mov   wc,(xl)+                             store one value
        bct   wb,vmak1                             loop till all stored
        exi                                        success return
    here if desired vector size too large
vmak2   exi   1                                    fail return
        enp   1                                    fail return
```

```
scane -- scan an element
scane is called at compile time (by expan ,cmpil,cncrd)
to scan one element from the input image.
(scncc)              non-zero if called from cncrd
jsr  scane           call to scan element
(xr)                 result pointer (see below)
(xl)                 syntax type code (t$xxx)
the following global locations are used.
r$cim                pointer to string block (scblk)
                     for current input image.
r$cni                pointer to next input image string
                     pointer (zero if none).
r$scp                save pointer (exit xr) from last
                     call in case rescan is set.
scnbl                this location is set non-zero on
                     exit if scane scanned past blanks
                     before locating the current element
                     the end of a line counts as blanks.
scncc                cncrd sets this non-zero to scan
                     control card names and clears it
                     on return
scnil                length of current input image
scngo                if set non-zero on entry, f and s
                     are returned as separate syntax
                     types (not letters) (goto pro-
                     cessing). scngo is reset on exit.
scnpt                offset to current loc in r$cim
scnrs                if set non-zero on entry, scane
                     returns the same result as on the
                     last call (rescan). scnrs is reset
                     on exit from any call to scane.
scntp                save syntax type from last
                     call (in case rescan is set).
```

```
scane (continued)
element scanned       xl        xr
---------------       --        --
control card name     0         pointer to scblk for name
unary operator        t$uop     ptr to operator dvblk
left paren            t$lpr     t$lpr
left bracket          t$lbr     t$lbr
comma                 t$cma     t$cma
function call         t$fnc     ptr to function vrblk
variable              t$var     ptr to vrblk
string constant       t$con     ptr to scblk
integer constant      t$con     ptr to icblk
```

---

if **.cnra**
else

```
real constant         t$con     ptr to rcblk
```

fi

```
binary operator       t$bop     ptr to operator dvblk
right paren           t$rpr     t$rpr
right bracket         t$rbr     t$rbr
colon                 t$col     t$col
semi-colon            t$smc     t$smc
f (scngo ne 0)        t$fgo     t$fgo
s (scngo ne 0)        t$sgo     t$sgo
```

```
    scane (continued)
    entry point
scane   prc   e,0                  entry point
        zer   scnbl                reset blanks flag
        mov   wa,scnsa             save wa
        mov   wb,scnsb             save wb
        mov   wc,scnsc             save wc
        bze   scnrs,scn03          jump if no rescan
    here for rescan request
        mov   scntp,xl             set previous returned scan type
        mov   r$scp,xr             set previous returned pointer
        zer   scnrs                reset rescan switch
        brn   scn13                jump to exit
    come here to read new image to test for continuation
scn01   jsr   readr                read next image
        mov   *dvubs,wb            set wb for not reading name
        bze   xr,scn30             treat as semi-colon if none
        plc   xr                   else point to first character
        lch   wc,(xr)              load first character
        beq   wc,=ch$dt,scn02      jump if dot for continuation
        bne   wc,=ch$pl,scn30      else treat as semicolon unless plus
    here for continuation line
scn02   jsr   nexts                acquire next source image
        mov   =num01,scnpt         set scan pointer past continuation
        mnz   scnbl                set blanks flag
```

```
    scane (continued)
    merge here to scan next element on current line
scn03   mov  scnpt,wa                   load current offset
        beq  wa,scnil,scn01             check continuation if end
        mov  r$cim,xl                   point to current line
        plc  xl,wa                      point to current character
        mov  wa,scnse                   set start of element location
        mov  =opdvs,wc                  point to operator dv list
        mov  *dvubs,wb                  set constant for operator circuit
        brn  scn06                      start scanning
    loop here to ignore leading blanks and tabs
scn05   bze  wb,scn10                   jump if trailing
        icv  scnse                      increment start of element
        beq  wa,scnil,scn01             jump if end of image
        mnz  scnbl                      note blanks seen
    the following jump is used repeatedly for scanning out
    the characters of a numeric constant or variable name.
    the registers are used as follows.
    (xr)                scratch
    (xl)                ptr to next character
    (wa)                current scan offset
    (wb)                *dvubs (0 if scanning name,const)
    (wc)                =opdvs (0 if scanning constant)
scn06   lch  xr,(xl)+                   get next character
        icv  wa                         bump scan offset
        mov  wa,scnpt                   store offset past char scanned
```

---

if **.cucf**

```
        bsw  xr,cfp$u,scn07             switch on scanned character
```

else

```
        bsw  xr,cfp$a,scn07             switch on scanned character
```

fi

```
    switch table for switch on character
        iff  ch$bl,scn05                blank
```

---

if **.caht**

```
        iff  ch$ht,scn05                horizontal tab
```

fi

---

if **.cavt**

```
        iff  ch$vt,scn05                vertical tab
```

fi

---

if **.caex**

```
        iff  ch$ey,scn37                up arrow
```

fi

```
        iff  ch$d0,scn08                digit 0
        iff  ch$d1,scn08                digit 1
        iff  ch$d2,scn08                digit 2
        iff  ch$d3,scn08                digit 3
        iff  ch$d4,scn08                digit 4
        iff  ch$d5,scn08                digit 5
        iff  ch$d6,scn08                digit 6
        iff  ch$d7,scn08                digit 7
        iff  ch$d8,scn08                digit 8
        iff  ch$d9,scn08                digit 9
```

```
scane (continued)
      iff   ch$la,scn09          letter a
      iff   ch$lb,scn09          letter b
      iff   ch$lc,scn09          letter c
      iff   ch$ld,scn09          letter d
      iff   ch$le,scn09          letter e
      iff   ch$lg,scn09          letter g
      iff   ch$lh,scn09          letter h
      iff   ch$li,scn09          letter i
      iff   ch$lj,scn09          letter j
      iff   ch$lk,scn09          letter k
      iff   ch$ll,scn09          letter l
      iff   ch$lm,scn09          letter m
      iff   ch$ln,scn09          letter n
      iff   ch$lo,scn09          letter o
      iff   ch$lp,scn09          letter p
      iff   ch$lq,scn09          letter q
      iff   ch$lr,scn09          letter r
      iff   ch$lt,scn09          letter t
      iff   ch$lu,scn09          letter u
      iff   ch$lv,scn09          letter v
      iff   ch$lw,scn09          letter w
      iff   ch$lx,scn09          letter x
      iff   ch$ly,scn09          letter y
      iff   ch$l$,scn09          letter z
```

---

if **.casl**

```
      iff   ch$$a,scn09          shifted a
      iff   ch$$b,scn09          shifted b
      iff   ch$$c,scn09          shifted c
      iff   ch$$d,scn09          shifted d
      iff   ch$$e,scn09          shifted e
      iff   ch$$f,scn20          shifted f
      iff   ch$$g,scn09          shifted g
      iff   ch$$h,scn09          shifted h
      iff   ch$$i,scn09          shifted i
      iff   ch$$j,scn09          shifted j
      iff   ch$$k,scn09          shifted k
      iff   ch$$l,scn09          shifted l
      iff   ch$$m,scn09          shifted m
      iff   ch$$n,scn09          shifted n
      iff   ch$$o,scn09          shifted o
      iff   ch$$p,scn09          shifted p
      iff   ch$$q,scn09          shifted q
      iff   ch$$r,scn09          shifted r
      iff   ch$$s,scn21          shifted s
      iff   ch$$t,scn09          shifted t
      iff   ch$$u,scn09          shifted u
      iff   ch$$v,scn09          shifted v
      iff   ch$$w,scn09          shifted w
      iff   ch$$x,scn09          shifted x
      iff   ch$$y,scn09          shifted y
      iff   ch$$$,scn09          shifted z
```

fi

