Portability Techniques for BLISS Programs

Bruce W. Leverett
Paul J. Knueven
Peter G. Hibbard

November, 1977

Abstract

The problems of writing large systems in BLISS that can be run in more than one environment are described. A method (a set of methods) for attacking these problems is explained, with examples of its use in a particular system (a compiler). Aspects of the BLISS language are discussed with regard to their usefulness or uselessness in solving these problems.

The work described here was supported by the Defense Advanced Research Projects Agency under contract F44620-73-C-0074, monitored by the Air Force Office of Scientific Research.

In 1974, the authors undertook to write a compiler for ALGOL 68. The compiler was to run on a PDP-11 and produce code suitable for a PDP-11. The first version was to run on C.mmp[1], a multiprocessor system comprising several (slightly modified) PDP-11 processors and other hardware, under the HYDRA[2] operating system.

The compiler was written in BLISS[3], for the usual variety of reasons, not least of which was that the C.mmp/HYDRA system offers a complete symbolic debugging system (SIX12[4]) and other support for BLISS programs. However, there were some initial problems with building a system in this environment.

- The environment itself was unstable. The hardware and the various levels of the operating system were full of bugs, and subject to frequent redesign as well.

- The environment was completely unfamiliar to the authors, who had never used it before.

- The compiler and linker for BLISS did not run on C.mmp/HYDRA, but on a PDP-10/TOPS-10, with no easily available link between the two. Thus changing the compiler involved loading a linked version onto a DECtape, carrying the tape over to C.mmp, mounting it there, and reading it--a time-consuming process. (This, incidentally, was one of the problems that the ALGOL 68 system was intended to solve, since it was to run entirely under C.mmp/HYDRA.)

Hence, the authors decided to build a preliminary version of the compiler, which would run entirely under PDP-10/TOPS-10 (suffering from none of the above three problems), and do most of the early development work on the preliminary version. To minimize the problems of changing over from PDP-10 versions to C.mmp versions and *vice versa*, the preliminary version was also written in BLISS. We wished to be able to switch from using either version to using the other, simply by making a small number of error-proof changes, and then running the appropriate compiler and linker to produce the version to be used. We completely met this goal, and in fact switched from using one version to using the other several times. In addition, we met it in a way that was very easy to maintain: although we had not used the PDP-10 version since April 1976, we used the procedure outlined above to create a PDP-10 version in February 1977, with essentially no problems (e.g. all files compiled correctly, in spite of language differences between BLISS-10 and BLISS-11).

Achieving this level of portability was a non-trivial task. BLISS is not as high-level a language as ALGOL nor as standardized as FORTRAN (though achieving portability even in FORTRAN is not always easy [5]). The machine-dependence of the design of even the simplest data structures is not hidden from the programmer; moreover, I/O and other interactions with the operating system are not part of the language, and must be supplied by the user's program (or at least in separately-compiled programs). Standard packages are available for handling routine problems for both BLISS-10/TOPS-10 and BLISS-11/HYDRA, but there are several problems with using these:

- o They are not compatible with each other.
- o They (especially the BLISS-11/HYDRA package) are inadequate for the needs of large programs such as compilers.

- There are no packages for any PDP-11 operating systems other than HYDRA, RSX-11, and RT-11, and even these three are not compatible with each other.

This paper describes the techniques we used to achieve portability, i.e. to minimize the work necessary to change over from one version to the other. Chapter 1 describes our system for isolating the differences between the architecures of the PDP-10 and the PDP-11. Chapter 2 describes our treatment of the differences between the BLISS-10 and BLISS-11 languages. Chapter 3 describes the compiler interface with the operating system, i.e. the system for isolating the differences between TOPS-10 and HYDRA. In Chapter 4 we draw some conclusions about the general applicability of our techniques, and about the design of system implementation languages (such as BLISS) with portability in mind. Portions of the compiler are included in the Appendices for illustration.

1. Hardware Differences

By using a high-level language, we were able to avoid having to deal with differences between the instruction sets and instruction formats of the PDP-10 and the PDP-11 (the few exceptions are noted below). However, because of the nature of the BLISS language, we could not avoid dealing with differences between their data formats.

Our method of isolating these differences was to use a "REQUIRE file". BLISS has a declaration similar in purpose to the PL/I %INCLUDE statement, by means of which a program may incorporate several files as part of its source text; this is the REQUIRE declaration. Any particular separately-compiled module of the compiler begins with a series of REQUIRE declarations, each of which names a file containing some set of related definitions. The very first such file is always SYSPRM.REQ, which contains the definitions which hide the differences between the PDP-10/BLISS-10 and the PDP-11/BLISS-11 systems. The 10 version of this file is B10PRM.REQ; the 11 version is B11PRM.REQ. The first step in the changeover from one version to the other, say from the 10 version to the 11 version, is to make sure that SYSPRM.REQ is a copy of the proper version of this file (in this case B11PRM.REQ). This requires a single command to the file system.

Not all of the differences between language/machine systems could be handled by a single file of definitions, however. Those that could not may be characterized as differences between libraries. Neither of the two BLISS compilers generates code to support unsigned division, conversion of strings of characters to fixed- or floating-point numbers (or to BITS values, for ALGOL 68), or the SIGNAL/ENABLE feature of BLISS-11. These must be handled by library subroutines, and the libraries, even those which can be written in BLISS, are of a very low-level, machine-dependent nature. The two libraries are kept as separate groups of source files, and the differences in the interfaces to them are hidden in SYSPRM.REQ.

A few words of explanation of BLISS are in order before we describe the contents of SYSPRM.REQ. The data structures commonly available in other high-level languages, such as records, stacks (except for the control stack), arrays, list cells, and so forth, are not built into BLISS. To use such a structure, a program must define it, by defining

- the layout in memory of each instance of the structure. This includes a list of the parameters (e.g. dimensions) of any declaration of a variable with that structure, and specification of how those parameters are used (if at all) in determining the amount of storage allocated to that variable.

- the method by which the fields of the structure are accessed. A particular field need not be an entire machine word; then it must be specified which group of bits within the word it must be.

For instance, the standard definition of a one-dimensional zero-based array of one-word values (in BLISS-10; it is slightly different in BLISS-11) is as follows:

```
structure Vector [Index] = [Index] (.Vector + .Index) <0, 36>;
```

The first pair of square brackets indicates that both declarations and accesses of Vector variables have one parameter. The expression in the second pair of square brackets indicates how large an array is allocated to such a variable when it is declared; in this case, the number of words allocated is equal to the parameter ("Index") used in the declaration. (For instance, a declaration like

```
local Vector QQQQ [445];
```

causes the array QQQQ to have length 445 words.) The remaining expression indicates how this structure is to be accessed: the location of the base variable and the value of the index are summed to get an address, and the 36 bits starting at bit 0 of the word at that address (that is, the whole word) are referred to.

The Vector structure happens to be predefined, but all other structures must be defined by the user's program. The definition of this structure in BLISS-11 is slightly different from its definition in BLISS-10, because of the different data formats of the two machines: one can declare vectors of bytes or vectors of full words, and thus take advantage of either the byte addressing or the word addressing of the PDP-11.

This background helps to explain the organization of SYSPRM.REQ. The definitions in this file which hide differences between the PDP-10 and the PDP-11 can be grouped into four categories:

1. the definitions of "standard structures". These are:

   BYTVECT - array of 8-bit bytes
   CHARVECT - array of ASCII (7-bit) characters
   HFVECT - array of half words
   HFTABLE - array of half words, from which subfields may be accessed

2. the definitions of machine characteristics. These are occasionally useful even outside the definitions of the standard structures. They are:

	PDP-10	PDP-11	
QADRINC	1	2	- address increment from one full word to the next

QCHSWRD	5	2	- # of ASCII characters packed into one full word
QADRSIZ	18	16	- width in bits of a pointer
QWRDSIZ	36	16	- width in bits of a full word
QHFSIZ	18	8	- width in bits of a half word
QSPARE	35	1	- position within a word of a bit which, if the word contains a pointer, is not relevant (by convention) to that pointer. For the PDP-11, this is the lowest order bit of the word, since full words are aligned on even boundaries, and pointers to them always have zero in the lowest bit. For the PDP-10, this may be any of the bits in the high order half of the word, since by convention, pointers are always held in the lower half.