```
scane (continued)
        iff   ch$sq,scn16                       single quote
        iff   ch$dq,scn17                       double quote
        iff   ch$lf,scn20                       letter f
        iff   ch$ls,scn21                       letter s
        iff   ch$un,scn24                       underline
        iff   ch$pp,scn25                       left paren
        iff   ch$rp,scn26                       right paren
        iff   ch$rb,scn27                       right bracket
        iff   ch$bb,scn28                       left bracket
        iff   ch$cb,scn27                       right bracket
        iff   ch$ob,scn28                       left bracket
        iff   ch$cl,scn29                       colon
        iff   ch$sm,scn30                       semi-colon
        iff   ch$cm,scn31                       comma
        iff   ch$dt,scn32                       dot
        iff   ch$pl,scn33                       plus
        iff   ch$mn,scn34                       minus
        iff   ch$nt,scn35                       not
        iff   ch$dl,scn36                       dollar
        iff   ch$ex,scn37                       exclamation mark
        iff   ch$pc,scn38                       percent
        iff   ch$sl,scn40                       slash
        iff   ch$nm,scn41                       number sign
        iff   ch$at,scn42                       at
        iff   ch$br,scn43                       vertical bar
        iff   ch$am,scn44                       ampersand
        iff   ch$qu,scn45                       question mark
        iff   ch$eq,scn46                       equal
        iff   ch$as,scn49                       asterisk
        esw                                     end switch on character
    here for illegal character (underline merges)
scn07   bze   wb,scn10                          jump if scanning name or constant
        erb   230,syntax error:                 illegal character
```

```
    scane (continued)
    here for digits 0-9
scn08  bze  wb,scn09          keep scanning if name/constant
       zer  wc                else set flag for scanning constant
    here for letter. loop here when scanning name/constant
scn09  beq  wa,scnil,scn11    jump if end of image
       zer  wb                set flag for scanning name/const
       brn  scn06             merge back to continue scan
    come here for delimiter ending name or constant
scn10  dcv  wa                reset offset to point to delimiter
    come here after finishing scan of name or constant
scn11  mov  wa,scnpt          store updated scan offset
       mov  scnse,wb          point to start of element
       sub  wb,wa             get number of characters
       mov  r$cim,xl          point to line image
       bnz  wc,scn15          jump if name
    here after scanning out numeric constant
       jsr  sbstr             get string for constant
       mov  xr,dnamp          delete from storage (not needed)
       jsr  gtnum             convert to numeric
       ppm  scn14             jump if conversion failure
    merge here to exit with constant
scn12  mov  =t$con,xl         set result type of constant
```

```
    scane (continued)
    common exit point (xr,xl) set
scn13   mov  scnsa,wa                    restore wa
        mov  scnsb,wb                    restore wb
        mov  scnsc,wc                    restore wc
        mov  xr,r$scp                    save xr in case rescan
        mov  xl,scntp                    save xl in case rescan
        zer  scngo                       reset possible goto flag
        exi                              return to scane caller
    here if conversion error on numeric item
scn14   erb  231,syntax error:           invalid numeric item
    here after scanning out variable name
scn15   jsr  sbstr                       build string name of variable
        bnz  scncc,scn13                 return if cncrd call
        jsr  gtnvr                       locate/build vrblk
        ppm                              dummy (unused) error return
        mov  =t$var,xl                   set type as variable
        brn  scn13                       back to exit
    here for single quote (start of string constant)
scn16   bze  wb,scn10                    terminator if scanning name or cnst
        mov  =ch$sq,wb                   set terminator as single quote
        brn  scn18                       merge
    here for double quote (start of string constant)
scn17   bze  wb,scn10                    terminator if scanning name or cnst
        mov  =ch$dq,wb                   set double quote terminator, merge
    loop to scan out string constant
scn18   beq  wa,scnil,scn19              error if end of image
        lch  wc,(xl)+                    else load next character
        icv  wa                          bump offset
        bne  wc,wb,scn18                 loop back if not terminator
```

```
    scane (continued)
    here after scanning out string constant
        mov  scnpt,wb                  point to first character
        mov  wa,scnpt                  save offset past final quote
        dcv  wa                        point back past last character
        sub  wb,wa                     get number of characters
        mov  r$cim,xl                  point to input image
        jsr  sbstr                     build substring value
        brn  scn12                     back to exit with constant result
    here if no matching quote found
scn19   mov  wa,scnpt                  set updated scan pointer
        erb  232,syntax error:         unmatched string quote
    here for f (possible failure goto)
scn20   mov  =t$fgo,xr                 set return code for fail goto
        brn  scn22                     jump to merge
    here for s (possible success goto)
scn21   mov  =t$sgo,xr                 set success goto as return code
    special goto cases merge here
scn22   bze  scngo,scn09               treat as normal letter if not goto
    merge here for special character exit
scn23   bze  wb,scn10                  jump if end of name/constant
        mov  xr,xl                     else copy code
        brn  scn13                     and jump to exit
    here for underline
scn24   bze  wb,scn09                  part of name if scanning name
        brn  scn07                     else illegal
```

```
    scane (continued)
    here for left paren
scn25   mov  =t$lpr,xr                        set left paren return code
        bnz  wb,scn23                         return left paren unless name
        bze  wc,scn10                         delimiter if scanning constant
    here for left paren after name (function call)
        mov  scnse,wb                         point to start of name
        mov  wa,scnpt                         set pointer past left paren
        dcv  wa                               point back past last char of name
        sub  wb,wa                            get name length
        mov  r$cim,xl                         point to input image
        jsr  sbstr                            get string name for function
        jsr  gtnvr                            locate/build vrblk
        ppm                                   dummy (unused) error return
        mov  =t$fnc,xl                        set code for function call
        brn  scn13                            back to exit
    processing for special characters
scn26   mov  =t$rpr,xr                        right paren, set code
        brn  scn23                            take special character exit
scn27   mov  =t$rbr,xr                        right bracket, set code
        brn  scn23                            take special character exit
scn28   mov  =t$lbr,xr                        left bracket, set code
        brn  scn23                            take special character exit
scn29   mov  =t$col,xr                        colon, set code
        brn  scn23                            take special character exit
scn30   mov  =t$smc,xr                        semi-colon, set code
        brn  scn23                            take special character exit
scn31   mov  =t$cma,xr                        comma, set code
        brn  scn23                            take special character exit
```

```
    scane (continued)
    here for operators. on entry, wc points to the table of
    operator dope vectors and wb is the increment to step
    to the next pair (binary/unary) of dope vectors in the
    list. on reaching scn46, the pointer has been adjusted to
    point to the appropriate pair of dope vectors.
    the first three entries are special since they can occur
    as part of a variable name (.) or constant (.+-).
scn32  bze  wb,scn09              dot can be part of name or constant
       add  wb,wc                 else bump pointer
scn33  bze  wc,scn09              plus can be part of constant
       bze  wb,scn48              plus cannot be part of name
       add  wb,wc                 else bump pointer
scn34  bze  wc,scn09              minus can be part of constant
       bze  wb,scn48              minus cannot be part of name
       add  wb,wc                 else bump pointer
scn35  add  wb,wc                 not
scn36  add  wb,wc                 dollar
scn37  add  wb,wc                 exclamation
scn38  add  wb,wc                 percent
scn39  add  wb,wc                 asterisk
scn40  add  wb,wc                 slash
scn41  add  wb,wc                 number sign
scn42  add  wb,wc                 at sign
scn43  add  wb,wc                 vertical bar
scn44  add  wb,wc                 ampersand
scn45  add  wb,wc                 question mark
    all operators come here (equal merges directly)
    (wc) points to the binary/unary pair of operator dvblks.
scn46  bze  wb,scn10              operator terminates name/constant
       mov  wc,xr                 else copy dv pointer
       lch  wc,(xl)               load next character
       mov  =t$bop,xl             set binary op in case
       beq  wa,scnil,scn47        should be binary if image end
       beq  wc,=ch$bl,scn47       should be binary if followed by blk
```

```
if .caht
       beq  wc,=ch$ht,scn47       jump if horizontal tab
fi
```

```
if .cavt
       beq  wc,=ch$vt,scn47       jump if vertical tab
fi
       beq  wc,=ch$sm,scn47       semicolon can immediately follow =
       beq  wc,=ch$cl,scn47       colon can immediately follow =
       beq  wc,=ch$rp,scn47       right paren can immediately follow =
       beq  wc,=ch$rb,scn47       right bracket can immediately follow =
       beq  wc,=ch$cb,scn47       right bracket can immediately follow =
    here for unary operator
       add  *dvbs$,xr             point to dv for unary op
       mov  =t$uop,xl             set type for unary operator
       ble  scntp,=t$uok,scn13    ok unary if ok preceding element
```

```
    scane (continued)
    merge here to require preceding blanks
scn47   bnz   scnbl,scn13                    all ok if preceding blanks, exit
    fail operator in this position
scn48   erb   233,syntax error:              invalid use of operator
    here for asterisk, could be ** substitute for exclamation
scn49   bze   wb,scn10                       end of name if scanning name
        beq   wa,scnil,scn39                 not ** if * at image end
        mov   wa,xr                          else save offset past first *
        mov   wa,scnof                       save another copy
        lch   wa,(xl)+                       load next character
        bne   wa,=ch$as,scn50                not ** if next char not *
        icv   xr                             else step offset past second *
        beq   xr,scnil,scn51                 ok exclam if end of image
        lch   wa,(xl)                        else load next character
        beq   wa,=ch$bl,scn51                exclamation if blank
```

---

if **.caht**

```
        beq   wa,=ch$ht,scn51                exclamation if horizontal tab
```

fi

---

if **.cavt**

```
        beq   wa,=ch$vt,scn51                exclamation if vertical tab
```

fi

```
    unary *
scn50   mov   scnof,wa                       recover stored offset
        mov   r$cim,xl                       point to line again
        plc   xl,wa                          point to current char
        brn   scn39                          merge with unary *
    here for ** as substitute for exclamation
scn51   mov   xr,scnpt                       save scan pointer past 2nd *
        mov   xr,wa                          copy scan pointer
        brn   scn37                          merge with exclamation
        enp                                  end procedure scane
```

```
    scngf -- scan goto field
    scngf is called from cmpil to scan and analyze a goto
    field including the surrounding brackets or parentheses.
    for a normal goto, the result returned is either a vrblk
    pointer for a simple label operand, or a pointer to an
    expression tree with a special outer unary operator
    (o$goc). for a direct goto, the result returned is a
    pointer to an expression tree with the special outer
    unary operator o$god.
    jsr  scngf            call to scan goto field
    (xr)                  result (see above)
    (xl,wa,wb,wc)         destroyed
scngf   prc  e,0                                 entry point
        jsr  scane                               scan initial element
        beq  xl,=t$lpr,scng1                     skip if left paren (normal goto)
        beq  xl,=t$lbr,scng2                     skip if left bracket (direct goto)
        erb  234,syntax error:                   goto field incorrect
    here for left paren (normal goto)
scng1   mov  =num01,wb                           set expan flag for normal goto
        jsr  expan                               analyze goto field
        mov  =opdvn,wa                           point to opdv for complex goto
        ble  xr,statb,scng3                      jump if not in static (sgd15)
        blo  xr,state,scng4                      jump to exit if simple label name
        brn  scng3                               complex goto - merge
    here for left bracket (direct goto)
scng2   mov  =num02,wb                           set expan flag for direct goto
        jsr  expan                               scan goto field
        mov  =opdvd,wa                           set opdv pointer for direct goto
```

```
    scngf (continued)
    merge here to build outer unary operator block
scng3   mov  wa,-(xs)                                   stack operator dv pointer
        mov  xr,-(xs)                                   stack pointer to expression tree
        jsr  expop                                      pop operator off
        mov  (xs)+,xr                                   reload new expression tree pointer
    common exit point
scng4   exi                                             return to caller
        enp                                             end procedure scngf
```

```
    setvr -- set vrget,vrsto fields of vrblk
    setvr sets the proper values in the vrget and vrsto
    fields of a vrblk. it is called whenever trblks are
    added or subtracted (trace,stoptr,input,output,detach)
    (xr)                 pointer to vrblk
    jsr  setvr           call to set fields
    (xl,wa)              destroyed
    note that setvr ignores the call if xr does not point
    into the static region (i.e. is some other name base)
setvr  prc  e,0                         entry point
       bhi  xr,state,setv1              exit if not natural variable
    here if we have a vrblk
       mov  xr,xl                       copy vrblk pointer
       mov  =b$vrl,vrget(xr)            store normal get value
       beq  vrsto(xr),=b$vre,setv1      skip if protected variable
       mov  =b$vrs,vrsto(xr)            store normal store value
       mov  vrval(xl),xl                point to next entry on chain
       bne  (xl),=b$trt,setv1           jump if end of trblk chain
       mov  =b$vra,vrget(xr)            store trapped routine address
       mov  =b$vrv,vrsto(xr)            set trapped routine address
    merge here to exit to caller
setv1  exi                              return to setvr caller
       enp                              end procedure setvr
```

---

if **.cnsr**

else

```
sorta -- sort array
routine to sort an array or table on same basis as in
sitbol. a table is converted to an array, leaving two
dimensional arrays and vectors as cases to be considered.
whole rows of arrays are permuted according to the
ordering of the keys they contain, and the stride
referred to, is the the length of a row. it is one
for a vector.
the sort used is heapsort, fundamentals of data structure
horowitz and sahni, pitman 1977, page 347.
it is an order n*log(n) algorithm. in order
to make it stable, comparands may not compare equal. this
is achieved by sorting a copy array (referred to as the
sort array) containing at its high address end, byte
offsets to the rows to be sorted held in the original
array (referred to as the key array). sortc, the
comparison routine, accesses the keys through these
offsets and in the case of equality, resolves it by
comparing the offsets themselves. the sort permutes the
offsets which are then used in a final operation to copy
the actual items into the new array in sorted order.
references to zeroth item are to notional item
preceding first actual item.
reverse sorting for rsort is done by having the less than
test for keys effectively be replaced by a
greater than test.
1(xs)                 first arg - array or table
0(xs)                 2nd arg - index or pdtype name
(wa)                  0 , non-zero for sort , rsort
jsr  sorta            call to sort array
ppm  loc              transfer loc if table is empty
(xr)                  sorted array
(xl,wa,wb,wc)         destroyed
```

```
    sorta (continued)
sorta   prc  n,1                                  entry point
        mov  wa,srtsr                             sort/rsort indicator
        mov  *num01,srtst                         default stride of 1
        zer  srtof                                default zero offset to sort key
        mov  =nulls,srtdf                         clear datatype field name
        mov  (xs)+,r$sxr                          unstack argument 2
        mov  (xs)+,xr                             get first argument
        mnz  wa                                   use key/values of table entries
        jsr  gtarr                                convert to array
        ppm  srt18                                signal that table is empty
        ppm  srt16                                error if non-convertable
        mov  xr,-(xs)                             stack ptr to resulting key array
        mov  xr,-(xs)                             another copy for copyb
        jsr  copyb                                get copy array for sorting into
        ppm                                       cant fail
        mov  xr,-(xs)                             stack pointer to sort array
        mov  r$sxr,xr                             get second arg
        mov  num01(xs),xl                         get ptr to key array
        bne  (xl),=b$vct,srt02                    jump if arblk
        beq  xr,=nulls,srt01                      jump if null second arg
        jsr  gtnvr                                get vrblk ptr for it
        err  257,erroneous 2nd                    arg in sort/rsort of vector
        mov  xr,srtdf                             store datatype field name vrblk
    compute n and offset to item a(0) in vector case
srt01   mov  *vclen,wc                            offset to a(0)
        mov  *vcvls,wb                            offset to first item
        mov  vclen(xl),wa                         get block length
        sub  *vcsi$,wa                            get no. of entries, n (in bytes)
        brn  srt04                                merge
    here for array
srt02   ldi  ardim(xl)                            get possible dimension
        mfi  wa                                   convert to short integer
        wtb  wa                                   further convert to baus
        mov  *arvls,wb                            offset to first value if one
        mov  *arpro,wc                            offset before values if one dim.
        beq  arndm(xl),=num01,srt04               jump in fact if one dim.
        bne  arndm(xl),=num02,srt16                fail unless two dimens
        ldi  arlb2(xl)                            get lower bound 2 as default
        beq  xr,=nulls,srt03                      jump if default second arg
        jsr  gtint                                convert to integer
        ppm  srt17                                fail
        ldi  icval(xr)                            get actual integer value
```