3. the definition of the packed-string data type. This is a set of definitions, primarily of operators, for manipulating strings of characters. What makes these definitions interesting--they are the only section of SYSPRM.REQ which underwent a major design iteration--is the crucial difference between the basic string operations "read (write) a character and step to the next character", on the two machines. On the PDP-10, these operations (the IDPB and ILDB instructions) move to the next character before reading or writing; on the PDP-11, the corresponding operations (the autoincrement addressing mode) cause reading or writing before moving to the next character. This difference cannot be hidden without imposing some restrictions on the operations which may be performed on strings. We therefore divide string pointers into two categories: those for which every access of a character is accompanied by a step from one character to the next ("I-pointers") and those for which stepping to the next character must always be done separately ("N-pointers").

4. the definitions of two "library" routines which happen to be coded as macros, rather than as closed subroutines. These are the routines to clear out a block of core and to copy one block of core into another.

2. Language Differences

Neither BLISS-11 nor BLISS-10 is a proper superset of the other. Generally, however, we did not make much use of features of one language which were not present in the other, and the list of such features for which we put fake definitions in the versions of SYSPRM.REQ is uninteresting. However, three exceptions should be noted:

- The BLISS-11 SIGNAL/ENABLE construct caught our fancy, and we used it for compilation error handling. Therefore, we were obliged to write BLISS-10 constructs to simulate it. We were surprised to find that this was possible. The BLISS-11 library file, SIGENB.MAC, which is a standard part of the BLISS-11 compiler distribution, corresponds to the BLISS-10 library file, SYSDEP.MAC

(which contains some miscellaneous material not related to SIGNAL/ENABLE as well). The only problem with our simulation of SIGNAL/ENABLE is that some extra code is required on normal termination of a block containing an ENABLE declaration, to pop a frame off the (conceptual) "ENABLE stack". The BLISS-11 compiler outputs this code automatically, but of course the BLISS-10 compiler does not; we put the code in the macro ENABEND, and we enforce the convention that every path to exit of an ENABLEd block must end with an ENABEND, to get around this problem.

- We wished to set up extensive preloaded symbol tables with the BLISS PLIT facility. We stumbled over one characteristic of these symbol tables, namely, that they contained many pointers to themselves and each other. The BLISS-10 NAMES and GLOBALLY NAMES feature allowed us to generate the necessary pointers easily and cleanly, but there is no corresponding feature in BLISS-11. Therefore we wrote a set of iterated and recursive macros which simulated these, as well as the related INDEXES and GLOBALLY INDEXES feature. The simulation is not quite complete, because the iterated/recursive macro processor attempts to update an index into a PLIT that it is building by counting the arguments which fly by it; it doesn't understand other features of the PLIT facility, such as duplication factors and strings. However, we did not need to make use of these features in our symbol tables.

- The BLISS-10 and BLISS-11 macro processors do not expand macro arguments in quite the same way. From our point of view, the problem is this: Let there be a macro A, with a formal parameter FA, and a macro B, with possibly some formal parameters. Suppose there is a call such as

  ```
  A ( B (...) )
  ```

  and the call of B results in a string containing a comma--that is, a comma which is not hidden by being between a matched pair of parentheses or brackets of some sort. The BLISS-11 macro processor completes the expansion of the call of B while setting up the call of A, and since it regards the comma as an argument separator, it concludes that A has (wrongly) been passed two actual parameters, instead of just one. The BLISS-10 macro processor does not do this, treating the entire call of B as a single parameter to A.

  In the ordinary course of programming, we did not encounter such problems, because of our generally conservative use of macros. However, the slightly bizarre tricks we used in the set of macros which set up the table of productions tripped across exactly this problem. We solved it by "quoting" the comma which was generated, in the BLISS-11 version, so that the BLISS-11 macro processor would not recognize it as an argument separator; this is the origin of the macro "quoted" which is defined in SYSPRM.REQ.

3. Operating System Differences

Because more than 20 complete operating systems are in use on PDP-11's and configurations of PDP-11 processors, the problem of isolating the operating system interface (or Operating System Environment, OSE) would have to have been faced even if we had not had to write a PDP-10 version of the compiler. It almost goes without saying that this problem cannot be solved by a single "REQUIRE file" of useful definitions. The TOPS-10 OSE consists of 2K words of code distributed over 2 source files (and some files from a Bliss-10 library), and the HYDRA OSE is about half as large. Thus the OSE constitutes a separate subdivision of the compiler, and, as with other subdivisions, the interface between the OSE and the rest of the compiler has been redefined several times.

Originally we thought of the OSE as a collection of utility subroutines; the compiler at the top level was a system-independent controller consisting, like many compilers and other file processing programs under TOPS-10, of an infinite loop:

```
DO
  <read input/output file specifications>
  <create object file from source file>
OD
```

Eventually we realized that the top level of the compiler must itself be part of the OSE. The system-independent portion of the compiler is, at its top level, a subroutine which compiles a single Algol source program. This seems to be the largest unit of computation which all possible OSE's can deal with, ranging from the simplest paper-tape load-and-go systems for bare PDP-11's to complete operating subsystems with their own text editors, linkers, and version backup systems. Thus the HYDRA OSE consists of:

(1) A set of utility subroutines, invoked from various points in the system-independent part of the compiler. These are:

(1a) OSEsrcchar — Reads a single character from the source file, and returns the character as its value. End-of-file is denoted by a special character. Note that end-of-file here means end-of-logical-source-program; for instance, in the TOPS-10 OSE (and perhaps in future HYDRA OSE's) the source program compiled in a single compilation may stretch over several of the entities that the operating system knows of as "files".

(1b) OSElstline — Sends a single line of text off to be listed. The system-independent portion of the compiler believes that there are two places to which text output may be sent: a "listing device" and a "command device", corresponding in many systems to a line printer and a user's terminal, respectively. Thus one of the arguments to this routine is an indicator of which place(s) the line is to be sent to.

(1c) OSEerrmsg — Outputs a single error message. The sole argument to this

routine is a number indicating which of many compilation errors occurred. In some OSE's, such as that for TOPS-10, this routine consists only of one or more invocations of OSElstline. It is included in the OSE itself because, in general, the method of obtaining the error message text to be output is expected to be system-dependent. For instance, in the HYDRA OSE, the error messages are not normally kept in addressable core; this routine makes them addressable temporarily.

(1d) OSEobjword — Outputs a single word of code to the object file. This would be too small a unit of output if the compiler had to deal with different loader formats for such things as fixups, relocatable segments, and overlay structures, but we have avoided those by writing our own loader. OSEobjword is a null routine in the TOPS-10 version.

(2) A "top level". This opens various standard channels for I/O, performs other system-dependent initializations (e.g. initializing the symbolic debugging system), and calls the system-independent compiler. In load-and-go systems, this top level might start the user's program when the system-independent compiler had finished. (Alternatively, this might be more conveniently left to the operating system itself.) The current HYDRA OSE top level also includes

(2a) A routine to handle aborted compilations. The system-independent compiler may find itself unable to continue due to such things as internal errors, lack of core, or I/O failures; in such cases it returns to its caller, with a result value indicating the cause of failure.

(2b) Routines to initialize the settings of various compilation options, based on information supplied by the user to the operating system and passed by the operating system to the OSE.

(2c) A routine to clear out a workspace in core for the compiler; this initialization must be in the OSE, because different versions may have completely different arrangements in memory of the compiler's code, workspace, and internal control stack.

(3) A set of global variables, which serve as an additional means of communication between the compiler and the OSE. They are:

(3a) GBLerrs — Count of ordinary compilation errors.

(3b) GBLwarns — Count of "mild" compilation errors--those which do not prevent the compiler from generating code.