```
    sorta (continued)
    here with sort column index in ia in array case
srt03  sbi  arlb2(xl)          subtract low bound
       iov  srt17              fail if overflow
       ilt  srt17              fail if below low bound
       sbi  ardm2(xl)          check against dimension
       ige  srt17              fail if too large
       adi  ardm2(xl)          restore value
       mfi  wa                 get as small integer
       wtb  wa                 offset within row to key
       mov  wa,srtof           keep offset
       ldi  ardm2(xl)          second dimension is row length
       mfi  wa                 convert to short integer
       mov  wa,xr              copy row length
       wtb  wa                 convert to bytes
       mov  wa,srtst           store as stride
       ldi  ardim(xl)          get number of rows
       mfi  wa                 as a short integer
       wtb  wa                 convert n to baus
       mov  arlen(xl),wc       offset past array end
       sub  wa,wc              adjust, giving space for n offsets
       dca  wc                 point to a(0)
       mov  arofs(xl),wb       offset to word before first item
       ica  wb                 offset to first item
    separate pre-processing for arrays and vectors done.
    to simplify later key comparisons, removal of any trblk
    trap blocks from entries in key array is effected.
    (xl) = 1(xs) = pointer to key array
    (xs) = pointer to sort array
    wa = number of items, n (converted to bytes).
    wb = offset to first item of arrays.
    wc = offset to a(0)
srt04  ble  wa,*num01,srt15    return if only a single item
       mov  wa,srtsn           store number of items (in baus)
       mov  wc,srtso           store offset to a(0)
       mov  arlen(xl),wc       length of array or vec (=vclen)
       add  xl,wc              point past end of array or vector
       mov  wb,srtsf           store offset to first row
       add  wb,xl              point to first item in key array
    loop through array
srt05  mov  (xl),xr            get an entry
    hunt along trblk chain
srt06  bne  (xr),=b$trt,srt07  jump out if not trblk
       mov  trval(xr),xr       get value field
       brn  srt06              loop
```

```
    sorta (continued)
    xr is value from end of chain
srt07   mov  xr,(xl)+                          store as array entry
        blt  xl,wc,srt05                       loop if not done
        mov  (xs),xl                           get adrs of sort array
        mov  srtsf,xr                          initial offset to first key
        mov  srtst,wb                          get stride
        add  srtso,xl                          offset to a(0)
        ica  xl                                point to a(1)
        mov  srtsn,wc                          get n
        btw  wc                                convert from bytes
        mov  wc,srtnr                          store as row count
        lct  wc,wc                             loop counter
    store key offsets at top of sort array
srt08   mov  xr,(xl)+                          store an offset
        add  wb,xr                             bump offset by stride
        bct  wc,srt08                          loop through rows
    perform the sort on offsets in sort array.
    (srtsn)              number of items to sort, n (bytes)
    (srtso)              offset to a(0)
srt09   mov  srtsn,wa                          get n
        mov  srtnr,wc                          get number of rows
        rsh  wc,1                              i = n / 2 (wc=i, index into array)
        wtb  wc                                convert back to bytes
    loop to form initial heap
srt10   jsr  sorth                             sorth(i,n)
        dca  wc                                i = i - 1
        bnz  wc,srt10                          loop if i gt 0
        mov  wa,wc                             i = n
    sorting loop. at this point, a(1) is the largest
    item, since algorithm initialises it as, and then maintains
    it as, root of tree.
srt11   dca  wc                                i = i - 1 (n - 1 initially)
        bze  wc,srt12                          jump if done
        mov  (xs),xr                           get sort array address
        add  srtso,xr                          point to a(0)
        mov  xr,xl                             a(0) address
        add  wc,xl                             a(i) address
        mov  num01(xl),wb                      copy a(i+1)
        mov  num01(xr),num01(xl)               move a(1) to a(i+1)
        mov  wb,num01(xr)                      complete exchange of a(1), a(i+1)
        mov  wc,wa                             n = i for sorth
        mov  *num01,wc                         i = 1 for sorth
        jsr  sorth                             sorth(1,n)
        mov  wa,wc                             restore wc
        brn  srt11                             loop
```

```
    sorta (continued)
    offsets have been permuted into required order by sort.
    copy array elements over them.
srt12   mov  (xs),xr                       base adrs of key array
        mov  xr,wc                         copy it
        add  srtso,wc                      offset of a(0)
        add  srtsf,xr                      adrs of first row of sort array
        mov  srtst,wb                      get stride
    copying loop for successive items. sorted offsets are
    held at end of sort array.
srt13   ica  wc                            adrs of next of sorted offsets
        mov  wc,xl                         copy it for access
        mov  (xl),xl                       get offset
        add  num01(xs),xl                  add key array base adrs
        mov  wb,wa                         get count of characters in row
        mvw                                copy a complete row
        dcv  srtnr                         decrement row count
        bnz  srtnr,srt13                   repeat till all rows done
    return point
srt15   mov  (xs)+,xr                      pop result array ptr
        ica  xs                            pop key array ptr
        zer  r$sxl                         clear junk
        zer  r$sxr                         clear junk
        exi                                return
    error point
srt16   erb  256,sort/rsort 1st            arg not suitable array or table
srt17   erb  258,sort/rsort 2nd            arg out of range or non-integer
    return point if input table is empty
srt18   exi  1                             return indication of null table
        enp                                end procudure sorta
```

```
    sortc --  compare sort keys
    compare two sort keys given their offsets. if
    equal, compare key offsets to give stable sort.
    note that if srtsr is non-zero (request for reverse
    sort), the quoted returns are inverted.
    for objects of differing datatypes, the entry point
    identifications are compared.
    (xl)                  base adrs for keys
    (wa)                  offset to key 1 item
    (wb)                  offset to key 2 item
    (srtsr)               zero/non-zero for sort/rsort
    (srtof)               offset within row to comparands
    jsr  sortc            call to compare keys
    ppm  loc              key1 less than key2
                          normal return, key1 gt than key2
    (xl,xr,wa,wb)         destroyed
sortc   prc   e,1                          entry point
        mov   wa,srts1                     save offset 1
        mov   wb,srts2                     save offset 2
        mov   wc,srtsc                     save wc
        add   srtof,xl                     add offset to comparand field
        mov   xl,xr                        copy base + offset
        add   wa,xl                        add key1 offset
        add   wb,xr                        add key2 offset
        mov   (xl),xl                      get key1
        mov   (xr),xr                      get key2
        bne   srtdf,=nulls,src12           jump if datatype field name used
```

```
    sortc (continued)
    merge after dealing with field name. try for strings.
src01   mov  (xl),wc                              get type code
        bne  wc,(xr),src02                        skip if not same datatype
        beq  wc,=b$scl,src09                      jump if both strings
        beq  wc,=b$icl,src14                      jump if both integers
```

if **.cnbf**

else

```
        beq  wc,=b$bct,src09                      jump if both buffers
```

fi

```
    datatypes different.  now try for numeric
src02   mov  xl,r$sxl                             keep arg1
        mov  xr,r$sxr                             keep arg2
```

if **.cnbf**

if **.cnsc**

```
        beq  wc,=b$scl,src11                      do not allow conversion to number
        beq  (xr),=b$scl,src11                    if either arg is a string
```

fi

else

```
    first examine for string/buffer comparison.  if so,
    allow lcomp to compare chars in string and buffer
    without converting buffer to a string.
        beq  wc,=b$scl,src13                      jump if key1 is a string
```

if **.cnsc**

```
        bne  wc,=b$bct,src15                      j if key1 is not a string or buffer
```

else

```
        bne  wc,=b$bct,src14                      try converting key 2 to a number
```

fi

```
    here if key1 is a buffer, key2 known not to be a buffer.
    if key2 is a string, then lcomp can proceed.
        beq  (xr),=b$scl,src09                    j if keys 1/2 are buffer/string
```

if **.cnsc**

```
        brn  src11                                prevent convert of key 1 to number
```

else

```
        brn  src14                                try converting key 1 to number
```

fi

```
    here if key1 is a string, key2 known not to be a string.
    if key2 is a buffer, then lcomp can proceed.
src13   beq  (xr),=b$bct,src09                    j if keys 1/2 are string/buffer
```

if **.cnsc**

```
        brn  src11                                prevent convert of key 1 to number
    here if key1 is not a string or buffer.
    examine key2.  if it is a string or buffer, then do not
    convert key2 to a number.
src15   beq  (xr),=b$scl,src11                    j if key 2 is a string
        beq  (xr),=b$bct,src11                    j if key 2 is a buffer
    here with keys 1/2 not strings or buffers
```

fi

fi

```
src14   mov  xl,-(xs)                             stack
        mov  xr,-(xs)                             args
```

```
        jsr   acomp                                       compare objects
        ppm   src10                                       not numeric
        ppm   src10                                       not numeric
        ppm   src03                                       key1 less
        ppm   src08                                       keys equal
        ppm   src05                                       key1 greater
    return if key1 smaller (sort), greater (rsort)
src03   bnz   srtsr,src06                                 jump if rsort
src04   mov   srtsc,wc                                    restore wc
        exi   1                                           return
    return if key1 greater (sort), smaller (rsort)
src05   bnz   srtsr,src04                                 jump if rsort
src06   mov   srtsc,wc                                    restore wc
        exi                                               return
    keys are of same datatype
src07   blt   xl,xr,src03                                 item first created is less
        bgt   xl,xr,src05                                 addresses rise in order of creation
    drop through or merge for identical or equal objects
src08   blt   srts1,srts2,src04                           test offsets or key addrss instead
        brn   src06                                       offset 1 greater
```