(3c) GBLfreelo — Points to the start of the compiler's workspace.

(3d) GBLfreehi — Points to the end of the compiler's workspace.

(3e) GBLprogstart — An indication of the starting address of the compiled program.

(3f) GBLpragflags — The area (currently a single word) in which all the compilation option settings are kept.

4. Conclusions

There is a natural conflict between, on the one hand, the universal language design goal of portability and, on the other hand, one of the goals of system implementation languages (SIL's) in particular, namely that the programmer should be able to make use of all the facilities of the hardware, such as unusual instructions. The designers of PL/360[6] have resolved this conflict by sacrificing portability altogether. The designers of MARY[7] describe a subset of the language, "safe MARY", and a set of features which extend it to "unsafe MARY"; programs written entirely in the subset are guaranteed to be portable from one computer to another. The compiler can enforce the restriction to the subset, if the programmer desires. The approach taken by BLISS is not quite so strict as this. The designers regarded the various dialects of BLISS as "a class of languages that are similar in philosophy and that mirror a similar concern for the important aspects of systems programming, but each of which is tailored to its host machine[3]."

Usually the requirements of portability cannot be ignored. The more successful SIL's are bound to be implemented on more than one kind of computer, and for all but the lowest-level, most hardware-dependent programs (device handlers, diagnostics, and so forth) it will always be attractive to copy a program already written, even from another computer, rather than to write a new program from scratch. Therefore it is useful to consider what aspects of BLISS we found helpful in constructing a portable program, and what aspects were useless or even harmful.

It is evident that our principal tools, namely the REQUIRE feature and the practice of keeping the two libraries in separate groups of files, are extremely powerful; they could be used, and were used, as a "last resort" in the solution of portability problems, when there was no way to avoid writing two different (machine-dependent) versions of some routine or group of routines. The effectiveness of our techniques is therefore judged by the extent to which we had to use the "last resort", i.e. by the size of the libraries. Indirectly we also judge the suitability of BLISS for writing portable programs by this criterion.

In this respect the STRUCTURE definition facility has been outstandingly successful. The string data-type is an example of what has been done with this. Strings are represented in completely different ways on the two machines: packed 5 characters to a word on the PDP-10, aligned on byte (half word) boundaries on the PDP-11. But by the use of the CHARVECT structure, together with a small set of operations based on the BLISS-10 "special functions" SCANN, REPLACEI, etc., we have made string manipulation completely uniform, so that no library routines whatever had to be written for it. Perhaps an even more striking example is that of the structures PBLOCK and OBLOCK, the standard structures onto which pointers to blocks of packed data are mapped. These did not even have to be defined in SYSPRM.REQ, and thanks to the macros RH, LH, WORDF, ADDRF, and SPCF, the definitions of fields in the various types of blocks need not be placed in the machine-dependent REQUIRE file either.

The LINKAGE definition facility should also be mentioned although we have so far made little use of it. By defining a linkage, such as REGO, one imposes special calling

conventions (e.g. special locations for actual parameters) on all routines which are declared with that linkage. It is particularly well designed for portability, in the following respects:

- Code which makes use of LINKAGE definitions and is correct is guaranteed to be correct when the use of special linkages is removed.

- The use of special linkages is particularly easy to remove. That is, the "fake definitions" which one must put in one version, corresponding to the linkage definitions and uses of another version, are extremely simple. The definition of REG0 in the BLISS-10 version, our only example, is a null macro.

Two features of BLISS which we had to essentially ignore (except in the construction of libraries) were the ability to descend into machine code (the MACHOP feature of BLISS-10, the OPCODE feature of BLISS-11) and the ability to force local variables to be allocated to the machine's fast registers (the REGISTER declaration). It is worth asking why we could not make use of the second of these.

The alternative to the REGISTER declaration, in both versions of BLISS, is the LOCAL declaration. The choice between these two alternatives is not quite the same in BLISS-10 as it is in BLISS-11. In the former, LOCAL variables are always allocated on the control stack, but in the code that is generated, copies of the variables are often put in registers to allow easier access to them. In the latter, any LOCAL variable may be allocated either to a register or to a location on the stack, depending on the whim of the compiler, which attempts to find an optimal allocation. In both cases, the usefulness of declaring a variable as REGISTER instead of LOCAL in order to make frequent accesses to it less expensive is clouded or even nullified by the actions of the compiler. Thus REGISTER declarations are primarily useful only when they are necessary, e.g. in conjunction with use of the MACHOP (OPCODE) feature.

This paper has not covered two aspects of the question of portability which may be of general interest. The first of these is portability between really different languages. Although a version of the compiler has been produced principally by transcribing the BLISS source code into another (unrelated) higher-level language (PASCAL), we were not sufficiently familiar with that effort to discuss the questions raised by it in this paper.

The second of these is portability of the compiler to installations where BLISS compilers are not available. Since the BLISS-11 compiler produces assemblable PDP-11 code, it is possible to export the PDP-11 version of the compiler in the form of a set of assemblable files. But this leaves little room for the user of the system to make local modifications to it, such as additions of new language features, or modifications to the compiler's pre-initialized symbol tables; it is difficult or impossible in most cases to follow the assembly code produced by the BLISS-11 compiler, since it is, after all, completely uncommented and unformatted, and labels in the assembly code do not correspond in general to labels in the BLISS-11 source code. We have not yet attacked the problems raised by the general unavailability of BLISS-11 compilers at user sites, and so we postpone the discussion of them to future articles.

References

[1] Wulf, W.A., and Bell, C.G., "C.mmp--a Multi-mini-processor", Proc. AFIPS 1972 FJCC, Vol. 41, AFIPS Press, Montvale, N.J., pp. 765-777.

[2] Wulf, W.A., Cohen, E., Corwin, W., Jones, A., Levin, R., Pierson, C., and Pollack, F., "HYDRA: The Kernel of a Multiprocessor Operating System", Comm. ACM *17*, 6 (June 1974), pp. 337-345.

[3] Wulf, W.A., Russell, D.B., and Habermann, A.N., "BLISS: A Language for Systems Programming", Comm. ACM *14*, 12 (Dec. 1971), pp. 780-790.

[4] Reiner, A., and Newcomer, J. (eds.), "The HYDRA Users' Manual", technical report, Dept. of Computer Science, Carnegie-Mellon University, August 1977.

[5] Sabin, M.A., "Portability--Some Experiences with FORTRAN", Software Practice and Experience *6*, 3 (July-Sept. 1976), pp. 393-396.

[6] Wirth, N., "PL360, A Programming Language for the 360 Computers", J. ACM *15*, 1 (Jan. 1968), pp. 37-74.

[7] Conradi, R., and Holager, P., "MARY Textbook", Regnesentret ved Universitetet, Trondheim, Norway, 1974.

Appendix

The following files are included here in their entirety:

B10PRM.REQ - PDP10 version of SYSPRM.REQ
B11PRM.REQ - PDP11 version of SYSPRM.REQ
COMMON.REQ - REQUIRE file used by every module in the compiler

SYSDEP.MAC - support routines for BLISS-10 SIGNAL/ENABLE
SIGENB.MAC - support routines for BLISS-11 SIGNAL/ENABLE