```
if .cnbf
    strings
else
    strings or buffers or some combination of same
fi
src09   mov  xl,-(xs)                  stack
        mov  xr,-(xs)                  args
        jsr  lcomp                     compare objects
        ppm                            cant
        ppm                            fail
        ppm  src03                     key1 less
        ppm  src08                     keys equal
        ppm  src05                     key1 greater
    arithmetic comparison failed - recover args
src10   mov  r$sxl,xl                  get arg1
        mov  r$sxr,xr                  get arg2
        mov  (xl),wc                   get type of key1
        beq  wc,(xr),src07             jump if keys of same type
    here to compare datatype ids
src11   mov  wc,xl                     get block type word
        mov  (xr),xr                   get block type word
        lei  xl                        entry point id for key1
        lei  xr                        entry point id for key2
        bgt  xl,xr,src05               jump if key1 gt key2
        brn  src03                     key1 lt key2
    datatype field name used
src12   jsr  sortf                     call routine to find field 1
        mov  xl,-(xs)                  stack item pointer
        mov  xr,xl                     get key2
        jsr  sortf                     find field 2
        mov  xl,xr                     place as key2
        mov  (xs)+,xl                  recover key1
        brn  src01                     merge
        enp                            procedure sortc
```

```
    sortf -- find field for sortc
    routine used by sortc to obtain item corresponding
    to a given field name, if this exists, in a programmer
    defined object passed as argument.
    if such a match occurs, record is kept of datatype
    name, field name and offset to field in order to
    short-circuit later searches on same type. note that
    dfblks are stored in static and hence cannot be moved.
    (srtdf)              vrblk pointer of field name
    (xl)                 possible pdblk pointer
    jsr  sortf           call to search for field name
    (xl)                 item found or original pdblk ptr
    (wc)                 destroyed
sortf   prc  e,0                                 entry point
        bne  (xl),=b$pdt,srtf3                   return if not pdblk
        mov  xr,-(xs)                            keep xr
        mov  srtfd,xr                            get possible former dfblk ptr
        bze  xr,srtf4                            jump if not
        bne  xr,pddfp(xl),srtf4                  jump if not right datatype
        bne  srtdf,srtff,srtf4                   jump if not right field name
        add  srtfo,xl                            add offset to required field
    here with xl pointing to found field
srtf1   mov  (xl),xl                             get item from field
    return point
srtf2   mov  (xs)+,xr                            restore xr
srtf3   exi                                      return
```

```
    sortf (continued)
    conduct a search
srtf4  mov  xl,xr              copy original pointer
       mov  pddfp(xr),xr       point to dfblk
       mov  xr,srtfd           keep a copy
       mov  fargs(xr),wc       get number of fields
       wtb  wc                 convert to bytes
       add  dflen(xr),xr       point past last field
    loop to find name in pdfblk
srtf5  dca  wc                 count down
       dca  xr                 point in front
       beq  (xr),srtdf,srtf6   skip out if found
       bnz  wc,srtf5           loop
       brn  srtf2              return - not found
    found
srtf6  mov  (xr),srtff         keep field name ptr
       add  *pdfld,wc          add offset to first field
       mov  wc,srtfo           store as field offset
       add  wc,xl              point to field
       brn  srtf1              return
       enp                     procedure sortf
```

```
    sorth -- heap routine for sorta
    this routine constructs a heap from elements of array, a.
    in this application, the elements are offsets to keys in
    a key array.
    (xs)                  pointer to sort array base
    1(xs)                 pointer to key array base
    (wa)                  max array index, n (in bytes)
    (wc)                  offset j in a to root (in *1 to *n)
    jsr  sorth            call sorth(j,n) to make heap
    (xl,xr,wb)            destroyed
sorth   prc   n,0                                 entry point
        mov   wa,srtsn                            save n
        mov   wc,srtwc                            keep wc
        mov   (xs),xl                             sort array base adrs
        add   srtso,xl                            add offset to a(0)
        add   wc,xl                               point to a(j)
        mov   (xl),srtrt                          get offset to root
        add   wc,wc                               double j - cant exceed n
    loop to move down tree using doubled index j
srh01   bgt   wc,srtsn,srh03                      done if j gt n
        beq   wc,srtsn,srh02                      skip if j equals n
        mov   (xs),xr                             sort array base adrs
        mov   num01(xs),xl                        key array base adrs
        add   srtso,xr                            point to a(0)
        add   wc,xr                               adrs of a(j)
        mov   num01(xr),wa                        get a(j+1)
        mov   (xr),wb                             get a(j)
    compare sons. (wa) right son, (wb) left son
        jsr   sortc                               compare keys - lt(a(j+1),a(j))
        ppm   srh02                               a(j+1) lt a(j)
        ica   wc                                  point to greater son, a(j+1)
```

```
    sorth (continued)
    compare root with greater son
srh02   mov  num01(xs),xl          key array base adrs
        mov  (xs),xr               get sort array address
        add  srtso,xr              adrs of a(0)
        mov  xr,wb                 copy this adrs
        add  wc,xr                 adrs of greater son, a(j)
        mov  (xr),wa               get a(j)
        mov  wb,xr                 point back to a(0)
        mov  srtrt,wb              get root
        jsr  sortc                 compare them - lt(a(j),root)
        ppm  srh03                 father exceeds sons - done
        mov  (xs),xr               get sort array adrs
        add  srtso,xr              point to a(0)
        mov  xr,xl                 copy it
        mov  wc,wa                 copy j
        btw  wc                    convert to words
        rsh  wc,1                  get j/2
        wtb  wc                    convert back to bytes
        add  wa,xl                 point to a(j)
        add  wc,xr                 adrs of a(j/2)
        mov  (xl),(xr)             a(j/2) = a(j)
        mov  wa,wc                 recover j
        aov  wc,wc,srh03           j = j*2. done if too big
        brn  srh01                 loop
    finish by copying root offset back into array
srh03   btw  wc                    convert to words
        rsh  wc,1                  j = j/2
        wtb  wc                    convert back to bytes
        mov  (xs),xr               sort array adrs
        add  srtso,xr              adrs of a(0)
        add  wc,xr                 adrs of a(j/2)
        mov  srtrt,(xr)            a(j/2) = root
        mov  srtsn,wa              restore wa
        mov  srtwc,wc              restore wc
        exi                        return
        enp                        end procedure sorth
fi
```

```
    trace -- set/reset a trace association
    this procedure is shared by trace and stoptr to
    either initiate or stop a trace respectively.
    (xl)                  trblk ptr (trace) or zero (stoptr)
    1(xs)                 first argument (name)
    0(xs)                 second argument (trace type)
    jsr  trace            call to set/reset trace
    ppm  loc              transfer loc if 1st arg is bad name
    ppm  loc              transfer loc if 2nd arg is bad type
    (xs)                  popped
    (xl,xr,wa,wb,wc,ia)   destroyed
trace   prc  n,2                              entry point
        jsr  gtstg                            get trace type string
        ppm  trc15                            jump if not string
        plc  xr                               else point to string
        lch  wa,(xr)                          load first character
```

---

if **.culc**

```
        flc  wa                               fold to upper case
```

fi

```
        mov  (xs),xr                          load name argument
        mov  xl,(xs)                          stack trblk ptr or zero
        mov  =trtac,wc                        set trtyp for access trace
        beq  wa,=ch$la,trc10                  jump if a (access)
        mov  =trtvl,wc                        set trtyp for value trace
        beq  wa,=ch$lv,trc10                  jump if v (value)
        beq  wa,=ch$bl,trc10                  jump if blank (value)
    here for l,k,f,c,r
        beq  wa,=ch$lf,trc01                  jump if f (function)
        beq  wa,=ch$lr,trc01                  jump if r (return)
        beq  wa,=ch$ll,trc03                  jump if l (label)
        beq  wa,=ch$lk,trc06                  jump if k (keyword)
        bne  wa,=ch$lc,trc15                  else error if not c (call)
    here for f,c,r
trc01   jsr  gtnvr                            point to vrblk for name
        ppm  trc16                            jump if bad name
        ica  xs                               pop stack
        mov  vrfnc(xr),xr                     point to function block
        bne  (xr),=b$pfc,trc17                error if not program function
        beq  wa,=ch$lr,trc02                  jump if r (return)
```