A68S10.B10 - heart of the TOPS-10 OSE
A68S11.B11 - heart of the HYDRA OSE

MDSTRC.REQ - typical REQUIRE file
LEXAN .BLI - representative code module
SRTABS.BLI - representative use of the DATAAREA macros

```
! B10PRM.REQ    (must be copied to SYSPRM.REQ)
!

! Copyright 1977        P. Hibbard and P. Knueven
!                       Pittsburgh, Pennsylvania


! System-dependent declarations for the Bliss 10 version.

bind
        Qadrinc = 1,    ! Increment from word address to next word address
        Qadrsiz = 18,   ! Number of bits in an address value (pointer)
        Qchswrd = 5,    ! Number of characters per word
        Qhfsiz  = 18,   ! Number of bits per half-word
        Qhtab   = 1,    ! True iff horizontal tabs may be sent to output device
        Qspare  = 35,   ! Must be 0 or Qwrdsiz-1
        Qwrdsiz = 36;   ! Number of bits per word

!
! Macros to supply actual names of the Operating System Environment
!   routines and globals.
!
macro
        GBLerrs         = ?.errs $,
        GBLwarns        = ?.warns $,
        GBLprogflags    = ?.pragf $,
        GBLprogstart    = ?.progs $,
        GBLfreehi       = ?.freeh $,
        GBLfreelo       = ?.freel $,

        OSEsrcchar      = ?.SRCch $,
        OSElstline      = ?.LSTli $,
        OSEobjword      = ?.OBJwo $,
        OSEerrmsg       = ?.Errms $;

!
! Macros to smooth over the differences in the Bliss-10 syntax and
!   the Bliss-11 syntax.
!
macro   signal(e)=
            begin external ?.SIGNL;
            (.Freg-1)<0,36>←?.SIGNL<0,0>;
            return e
            end $,
        enable=
            register R[5];      ! Forces all declarable regs to be preserved.
            local ?.Frame[4];
            external ?.ENABL,Sigval,Sigreg;
            Vreg ← (?.Frame-1)<0,0>;
            ?.ENABL();
            if .Vreg eqi 0
                then 0
                else exitblock select .Sigval of
                     nset $,

        elbanem
                        ;
```

```
                    Otherwise: signal(.Sigval)
                   tesn $,
        enabend=
            Sigreg←@(.Sigreg-2) $,
        enableave=              ! The programmer is cautioned against
            enabend; leave $,   ! using enableave or enabreturn to
        enabreturn=             ! exit more than one level of ENABLEd
            enabend; return $,  ! blocks; an extra enabend is required
                                ! for each level.
        byte    = $,
        quoted  = $,
        uplit   = plit $,
        unames  = list $,
        stacklocal = local $,
        Reg0    = $,
        maxi(A,B) = (if (A) geq (B) then (A) else (B)) $,
        mini(A,B) = (if (A) leq (B) then (A) else (B)) $,
        lssu = lss $, lequ = leq $,
        eqlu = eql $, nequ = neq $,
        gequ = geq $, gtru = gtr $,

        rebindit(N,V) =
                bind XXXtmp = V;
                undeclare N;
                bind N = XXXtmp;
                undeclare XXXtmp $,

!
! Handy macro in Bliss-10
!
        ID(x) = x$,

!
! Create pointer which consists of the address OR'd with the
!   Qspare bit.  This cannot be done in a straightforward way
!   due to a Bliss-10 bug.
!
        SpcPtr(x) = (x)<32,0> $,

!
! Cheap non-zero value
!
        nonzed = .Sreg $,

!
! Some attempts at solving the problem of declaring data in a
!   machine-independent, language-independent way.
! Major obstacles to be surmounted:
!   GLOBAL BINDs to symbols are illegal in Bliss-10;
!   Bliss-10 does not have initialized OWNs or GLOBALs;
!   Bliss-11 does not have NAMES, INDEXES;
!   Bliss-10 does not have iterated or recursive macros.
!
        globaldata(strc,nm,sz) = bind strc ID(QQQ)nm[sz]=plit(nm globally names $,
        dataarea(n) = bind Vector ID(QQQ)n = plit( n globally names $,
        dataend = ) $,
        gblname(name,d) = name globally names d $,
```

```
        gbindex(name,d) = name globally indexes d $,
        locname(name,d) = name names d $,
        locndex(name,d) = name indexes d $,
        data(d) = d $,

        glbname(x) = x globally names $,
        glbbind(x,v) = switches optimize $,

!
! Support for machine-independent scan and replace operations
!   See B11PRM.REQ for more explanation.
!
        Nptrtochars(x)  = (x)<29,7> $,
        Nptrtobytes(x)  = (x)<28,8> $,
        Iptrtochars(x)  = (x)<36,7> $,
        Iptrtobytes(x)  = (x)<36,8> $,
        NtoIptr(x)      = (x) $,
        ItoNptr(x)      = (x) $,
        Iptrndx(x)      = ((x)-1) $,

!
! Perform zero and move operations in best machine-dependent way.
!
        Zerocor(start,cnt)=
            begin machop BLT=#251;
            register R;
            R<-(start)+1; R<18,18><-(start);
            start<-0;
            BLT(R,(start)+((cnt)-1));
            novalue
            end $,

        Movecor(s,d,cnt)=
            begin machop BLT=#251;
            register R;
            R<-(d); R<18,18><-(s);
            BLT(R,(d)+((cnt)-1));
            novalue
            end $;

!
! Some structures.
!
structure Bytevect[I] = [(I+3)/4] (.Bytevect+.I/4)<28-8*(.I and 3),8>;
structure Charvect[I] = [(I+4)/5] (.Charvect+.I/5)<29-7*(.I mod 5),7>;
structure Hfvect[I] = [(I+1)/2] (if .I then (.Hfvect+.I/2)<18,18>
                                       else (.Hfvect+.I/2)<0,18>);
structure Hftable[I,J,P,S] = [(I+1)/2] (if .I+.J then (.Hftable+(.I+.J)/2)<18+.P,.S>
                                                 else (.Hftable+(.I+.J)/2)<0+.P,.S>);

!
! END OF B10PRM.REQ
```

```
! B11PRM.REQ     (must be copied to SYSPRM.REQ)
!

! Copyright 1977        P. Hibbard and P. Knueven
!                       Pittsburgh, Pennsylvania


! System-dependent declarations for the Bliss 11 version.

bind
        Qadrinc = 2,    ! Increment from word address to next word address
        Qadrsiz = 16,   ! Number of bits in an address value (pointer)
        Qchswrd = 2,    ! Number of characters per word
        Qhfsiz  = 8,    ! Number of bits per half-word
        Qhtab   = 0,    ! True iff horizontal tabs may be sent to output device
        Qspare  = 0,    ! Must be 0 or Qwrdsiz-1
        Qwrdsiz = 16;   ! Number of bits per word

!
! Macros to supply actual names of the Operating System Environment
!   routines and globals.
!
macro
        GBLerrs         = $Errs $,
        GBLwarns        = $Warns $,
        GBLfreelo       = $Freelo $,
        GBLfreehi       = $Freehi $,
        GBLprogstart    = $Progst $,
        GBLpragflags    = $Pragfl $,

        OSEsrcchar      = $SRCchar $,
        OSElstline      = $LSTline $,
        OSEobjword      = $OBJword $,
        OSEerrmsg       = $Errmsg $;

!
! Macros to smooth over the differences in the Bliss-10 syntax and
!   the Bliss-11 syntax.
!
macro   enabend = $,
        enableave = leave $,
        enabreturn = return $,
        quoted = $quote $quote $,
        maxi(A,B) = ((A) max (B)) $,
        mini(A,B) = ((A) min (B)) $,
        semicolon = ; $,
        rebindit(N,V) = rebind N = V $,

!
! Create pointer consisting of address OR'd with the Qspare bit.
!
        SpcPtr(x) = (x) + 1 $,

!
! Cheap non-zero value.