```
    trace (continued)
    here for f,c to set/reset call trace
        mov  xl,pfctr(xr)                    set/reset call trace
        beq  wa,=ch$lc,exnul                 exit with null if c (call)
    here for f,r to set/reset return trace
trc02   mov  xl,pfrtr(xr)                    set/reset return trace
        exi                                  return
    here for l to set/reset label trace
trc03   jsr  gtnvr                           point to vrblk
        ppm  trc16                           jump if bad name
        mov  vrlbl(xr),xl                    load label pointer
        bne  (xl),=b$trt,trc04               jump if no old trace
        mov  trlbl(xl),xl                    else delete old trace association
    here with old label trace association deleted
trc04   beq  xl,=stndl,trc16                 error if undefined label
        mov  (xs)+,wb                        get trblk ptr again
        bze  wb,trc05                        jump if stoptr case
        mov  wb,vrlbl(xr)                    else set new trblk pointer
        mov  =b$vrt,vrtra(xr)                set label trace routine address
        mov  wb,xr                           copy trblk pointer
        mov  xl,trlbl(xr)                    store real label in trblk
        exi                                  return
    here for stoptr case for label
trc05   mov  xl,vrlbl(xr)                    store label ptr back in vrblk
        mov  =b$vrg,vrtra(xr)                store normal transfer address
        exi                                  return
```

```
    trace (continued)
    here for k (keyword)
trc06   jsr    gtnvr                  point to vrblk
        ppm    trc16                  error if not natural var
        bnz    vrlen(xr),trc16        error if not system var
        ica    xs                     pop stack
        bze    xl,trc07               jump if stoptr case
        mov    xr,trkvr(xl)           store vrblk ptr in trblk for ktrex
    merge here with trblk set up in wb (or zero)
trc07   mov    vrsvp(xr),xr           point to svblk
        beq    xr,=v$ert,trc08        jump if errtype
        beq    xr,=v$stc,trc09        jump if stcount
        bne    xr,=v$fnc,trc17        else error if not fnclevel
    fnclevel
        mov    xl,r$fnc               set/reset fnclevel trace
        exi                           return
    errtype
trc08   mov    xl,r$ert               set/reset errtype trace
        exi                           return
    stcount
trc09   mov    xl,r$stc               set/reset stcount trace
        jsr    stgcc                  update countdown counters
        exi                           return
```

```
    trace (continued)
    a,v merge here with trtyp value in wc
trc10  jsr  gtvar                 locate variable
       ppm  trc16                 error if not appropriate name
       mov  (xs)+,wb              get new trblk ptr again
       add  xl,wa                 point to variable location
       mov  wa,xr                 copy variable pointer
    loop to search trblk chain
trc11  mov  (xr),xl               point to next entry
       bne  (xl),=b$trt,trc13     jump if not trblk
       blt  wc,trtyp(xl),trc13    jump if too far out on chain
       beq  wc,trtyp(xl),trc12    jump if this matches our type
       add  *trnxt,xl             else point to link field
       mov  xl,xr                 copy pointer
       brn  trc11                 and loop back
    here to delete an old trblk of the type we were given
trc12  mov  trnxt(xl),xl          get ptr to next block or value
       mov  xl,(xr)               store to delete this trblk
    here after deleting any old association of this type
trc13  bze  wb,trc14              jump if stoptr case
       mov  wb,(xr)               else link new trblk in
       mov  wb,xr                 copy trblk pointer
       mov  xl,trnxt(xr)          store forward pointer
       mov  wc,trtyp(xr)          store appropriate trap type code
    here to make sure vrget,vrsto are set properly
trc14  mov  wa,xr                 recall possible vrblk pointer
       sub  *vrval,xr             point back to vrblk
       jsr  setvr                 set fields if vrblk
       exi                        return
    here for bad trace type
trc15  exi  2                     take bad trace type error exit
    pop stack before failing
trc16  ica  xs                    pop stack
    here for bad name argument
trc17  exi  1                     take bad name error exit
       enp                        end procedure trace
```

```
    trbld -- build trblk
    trblk is used by the input, output and trace functions
    to construct a trblk (trap block)
    (xr)                 trtag or trter
    (xl)                 trfnc or trfpt
    (wb)                 trtyp
    jsr  trbld           call to build trblk
    (xr)                 pointer to trblk
    (wa)                 destroyed
trbld  prc  e,0                                  entry point
       mov  xr,-(xs)                             stack trtag (or trfnm)
       mov  *trsi$,wa                            set size of trblk
       jsr  alloc                                allocate trblk
       mov  =b$trt,(xr)                          store first word
       mov  xl,trfnc(xr)                         store trfnc (or trfpt)
       mov  (xs)+,trtag(xr)                      store trtag (or trfnm)
       mov  wb,trtyp(xr)                         store type
       mov  =nulls,trval(xr)                     for now, a null value
       exi                                       return to caller
       enp                                       end procedure trbld
```

```
    trimr -- trim trailing blanks
    trimr is passed a pointer to an scblk which must be the
    last block in dynamic storage. trailing blanks are
    trimmed off and the dynamic storage pointer reset to
    the end of the (possibly) shortened block.
    (wb)                  non-zero to trim trailing blanks
    (xr)                  pointer to string to trim
    jsr  trimr            call to trim string
    (xr)                  pointer to trimmed string
    (xl,wa,wb,wc)         destroyed
    the call with wb zero still performs the end zero pad
    and dnamp readjustment. it is used from acess if kvtrm=0.
trimr   prc  e,0                                 entry point
        mov  xr,xl                               copy string pointer
        mov  sclen(xr),wa                        load string length
        bze  wa,trim2                            jump if null input
        plc  xl,wa                               else point past last character
        bze  wb,trim3                            jump if no trim
        mov  =ch$bl,wc                           load blank character
    loop through characters from right to left
trim0   lch  wb,-(xl)                            load next character
```

---

if **.caht**

```
        beq  wb,=ch$ht,trim1                     jump if horizontal tab
```

fi

```
        bne  wb,wc,trim3                         jump if non-blank found
trim1   dcv  wa                                  else decrement character count
        bnz  wa,trim0                            loop back if more to check
    here if result is null (null or all-blank input)
trim2   mov  xr,dnamp                            wipe out input string block
        mov  =nulls,xr                           load null result
        brn  trim5                               merge to exit
```

```
    trimr (continued)
    here with non-blank found (merge for no trim)
trim3   mov  wa,sclen(xr)                        set new length
        mov  xr,xl                               copy string pointer
        psc  xl,wa                               ready for storing blanks
        ctb  wa,schar                            get length of block in bytes
        add  xr,wa                               point past new block
        mov  wa,dnamp                            set new top of storage pointer
        lct  wa,=cfp$c                           get count of chars in word
        zer  wc                                  set zero char
    loop to zero pad last word of characters
trim4   sch  wc,(xl)+                            store zero character
        bct  wa,trim4                            loop back till all stored
        csc  xl                                  complete store characters
    common exit point
trim5   zer  xl                                  clear garbage xl pointer
        exi                                      return to caller
        enp                                      end procedure trimr
```

```
    trxeq -- execute function type trace
    trxeq is used to execute a trace when a fourth argument
    has been supplied. trace has already been decremented.
    (xr)                  pointer to trblk
    (xl,wa)               name base,offset for variable
    jsr  trxeq            call to execute trace
    (wb,wc,ra)            destroyed
    the following stack entries are made before passing
    control to the trace function using the cfunc routine.
                          trxeq return point word(s)
                          saved value of trace keyword
                          trblk pointer
                          name base
                          name offset
                          saved value of r$cod
                          saved code ptr (-r$cod)
                          saved value of flptr
    flptr --------------- zero (dummy fail offset)
                          nmblk for variable name
    xs ------------------ trace tag
    r$cod and the code ptr are set to dummy values which
    cause control to return to the trxeq procedure on success
    or failure (trxeq ignores a failure condition).
trxeq   prc  r,0                                   entry point (recursive)
        mov  r$cod,wc                              load code block pointer
        scp  wb                                    get current code pointer
        sub  wc,wb                                 make code pointer into offset
        mov  kvtra,-(xs)                           stack trace keyword value
        mov  xr,-(xs)                              stack trblk pointer
        mov  xl,-(xs)                              stack name base
        mov  wa,-(xs)                              stack name offset
        mov  wc,-(xs)                              stack code block pointer
        mov  wb,-(xs)                              stack code pointer offset
        mov  flptr,-(xs)                           stack old failure pointer
        zer  -(xs)                                 set dummy fail offset
        mov  xs,flptr                              set new failure pointer
        zer  kvtra                                 reset trace keyword to zero
        mov  =trxdc,wc                             load new (dummy) code blk pointer
        mov  wc,r$cod                              set as code block pointer
        lcp  wc                                    and new code pointer
```