!
        nonzed = .PC $,
```

```
!
! Some attempts at solving the problem of declaring data in a
!   machine-independent, language-independent way.
! Major obstacles to be surmounted:
!   GLOBAL BINDs to symbols are illegal in Bliss-10;
!   Bliss-10 does not have initialized OWNs or GLOBALs;
!   Bliss-11 does not have NAMES, INDEXES;
!   Bliss-10 does not have iterated or recursive macros.
! Macros whose names begin with $ ($Makdata, $Setupdata, $Stripparens,
!   $Setupbinds) are not intended to be used outside B11PRM.REQ .
!
        globaldata(strc,nm,sz) = global bind strc nm[sz] = uplit( $,
        dataarea(name) = $Makdata(name, $,
        dataend = ) $,
        gbiname(name)[] =
                $quote2 $namebind, global, name, $Length-1, ($Remaining) $,
        gbindex(name)[] =
                $quote2 $indxbind, global, name, $Length-1, ($Remaining) $,
        locname(name)[] =
                $quote2 $namebind,       , name, $Length-1, ($Remaining) $,
        locndex(name)[] =
                $quote2 $indxbind,       , name, $Length-1, ($Remaining) $,
        data[] =
                $quote2 $nobind,         ,     , $Length  , ($Remaining) $,

        $Makdata(name)[] =
                external name;
                bind $Name('.NAME.') = name, $Name('.INDEX.') = 0;
                $Setupbinds($Remaining);
                undeclare  name,$Name('.NAME.'), $Name('.INDEX.');
                global bind name = plit $Setupdata($Remaining)  $,

         $quote2 = quoted quoted quoted $,
         $namebind(name) = bind name = $Name('.NAME.') + 2*$Name('.INDEX.') $,
         $indxbind(name) = bind name = $Name('.INDEX.') $,
         $nobind(name) = switches optimize $,
        $Setupdata[bnd,gbl,name,len,list] = $Stripparens list $,
        $Stripparens[] = $Remaining $,
        $Setupbinds[bnd,gbl,name,len,list] =
                gbl bnd(name);
                rebind $Name('.INDEX.') = $Name('.INDEX.')+len  $,

        glbname(x) = $,
        glbbind(x,v) = global bind x = v $,

!
! Support for machine-independent scan and replace operations
!       The goal is to have efficient character/byte scanning operations
!   which are machine independent.  Bliss-10 provides scan and replace
!   operators which allow the programmer to perform PDP-10 byte instructions
!   and thus are relatively efficient.  An efficient method of scanning on
!   the PDP-11 is through the use of auto-increment addressing modes in
!   byte instructions.  Thus, an obvious approach to reaching the goal is
!   to provide Bliss-11 scan and replace macros which cause the generation of
!   byte instructions with auto-increment operands.  The major difficulty
!   in implementing this solution is that the PDP-10 pointer-incrementing
```

```
!   Instructions perform the increment before a fetch or store, while the PDP-11
!   instructions increment afterwards.  This difficulty is overcome by forcing
!   the programmer to pay special attention to the way in which pointers are
!   initialized and used.
!
!       Pointers may be one of two varieties, I-pointer or N-pointer.
!   N-pointers always refer to a particular byte until an incp is performed.
!   I-pointers increment before each access, i.e. after an access an
!   I-pointer refers to the byte accessed.  The rules for using these
!   kinds of pointers may be stated by consistently substituting N or I
!   for X in the following paragraph.
!
!       X-pointers must be initialized by assigning a value yielded by an
!   Xptrto... operator.  All operations on an X-pointer must be an incp,
!   X-operation (i.e. scanX, replaceX or copyXI) or the appropriate conversion
!   operator.
!
!       An I-pointer value may be converted to an N-pointer value by using
!   ItoNptr.  The new N-pointer refers to the byte last accessed by the
!   I-pointer.  Similarly, NtoIptr may be used to create an I-pointer from
!   an N-pointer.  The new I-pointer refers to the same byte as the N-pointer.
!   However, since only I-operations are permissible that byte is not
!   accessible via the I-pointer.
!
!       Iptrndx(n) is used in conjunction with the Bytevect and Charvect
!   structures.  When used as the actual parameter in a Bytevect or Charvect
!   structure access, it produces an I-pointer which will access the n-th
!   byte or character in the first I-operation in which it is used.
!

        incp(ptr)       = ( (ptr)<-.(ptr)+1 ) $,
        scanN(ptr)      = ( .(.(ptr))<0,8> ) $,
        replaceN(ptr,x) = ( (.(ptr))<0,8><-(x) ) $,
        replaceI(ptr,x) = ( (.(ptr))<0,8><-(x); incp((ptr)); ) $,
        copyNI(ipt,opt) = ( replaceI((opt),scanN((ipt))) ) $,
        copyII(ipt,opt) = ( replaceI((opt),scanN((ipt))); incp((ipt)); ) $,
        Nptrtochars(x)  = x $,
        Nptrtobytes(x)  = x $,
        Iptrtochars(x)  = x $,
        Iptrtobytes(x)  = x $,
        ItoNptr(x)      = ((x)-1) $,
        NtoIptr(x)      = ((x)+1) $,
        Iptrndx(x)      = x $,

!
! Perform zero and move operations in best machine-dependent way.
!
        Zerocor(adr,cnt)=
            begin
            register Ptr;
            opcode CLR;
            Ptr<-(adr)+(cnt)^2;
            while .Ptr gtru (adr) do
                CLR(&-Ptr);
            novalue
            end $,

        Movecor(s,d,cnt)=
```

```
            begin register Src,Dst;
            opcode MOV;
            Src←s; Dst←d;
            decr I from (cnt)-1 to 0 do
                MOV(&←Src,&←Dst);
            novalue
            end %;

!
! Some structures.
!
structure
        Bytevect[I] = [I] (.Bytevect+.I)<0,8>,
        Charvect[I] = [I] (.Charvect+.I)<0,8>,
        Hfvect[I] = [I] (.Hfvect+.I)<0,8>,
        Hftable[I,J,P,S] = [I] (.Hftable+.I+.J)<.P,.S>;

!
! Some linkages.
!
linkage
        Reg0 = Bliss(register=0);

!
! END OF B11PRM.REQ
```

```
! COMMON.REQ
!

! Copyright 1977        P. Hibbard and P. Knueven
!                       Pittsburgh, Pennsylvania


! Common definitions included in every module.


! Target machine parameters

bind
        Xbase   = 0,
        Ybase   = #100000,

        Zadrinc = 2,    ! Increment from word address to next word address
        Zchswrd = 2,    ! Characters per word
        Zhfsiz  = 8,    ! Bits per half-word
        Zspare  = 1,    ! Value to add to make an illegal word address
        Zwrdsiz = 16,   ! Bits per word

        ZRsizNaked      = 2,
        ZRsizLEGbase    = 7,
        ZRsizIGbase     = 16,
        ZRsizIGtop      = 4+(if ZRsizNaked eql 1 then 2),
        ZRsizDressed    = 2,
        ZRtypGenblock   = 74,
        ZRsizDesctrip   = 4;

! Useful keywords

macro
        thif    = then if $,
        elif    = else if $,
        allelse = always $,             ! To be used like "T" in a Lisp COND.
        repeat  = while 1 do $,
        exitL   = leave L with $,
        novalue = .Vreg $,
        bool(e) = if (e) then true $,
        times3(x) = (2*(x)+(x)) $,
        modulo(L,R) = (if ((R) and -(R)) eql (R)
                        then (L) and (R)-1
                        else (L) mod (R)) $,
        divide(L,R) = ( if   (R) eql 1 then (L)
                        elif (R) eql 2 then (L)↑(-1)
                        elif (R) eql 4 then (L)↑(-2)
                        else (L)/(R)) $,
        Msg(str) = Nptrtochars(uplit asciz str) $,
        comment = switches optimize; $;

! Some structures

structure Bitvect[P,S]    = (.Bitvect) <.P,.S>;
structure Pblock[I,P,S]   = (e.Pblock+Qadrinc*.I)<.P,.S>;
structure Oblock[I,P,S]   = (e.Oblock+Qadrinc*(.I-1))<.P,.S>;
structure Constbv[I,J,K]  = (if .K eql 0
```

```
                                then .Constbvf(-.I) and 1f.J-1
                                else..Constbvf(-.J) and 1f.K-1) <8,8>;

! More useful definitions

bind
        true    = 1,
        false   = 0,
        zed     = 0,
Constbv Minus1  = -1,
        HTab    = #11,
        LF      = #12,
        FF      = #14,
        CR      = #15,
        EOFchar = #200,
        SRPLstksize = 128,      ! Semantic/Syntax stack size
        Cbufsize= 132;          ! Standard character buffer size

! Macros useful in field definitions

macro
        ZRH     = 0,Zhfsiz $,
        ZLH     = Zhfsiz,Zhfsiz $,
        RH      = 0,Qhfsiz$,
        LH      = Qhfsiz,Qhfsiz$,
        Wordf   = 0,Qwrdsiz $,
        Addrf   = 0,Qadrsiz $,
        Spcf    = Qspare,1 $,
        Clrspare(x) = (x) and not 1fQspare $,
        Zhfwords(Lo,Hi) = ((Hi)fZhfsiz or (Lo)) $,
        Hfwords(Lo,Hi) = ((Hi)fQhfsiz or (Lo)) $;

! Fatal error codes

bind
        FatNospace      = -1,
        FatBufovfl      = -2,
        FatUser         = -3;

! Interface to Operating System Environment (OSE)

external
        GBLerrs, GBLwarns,
        GBLfreelo, GBLfreehi,
        GBLprogstart,
Bitvect GBLpragflags,

        OSEsrcchar,
 Reg0   OSElstline,
 Reg0   OSEobjword,
 Reg0   OSEerrmsg;

! Field definitions for GBLpragflags

macro
        Prgstrop    = RH $,
        Prglist     = Qhfsiz+8,1 $,
```

```
        Prgobj      = Qhfsiz+1,1 $,
        Prgwarn     = Qhfsiz+2,1 $,
        Prgwach     = Qhfsiz+3,1 $,
        Prgnonnked  = Qhfsiz+4,1 $,
        Prgnoghost  = Qhfsiz+5,1 $;

!
! END OF COMMON.REQ
```

```
; SYSDEP.MAC
;

; Copyright 1977        P. Hibbard and P. Knueven
;                       Pittsburgh, Pennsylvania

        TITLE   SYSDEP  ALGOL68 SYSTEM DEPENDENT MACHINE LANGUAGE ROUTINES
        TWOSEG

        JOBREN==124
        LOC JOBREN
        XWD 0,RENADR

        JOB41==41
        LOC JOB41
        JFCL    ; NOP FOR SIX12 UUO'S


        RELOC   400000

; COMPILER CONSTANTS
        LSTCHN==1               ; LISTING CHANNEL

; CALLI DEFINITIONS
        RESET==0
        EXIT==12


; CODE EXECUTED WHEN MONITOR "REENTER" COMMAND IS GIVEN
;
RENADR: CLOSE   LSTCHN,0        ; CLOSE LISTING FILE ON REENTRY
        CALLI   0,RESET         ; DO NOT WRITE OTHER FILES
        CALLI   0,EXIT          ; GO AWAY


; BLISS 10 SPECIAL REGISTERS
        $S==0           ; SREG: STACK POINTER REGISTER
        $F==2           ; FREG: POINTS TO CURRENT INVOCATION BLOCK
        $V==3           ; VREG: VALUE REGISTER


; DETECT FLOATING-POINT OVERFLOW
;
FLTOVF::
        SETO    $V,
        JFOV    .+2
        SETZ    $V,
        POPJ    $S,


; ENABLE FRAME FORMAT
;   FRAME+0:  POINTER TO ENABLE CODE
;   FRAME+1:  POINTER TO PREVIOUS FRAME+3
;   FRAME+2:  SAVED FREG
;   FRAME+3:  SAVED SREG
```

```
; ROUTINE TO SET UP ENABLE FRAME
;
; INPUTS
;   VREG - POINTER TO NEW 4-WORD FRAME-1
;
; OUTPUTS
;   SIGREG - POINTER TO FRAME+3
;   VREG - ZERO
;
.ENABL::
        EXCH    $S,1
        PUSH    $V,0(1)
        PUSH    $V,SIGREG
        PUSH    $V,$F
        PUSH    $V,1
        SOS     0($V)
        MOVEM   $V,SIGREG
        SETZ    $V,
        EXCH    $S,1
        POPJ    $S,


; ROUTINE TO PERFORM A SIGNAL
;
; INPUTS
;   VREG - SIGNALLED VALUE
;
; OUTPUTS
;   SIGVAL - SIGNALLED VALUE
;   SIGREG - RESTORED TO POINT TO PREVIOUS FRAME
;   VREG - NON-ZERO
;   RETURNS TO MOST RECENT (DYNAMICALLY) ENABLE DECLARATION
;
.SIGNL::
        MOVEM   $V,SIGVAL
        MOVE    $V,SIGREG
        POP     $V,$S
        POP     $V,$F
        POP     $V,SIGREG
        POPJ    $V,


; INITIAL ENABLE CODE
ENABO:  CALLI   1,EXIT

; INITIAL ENABLE FRAME
FRAMEO: EXP     ENABO, FRAMEO+3, 0, 0


        RELOC   0

; VARIABLES USED BY SIGNAL/ENABLE ROUTINES
SIGVAL::        0               ; SIGNALLED VALUE
SIGREG::        FRAMEO+3        ; POINTS TO TOP OF STACK OF ENABLE FRAMES

        END
```

```
; File:   SIGENB.M11[N818HY97]
;
;    This work was supported by the Advanced Research
;    Projects Agency of the Office of the Secretary of
;    Defense (F44620-73-C-0074) and is monitored by the
;    Air Force Office of Scientific Research.

        .TITLE  SIGENB

        .CSECT SIGE.C

; The BLISS-11 out of line routines to handle SIGNAL's and
; ENABLE frame creation.
;
; 11-Nov-73                     P. Knueven
; 26-Jul-76                     P. Karlton - change to location of
;                               SIGREG and SIGVAL to be SAVREG and
;                               SAVVAL as in STKPAG.REQ[N811HY97]
;                               Commented out LEV0 and EXIT also.
;
; The dynamically nested occurrences of ENABLE declarations
; are recorded in the LIFO ENABLE stack.  This is implemented
; as a linked list of 3-word stack frames.  The current top of
; the stack is pointed to by .SIGREG.  An ENABLE frame is
; created each time an ENABLE declaration is "executed".
; It looks like the following:
;
;     Offset    Contents
;       0       Pointer to most recent previous frame
;       2       Value of SP for the ENABLE body
;       4       Pointer to the ENABLE body
;

        .GLOBL  $SIGNL,$SIGN1
        .GLOBL  $ENABL
        .GLOBL  SIGVAL,SIGREG

R0=%0
SP=%6
PC=%7

; Calling Sequence
;   MOV E,R0
;   JMP $SIGNL
;
$SIGNL: MOV     R0,SIGVAL
$SIGN1: MOV     SIGREG,R0
        MOV     (R0)+,SIGREG
        MOV     (R0)+,SP
        MOV     (R0)+,PC

; Calling Sequence
;   MOV #n+6,R0
;   JSR PC,$ENABL
;   .WORD L2-L1
; L1:
;   ... ENABLE body ...
```

```
; L2:
;
;   where n is 2 plus the offset into the stack of the
;     space reserved for the frame
;
$ENABL: ADD     SP,R0
        MOV     (SP),(R0)
        ADD     #2,(R0)
        MOV     @(SP),(SP)
        ADD     (R0),(SP)
        MOV     SP,-(R0)
        ADD     #2,(R0)
        MOV     SIGREG,-(R0)
        MOV     R0,SIGREG
        RTS     PC

;$EXIT: HALT
;
;$LEV0: .WORD   0,0,$EXIT

SIGREG  =       270             ; SAVREG address
SIGVAL  =       272             ; SAVVAL address

        .END
```

```
! A68S10.B10
!

! Copyright 1977        P. Hibbard and P. Knueven
!                       Pittsburgh, Pennsylvania

module A68S10(stack(1000)) =
begin
!
!               PDP-11 ALGOL 68S Cross-compiler for PDP-10
!               ------------------------------------------
!
switches nolist;
  require B10PRM.REQ;
  require COMMON.REQ;
  require IOMACS.REQ;
switches list;

undeclare
        OSEistline,
        OSEerrmsg,
        OSEobjword,
        OSEsrcchar;

forward
    !   Initchnl,
    !   InitFDB,
        Cmdmsg,
        Cvtfile,
        Reset,
        Getbuf,
        Lkupntr,