```
    trxeq (continued)
    now prepare arguments for function
        mov  wa,wb                        save name offset
        mov  *nmsi$,wa                    load nmblk size
        jsr  alloc                        allocate space for nmblk
        mov  =b$nml,(xr)                  set type word
        mov  xl,nmbas(xr)                 store name base
        mov  wb,nmofs(xr)                 store name offset
        mov  6(xs),xl                     reload pointer to trblk
        mov  xr,-(xs)                     stack nmblk pointer (1st argument)
        mov  trtag(xl),-(xs)              stack trace tag (2nd argument)
        mov  trfnc(xl),xl                 load trace vrblk pointer
        mov  vrfnc(xl),xl                 load trace function pointer
        beq  xl,=stndf,trxq2              jump if not a defined function
        mov  =num02,wa                    set number of arguments to two
        brn  cfunc                        jump to call function
    see o$txr for details of return to this point
trxq1   mov  flptr,xs                     point back to our stack entries
        ica  xs                           pop off garbage fail offset
        mov  (xs)+,flptr                  restore old failure pointer
        mov  (xs)+,wb                     reload code offset
        mov  (xs)+,wc                     load old code base pointer
        mov  wc,xr                        copy cdblk pointer
        mov  cdstm(xr),kvstn              restore stmnt no
        mov  (xs)+,wa                     reload name offset
        mov  (xs)+,xl                     reload name base
        mov  (xs)+,xr                     reload trblk pointer
        mov  (xs)+,kvtra                  restore trace keyword value
        add  wc,wb                        recompute absolute code pointer
        lcp  wb                           restore code pointer
        mov  wc,r$cod                     and code block pointer
        exi                               return to trxeq caller
    here if the target function is not defined
trxq2   erb  197,trace fourth             arg is not function name or null
        enp                               end procedure trxeq
```

```
xscan -- execution function argument scan
xscan scans out one token in a prototype argument in
array,clear,data,define,load function calls. xscan
calls must be preceded by a call to the initialization
procedure xscni. the following variables are used.
r$xsc                pointer to scblk for function arg
xsofs                offset (num chars scanned so far)
(wa)                 non-zero to skip and trim blanks
(wc)                 delimiter one (ch$xx)
(xl)                 delimiter two (ch$xx)
jsr  xscan           call to scan next item
(xr)                 pointer to scblk for token scanned
(wa)                 completion code (see below)
(wc,xl)              destroyed
the scan starts from the current position and continues
until one of the following three conditions occurs.
1)   delimiter one is encountered  (wa set to 1)
2)   delimiter two encountered  (wa set to 2)
3)   end of string encountered  (wa set to 0)
the result is a string containing all characters scanned
up to but not including any delimiter character.
the pointer is left pointing past the delimiter.
if only one delimiter is to be detected, delimiter one
and delimiter two should be set to the same value.
in the case where the end of string is encountered, the
string includes all the characters to the end of the
string. no further calls can be made to xscan until
xscni is called to initialize a new argument scan
```

```
    xscan (continued)
xscan   prc   e,0                  entry point
        mov   wb,xscwb             preserve wb
        mov   wa,-(xs)             record blank skip flag
        mov   wa,-(xs)             and second copy
        mov   r$xsc,xr             point to argument string
        mov   sclen(xr),wa         load string length
        mov   xsofs,wb             load current offset
        sub   wb,wa                get number of remaining characters
        bze   wa,xscn3             jump if no characters left
        plc   xr,wb                point to current character
    loop to search for delimiter
xscn1   lch   wb,(xr)+             load next character
        beq   wb,wc,xscn4          jump if delimiter one found
        beq   wb,xl,xscn5          jump if delimiter two found
        bze   (xs),xscn2           jump if not skipping blanks
        icv   xsofs                assume blank and delete it
```

---

if **.caht**

```
        beq   wb,=ch$ht,xscn2      jump if horizontal tab
```

fi

---

if **.cavt**

```
        beq   wb,=ch$vt,xscn2      jump if vertical tab
```

fi

```
        beq   wb,=ch$bl,xscn2      jump if blank
        dcv   xsofs                undelete non-blank character
        zer   (xs)                 and discontinue blank checking
    here after performing any leading blank trimming.
xscn2   dcv   wa                   decrement count of chars left
        bnz   wa,xscn1             loop back if more chars to go
    here for runout
xscn3   mov   r$xsc,xl             point to string block
        mov   sclen(xl),wa         get string length
        mov   xsofs,wb             load offset
        sub   wb,wa                get substring length
        zer   r$xsc                clear string ptr for collector
        zer   xscrt                set zero (runout) return code
        brn   xscn7                jump to exit
```

```
    xscan (continued)
    here if delimiter one found
xscn4   mov  =num01,xscrt               set return code
        brn  xscn6                      jump to merge
    here if delimiter two found
xscn5   mov  =num02,xscrt               set return code
    merge here after detecting a delimiter
xscn6   mov  r$xsc,xl                   reload pointer to string
        mov  sclen(xl),wc               get original length of string
        sub  wa,wc                      minus chars left = chars scanned
        mov  wc,wa                      move to reg for sbstr
        mov  xsofs,wb                   set offset
        sub  wb,wa                      compute length for sbstr
        icv  wc                         adjust new cursor past delimiter
        mov  wc,xsofs                   store new offset
    common exit point
xscn7   zer  xr                         clear garbage character ptr in xr
        jsr  sbstr                      build sub-string
        ica  xs                         remove copy of blank flag
        mov  (xs)+,wb                   original blank skip/trim flag
        bze  sclen(xr),xscn8            cannot trim the null string
        jsr  trimr                      trim trailing blanks if requested
    final exit point
xscn8   mov  xscrt,wa                   load return code
        mov  xscwb,wb                   restore wb
        exi                             return to xscan caller
        enp                             end procedure xscan
```

```
    xscni -- execution function argument scan
    xscni initializes the scan used for prototype arguments
    in the clear, define, load, data, array functions. see
    xscan for the procedure which is used after this call.
    -(xs)                  argument to be scanned (on stack)
    jsr  xscni             call to scan argument
    ppm  loc               transfer loc if arg is not string
    ppm  loc               transfer loc if argument is null
    (xs)                   popped
    (xr,r$xsc)             argument (scblk ptr)
    (wa)                   argument length
    (ia,ra)                destroyed
xscni   prc   n,2                                   entry point
        jsr   gtstg                                 fetch argument as string
        ppm   xsci1                                 jump if not convertible
        mov   xr,r$xsc                              else store scblk ptr for xscan
        zer   xsofs                                 set offset to zero
        bze   wa,xsci2                              jump if null string
        exi                                         return to xscni caller
    here if argument is not a string
xsci1   exi   1                                     take not-string error exit
    here for null string
xsci2   exi   2                                     take null-string error exit
        enp                                         end procedure xscni
```

spitbol–stack overflow section

```
    control comes here if the main stack overflows
        sec                                           start of stack overflow section
        add   =num04,errft                            force conclusive fatal error
        mov   flptr,xs                                pop stack to avoid more fails
        bnz   gbcfl,stak1                             jump if garbage collecting
        erb   gbcfl,stak1                             jump if garbage collecting
    no chance of recovery in mid garbage collection
stak1   mov   =endso,xr                               point to message
        zer   kvdmp                                   memory is undumpable
        brn   stopr                                   give up
```

spitbol–error section

```
this section of code is entered whenever a procedure
return via an err parameter or an erb opcode is obeyed.
(wa)                is the error code
the global variable stage indicates the point at which
the error occured as follows.
stage=stgic         error during initial compile
stage=stgxc         error during compile at execute
                    time (code, convert function calls)
stage=stgev         error during compilation of
                    expression at execution time
                    (eval, convert function call).
stage=stgxt         error at execute time. compiler
                    not active.
stage=stgce         error during initial compile after
                    scanning out the end line.
stage=stgxe         error during compile at execute
                    time after scanning end line.
stage=stgee         error during expression evaluation
```

```
        sec                                   start of error section
error   beq  r$cim,=cmlab,cmple               jump if error in scanning label
        mov  wa,kvert                         save error code
        zer  scnrs                            reset rescan switch for scane
        zer  scngo                            reset goto switch for scane
```

---

if **.cpol**