        Getcmd,
        Permsg,
        Initchns,
        Punt,
        Page,
        Lread;

bind    FatCmdlong = FatUser-8,
        LSTlinesperpage = 58;

own
        LSTheadpg,
        LSTpage,
        LSTcount;

Machop Call1 = #47;


bind
        Crlfm   = Msg('?M?J'),
        Spcsm   = Msg('   '),
        Errm    = Msg('Command error'),
        PPNm    = Msg('Invalid PPN');

bind
```

```
        Maxbuf  = #203,
        IObufl  = 6*Maxbuf;

external
        Buffers,
        Close,
        Date,
        Dayofweek,
        Enter,
        Filescan,
        Lookup,
        Open,
        Pdate,
        Ptime,
        Purgeout,
        Read,
        Writemsg;

own
        IObuff[IObufl],
        IObufp,
        Srcbufp,
Bitvect Greatem,
        FDB[5],
        Extseen,
        Chnl;

bind
        Device    = FDB[0],
        Filename  = FDB[1],
        Extension = FDB[2],
        PrTiDate  = FDB[3],     ! Protection, Time, Date
        PPN       = FDB[4],

        LkupBlk = FDB[1],
        NtrBlk  = FDB[1];

!
! Names of fields in words in the Chnlstat block
!

macro
        Speed   = 0,1 $,        ! True if channel has been correctly initialized
        TTYd    = 1,1 $,        ! True if device on channel is a TTY
        Binio   = 18,1 $,       ! True if channel is for binary I/O,
                                ! False if for ASCII
        Inpio   = 19,1 $;       ! True if channel is for input,
                                ! False if for output

bind
        Cmichn  = -2,
        Cmochn  = -1,
        Binchn  = 0,
        Lstchn  = 1,
        Srcchn  = 2,
        Hichn   = Srcchn;
```

```
structure Bufhdvec[I] = [3*I] (.Bufhdvec+3*.I);
structure Block[I,P,S] = [I*Qadrinc] (.Block+Qadrinc*.I)<.P,.S>;

own Bufhdvec Bufhd[Hichn+1];
own Block Chnlstat[Hichn+1];

structure Chnstatwd[P,S] = (Chnlstat+..Chnstatwd)<.P,.S>;

map Chnstatwd Chnl;     ! so that .Chnl[Specd] means .Chnlstat[.Chnl,Specd]

bind Vector Initstat[Hichn+1] = plit(
        Hfwords(0,1),
        Hfwords(0,0),
        Hfwords(0,2));

macro Initchnl(chan,blk) =
    begin
!   blk is always FDB
    Chnl←(chan);
    Lkupntr()
    end%;

macro InitFDB(dev,file,ext) =
    begin
    Device←dev;
    Filename←file;
    Extension←ext;
    PrTiDate←PPN←0;
    end%;


comment ! Cvtfile
!
! Function: set up a character string, to be output by the caller,
!   that describes a file (device, name, extension).
!
! Locals
!   Msgbuf - space to hold the string
!   Dstp - a byte pointer to the string
!   FDB - the block holding the information about the file
!
! Output
!   VALUE - byte pointer to the 1st character of the string
!
routine Cvtfile =
    begin
    own Msgbuf[4],Dstp;


    comment ! Cvtsix(N)
    !
    ! Function: convert a SIXBIT string to ASCII and append it to
    !   the current string (DSTP).
    !
    ! Inputs
    !   N - word containing the SIXBIT string
    !
```

```
    routine Cvtsix(N) =
        begin
        local Srcp;
        Srcp←W<36,6>;
        decr I from 5 to 0 do
            if scani(Srcp) eql 0
                then exitloop
                else replacei(Dstp,scann(Srcp)+#40);
        novalue
        end;

    Dstp←Iptrtochars(Msgbuf);
    Cvtsix(.Device);
    Replacei(Dstp,":");
    Cvtsix(.Filename);
    Replacei(Dstp,".");
    Cvtsix(.Extension and #777777700hfsiz);
    Replacei(Dstp,0);
    Nptrtochars(Msgbuf)
    end;


comment ! Cmdmsg(Str)
!
! Function: output a message to the user's terminal, ending
!   with CRLF, using TTCALL's.
!
! Inputs
!   Str - pointer to a word whose high byte begins the
!         string to be output.
!
routine Cmdmsg(Str) =
    begin
    Writemsg(Cmochn,.Str);
    Writemsg(Cmochn,Crlfm);
    Purgeout(Cmochn);
    novalue
    end;


comment ! Reset
!
! Function: do a RESET UUO, and reinitialize the Chnlstat words
!
routine Reset =
    begin
    map Vector Chnlstat;
    IObufp←IObuff<0,8>;
    Movecor(Initstat,Chnlstat,Hichn+1);
    Calli(0,8);
    novalue
    end;


comment ! Getbuf
!
! Function: allocate space for I/O buffers for a given channel,
```

```
!   calling the BLILIB routine BUFFERS.
!
! Inputs
!   Chnl - a channel number, between Cmichn and Hichn inclusive
!   Iobuff[Iobufl] - a large block of space from which all I/O buffers are taken
!   Iobufp - a pointer to the beginning of unused space in Iobuff
!
! Outputs
!   Iobufp - updated
!   VALUE - True if buffer space was allocated,
!           false if some error was encountered
!
routine Getbuf =
    begin
    local Size;
    if .Chnl eql Srcchn then Srcbufp←.IObufp;
    Size←2*(2+Buffers(.Chnl,0,.Chnl[Inpio],0));
    if .IObufp+.Size leq IObuff[IObufl]<0,0> then
        begin
        Buffers(.Chnl,2,.Chnl[Inpio],,IObufp);
        IObufp←.IObufp+.Size;
        True
        end
    end;


comment ! Lkupntr
!
! Function: initialize a channel for I/O as indicated
!
! Inputs
!   Chnl - the channel number
!   FDB - a block containing all necessary information about the
!         current channel, except that contained in
!   Chnlstat[Hichn] - a vector of words, one for each channel, each
!                     of which contains a few bits of information about
!                     the channel.
! Outputs
!   VALUE - True if initialization succeeds, false otherwise
!
routine Lkupntr =
    begin
    macro Devchr(dvnm) = (register QQ; QQ←dvnm; calli(QQ,#4))$;
    macro Binarymf = 12,1 $,    ! Names of fields in the DEVCHR result word
          Asciimf  = 1,1 $,
          TTYdevf  = 21,1 $;
    bind Binarymode = #14, Asciilinemode = 1;
    bind L150 = #25627, L150AL71 = #25627010165;
    register Bitvect R;

    if not .Chnl[Speed]                 ! Is channel already in use?
      thif (R←Devchr(.Devico)) neq 0    ! Does device exist?
      thif (if .Chnl[Binio]             ! Can device do I/O of specified type?
                then .R[Binarymf]
                else .R[Asciimf])
      thif Open(.Chnl,                  ! Does OPEN UUO succeed?
                if .Chnl[Binio]
```

```
                then Binarymode
                else Asciilinemode,
              .Device,
              if .Chnl[Inpio]
                then Bufhd[.Chnl]<0,0>
                else Bufhd[.Chnl]<0,0>+Qhfsiz)
    thif Getbuf() neq 0                 ! Can buffers be gotten?
    thif (if .Chnl[Inpio]               ! Does LOOKUP or ENTER UUO succeed?
              then begin
                  local SavPPN;
                  SavPPN←.PPN;
                  if Lookup(.Chnl,LkupBlk) then exitblock true;
                  if not .Extseen then
                      begin
                      Extension←0;
                      if Lookup(.Chnl,LkupBlk) then exitblock true;
                      Extension←sixbit 'A68';
                      end;
                  if .SavPPN eql 0 and .Greetem[LH] eql L150 then
                      begin
                      PPN←L150AL71;
                      if Lookup(.Chnl,LkupBlk) then exitblock true;
                      end;
                  false
                  end
              else Enter(.Chnl,NtrBlk))
        then begin                      ! Success