```
        mov  =num01,polcs                     reset poll count
        mov  =num01,polct                     reset poll count
```

fi

```
        mov  stage,xr                         load current stage
        bsw  xr,stgno                         jump to appropriate error circuit
        iff  stgic,err01                      initial compile
        iff  stgxc,err04                      execute time compile
        iff  stgev,err04                      eval compiling expr.
        iff  stgee,err04                      eval evaluating expr
        iff  stgxt,err05                      execute time
        iff  stgce,err01                      compile - after end
        iff  stgxe,err04                      xeq compile-past end
        esw                                   end switch on error type
```

```
    error during initial compile
    the error message is printed as part of the compiler
    output. this printout includes the offending line (if not
    printed already) and an error flag under the appropriate
    column as indicated by scnse unless scnse is set to zero.
    after printing the message, the generated code is
    modified to an error call and control is returned to
    the cmpil procedure after resetting the stack pointer.
    if the error occurs after the end line, control returns
    in a slightly different manner to ensure proper cleanup.
err01   mov  cmpxs,xs                        reset stack pointer
        ssl  cmpss                           restore s-r stack ptr for cmpil
        bnz  errsp,err03                     jump if error suppress flag set
```

---

if **.cera**

---

if **.csfn**

```
        mov  cmpsn,wc                        current statement
        jsr  filnm                           obtain file name for this statement
```

fi

```
        mov  scnse,wb                        column number
        mov  rdcln,wc                        line number
        mov  rdcln,wc                        line number
        jsr  sysea                           advise system of error
        ppm  erra3                           if system does not want print
        mov  xr,-(xs)                        save any provided print message
```

fi

```
        mov  erich,erlst                     set flag for listr
        jsr  listr                           list line
        jsr  prtis                           terminate listing
        zer  erlst                           clear listr flag
        mov  scnse,wa                        load scan element offset
        bze  wa,err02                        skip if not set
```

---

if **.caht**

```
        lct  wb,wa                           loop counter
        icv  wa                              increase for ch$ex
        mov  r$cim,xl                        point to bad statement
        jsr  alocs                           string block for error flag
        mov  xr,wa                           remember string ptr
        psc  xr                              ready for character storing
        plc  xl                              ready to get chars
    loop to replace all chars but tabs by blanks
erra1   lch  wc,(xl)+                        get next char
        beq  wc,=ch$ht,erra2                 skip if tab
        mov  =ch$bl,wc                       get a blank
```

```
    merge to store blank or tab in error line
erra2   sch   wc,(xr)+                    store char
        bct   wb,erra1                    loop
        mov   =ch$ex,xl                   exclamation mark
        sch   xl,(xr)                     store at end of error line
        csc   xr                          end of sch loop
        mov   =stnpd,profs                allow for statement number
        mov   wa,xr                       point to error line
        jsr   prtst                       print error line
else
        mti   prlen                       get print buffer length
        mfi   gtnsi                       store as signed integer
        add   =stnpd,wa                   adjust for statement number
        mti   wa                          copy to integer accumulator
        rmi   gtnsi                       remainder modulo print bfr length
        sti   profs                       use as character offset
        mov   =ch$ex,wa                   get exclamation mark
        jsr   prtch                       generate under bad column
fi
    here after placing error flag as required
err02   jsr   prtis                       print blank line
```

```
if .cera
        mov   (xs)+,xr                    restore any sysea message
        bze   xr,erra0                    did sysea provide message to print
        jsr   prtst                       print sysea message
fi
erra0   jsr   ermsg                       generate flag and error message
        add   =num03,lstlc                bump page ctr for blank, error, blk
erra3   zer   xr                          in case of fatal error
        bhi   errft,=num03,stopr          pack up if several fatals
    count error, inhibit execution if required
        icv   cmerc                       bump error count
        add   cswer,noxeq                 inhibit xeq if -noerrors
        bne   stage,=stgic,cmp10          special return if after end line
```

```
    loop to scan to end of statement
err03   mov  r$cim,xr                         point to start of image
        plc  xr                               point to first char
        lch  xr,(xr)                          get first char
        beq  xr,=ch$mn,cmpce                  jump if error in control card
        zer  scnrs                            clear rescan flag
        mnz  errsp                            set error suppress flag
        jsr  scane                            scan next element
        bne  xl,=t$smc,err03                  loop back if not statement end
        zer  errsp                            clear error suppress flag
    generate error call in code and return to cmpil
        mov  *cdcod,cwcof                     reset offset in ccblk
        mov  =ocer$,wa                        load compile error call
        jsr  cdwrd                            generate it
        mov  cwcof,cmsoc(xs)                  set success fill in offset
        mnz  cmffc(xs)                        set failure fill in flag
        jsr  cdwrd                            generate succ. fill in word
        brn  cmpse                            merge to generate error as cdfal
    error during execute time compile or expression evaluatio
    execute time compilation is initiated through gtcod or
    gtexp which are called by compile, code or eval.
    before causing statement failure through exfal it is
    helpful to set keyword errtext and for generality
    these errors may be handled by the setexit mechanism.
err04   bge  errft,=num03,labo1               abort if too many fatal errors
```

---

if **.cpol**

```
        beq  kvert,=nm320,err06               treat user interrupt specially
```

fi

```
        zer  r$ccb                            forget garbage code block
        mov  *cccod,cwcof                     set initial offset (mbe catspaw)
        ssl  iniss                            restore main prog s-r stack ptr
        jsr  ertex                            get fail message text
        dca  xs                               ensure stack ok on loop start
    pop stack until find flptr for most deeply nested prog.
    defined function call or call of eval / code.
erra4   ica  xs                               pop stack
        beq  xs,flprt,errc4                   jump if prog defined fn call found
        bne  xs,gtcef,erra4                   loop if not eval or code call yet
        mov  =stgxt,stage                     re-set stage for execute
        mov  r$gtc,r$cod                      recover code ptr
        mov  xs,flptr                         restore fail pointer
        zer  r$cim                            forget possible image
```

---

if **.cinc**

```
        zer  cnind                            forget possible include
```

fi

```
    test errlimit
errb4   bnz  kverl,err07                      jump if errlimit non-zero
        brn  exfal                            fail
    return from prog. defined function is outstanding
errc4   mov  flptr,xs                         restore stack from flptr
        brn  errb4                            merge
```

```
    error at execute time.
    the action taken on an error is as follows.
    if errlimit keyword is zero, an abort is signalled,
    see coding for system label abort at l$abo.
    otherwise, errlimit is decremented and an errtype trace
    generated if required. control returns either via a jump
    to continue (to take the failure exit) or a specified
    setexit trap is executed and control passes to the trap.
    if 3 or more fatal errors occur an abort is signalled
    regardless of errlimit and setexit - looping is all too
    probable otherwise. fatal errors include stack overflow
    and exceeding stlimit.
err05   ssl   iniss                 restore main prog s-r stack ptr
        bnz   dmvch,err08           jump if in mid-dump
    merge here from err08 and err04 (error 320)
err06   bze   kverl,labo1           abort if errlimit is zero
        jsr   ertex                 get fail message text
    merge from err04
err07   bge   errft,=num03,labo1    abort if too many fatal errors
        dcv   kverl                 decrement errlimit
        mov   r$ert,xl              load errtype trace pointer
        jsr   ktrex                 generate errtype trace if required
        mov   r$cod,wa              get current code block
        mov   wa,r$cnt              set cdblk ptr for continuation
        scp   wb                    current code pointer
        sub   wa,wb                 offset within code block
        mov   wb,stxoc              save code ptr offset for scontinue
        mov   flptr,xr              set ptr to failure offset
        mov   (xr),stxof            save failure offset for continue
        mov   r$sxc,xr              load setexit cdblk pointer
        bze   xr,lcnt1              continue if no setexit trap
        zer   r$sxc                 else reset trap
        mov   =nulls,stxvr          reset setexit arg to null
        mov   (xr),xl               load ptr to code block routine
        bri   xl                    execute first trap statement
    interrupted partly through a dump whilst store is in a
    mess so do a tidy up operation. see dumpr for details.
err08   mov   dmvch,xr              chain head for affected vrblks
        bze   xr,err06              done if zero
        mov   (xr),dmvch            set next link as chain head
        jsr   setvr                 restore vrget field
    label to mark end of code
s$yyy   brn   err08                 loop through chain
```

spitbol–here endeth the code

`end of assembly`

```
    end                                   end macro-spitbol assembly
```