            Chnl[TTYd]←.R[TTYdevt];
            Chnl[Speed]←True;
            return True;
            end;
    Cmdmsg(Msg('Initialization failure')); ! Failure
    Cmdmsg(Cvtfile());
    False
    end;


comment ! Getcmd
!
! Function: input a command from the user's terminal,
!   using TTCALL's
!
! Outputs
!   Cmdbuf - a block of bytes in which the command is stored as an ASCII string
!
routine Getcmd =
    begin
    bind Cmdbufl = 20;
    own Cmdbuf[Cmdbufl];
    register Ptr,N;
    Ptr←Iptrtochars(Cmdbuf);
    N←Qchsmrd@Cmdbufl;
    do begin
        if (N←.N-1) lss 0 then
            Punt(FatCmdlong);
        replacei(Ptr,Read(Cmichn));
        end
```

```
      until .Vreg eql Lf;
    Iptrtochars(Cmdbuf)
    end;

routine Permsg =
    begin
    macro GetPPN = (register QQ; QQ←0; Calli(QQ,#24))$;
    macro Greet(Who,What) = Who: exitselect Cmdmsg(Msg(What))$;
    bind
        AH20 = #005040,
        AL68 = #010130,
        AL69 = #010131,
        AL70 = #010164,
        AL74 = #010170,
        BL03 = #032013,
        GA10 = #155604,
        PH02 = #427112,
        PK02 = #431202,
        SA20 = #511150,
        TR30 = #547404,
        WW17 = #641573;
    if .Greetem neq 0 then return novalue;
    Greetem←GetPPN;
    select .Greetem[RH] of
      nset
        Greet(AH20,'Watch me blaze Pascal');
        Greet(AL68,'Let me elaborate on that');
        Greet(AL69,'Let me elaborate on that');
        Greet(AL70,'Let me elaborate on that');
        Greet(AL74,'Where''s the missing link??');
        Greet(BL03,'Hi Bruce');
        Greet(GA10,'Hey Guy');
        Greet(PH02,'Thou art Peter');
        Greet(PK02,'Good luck Paul');
        Greet(SA20,'It''s that crazy Swede again');
        Greet(TR30,'Give my regards to Cm*');
        Greet(WW17,'It''s big daddy Bill!')
      tesn;
    novalue
    end;


comment ! Initchns(Srconly)
!
! Function
!     Initialize I/O channels.
!
!     If Srconly is geq 0 then get command line from
!     user and initialize specified channels.
!
!     If Srconly is lss 0 then initialize source input channel to read from
!     next specified input file.  Return False as value iff end-of-data or
!     error occurred.
!
routine Initchns(Srconly) =
    begin
    macro
```

```
        Cmddone = return True $,
        Cmdfail = return False $,
        Cmderr(x) = (Cmdmsg(x); Cmdfail)$;
    own Cmdeof,Ptr;
    local Bitvect Scanv;
    macro Rotcode = RH $,       ! Names of fields in the FILESCAN result word
          Breakch = LH $;

    if .Srconly geq 0
        then begin
            Reset();
            Permsg();
            Writemsg(Cmochn,Msg('*'));
            Purgeout(Cmochn);
            Cmdeof←False;
            LSTpage←0;
            LSTcount←LSTlinesperpage;
            Ptr←Getcmd();
            Chnl←Lstchn;
            end
        else begin
            Chnl←Srcchn;
            end;
    until .Cmdeof do
        begin
        InitFDB(sixbit 'DSK', 0,
            case .Chnl-Binchn of set sixbit 'OBJ'; sixbit 'P11'; %sixbit 'A68'% 0 tes);
        Scanv←Filescan(Ptr,FDB);
        if not (Extseen←(.Extension neq 0))
          then Extension←sixbit 'A68';
        case .Scanv[Rotcode]↑(-1) of
          set
            0;
            Cmderr(Errm);
            Cmderr(PPNm);
            Cmderr(PPNm);
            Cmderr(PPNm);
            Cmderr(Msg('Invalid switch specification'))
          tes;
        if .Scanv[Breakch] eql "←"
            then Scanv[Breakch]←"="
          elif .Scanv[Breakch] eql ";"
            then Scanv[Breakch]←Cr;
        select .Scanv[Breakch] of
          nset
            ",":exitselect begin
                if .Chnl eql Lstchn then Cmderr(Errm);
                if .Scanv then
                    if not Initchnl(.Chnl,FDB) then Cmdfail;
                if .Chnl eql Binchn
                    then Chnl←Lstchn
                    elif .Scanv then Cmddone;
                end;

            "=":exitselect begin
                if .Chnl eql Srcchn then Cmderr(Errm);
                if .Scanv then
```

```
                    if not Initchnl(.Chnl,FDB) then Cmdfail;
                Chnl←Srcchn;
                end;

            Cr: exitselect begin
                Cmdeof←True;
                if .Scanv
                    then (if .Chnl eql Srcchn then
                            if not Initchnl(.Chnl,FDB)
                                then Cmdfail
                                else Cmddone)
                    elif .Chnl eql Binchn then
                        Cmdfail;
                Cmderr(Errm);
                end;

            Allelse: Cmderr(Msg('Illegal delimiter'))
          tesn;
        end;
    false
    end;

routine Punt(N) =
    begin
    external JobDDT,Jobsa,Sixl2;
    decr I from Hichn to 0 do
        if .Chnlstat[.I,Specd] and not .Chnlstat[.I,Inpiol then
            Purgeout(.I);
    Cmdmsg(Msg('?M?J??PUNT!'));
    if .JobDDT neq 0
        then Sixl2(-1)
        else (.Jobsa)();
    novalue
    end;


!
! Source Input
!

comment ! Lread
!
! Function: call the BLILIB routine READ, passing Srcchn as
!   argument, and return the result, except that if the character read
!   is a character of an SOS line number, return an error code of -3.
!
routine Lread =
    begin
    bind Vector Bufhed = Bufhd[Srcchn];
    bind Bytptr = 1;
    if Read(Srcchn) gtr 0 then
        if @.Bufhed[Bytptr] then
            Vreg←-3;
    .Vreg
    end;
```

```
comment ! OSEsrcchar
!
! Function: get a character from the input buffer.  Process
!   line numbers, I/O errors, and end-of-file conditions correctly.
!
global routine OSEsrcchar =
    begin
    local Bitvect Char, InSOSno;
    InSOSno←zed;
    while (Char←Lread()) lss 0 do
        case .Char+3 of
          set
            ! 0 - Line number character
            InSOSno←nonzed;

            ! 1 - I/O error
            begin
            Cmdmsg(Msg('Input transmission error'));
            return EOFchar;
            end;

            ! 2 - End-of-file
            begin
            Close(Srcchn);
            Chnlstat[Srcchn,Wordf]←.Initstat[Srcchn];
            IObufp←.Srcbufp;
            if not Initchns(-1) then
                return EOFchar;
            end
          tes;
    if .InSOSno neq 0 and .Char eql HTab
        then 0
        else .Char
    end;


!
! Listing Output
!

routine Lstmsg(Str) =
    begin
    Writemsg(Lstchn,.Str);
    novalue
    end;


routine Page =
    begin
    external Versid;
    bind Days[7+2] = plit(
        'Sunday',
        'Monday',
        'Tuesday',
        'Wednesday',
        'Thursday',
        'Friday',
```

```
        'Saturday');
    local Today;
    LSTcount←0;
    if not .Chnistat[Lstchn,TTYd] then
        begin
        Today←Date();
        Lstmsg(Msg('?L;Algol 68 '));
        Lstmsg(Nptrtochars(Versid));
        Lstmsg(Spcsm);
        Lstmsg(Nptrtochars(Days[2*Dayofweek(.Today)]));
        Lstmsg(Spcsm);
        Pdate(Lstchn,.Today);
        Lstmsg(Spcsm);
        Ptime(Lstchn,-1);
        Lstmsg(Spcsm);
        Lstmsg(Cvtfile());
        Lstmsg(Msg('   Page '));
        LSTheadpg←0;
        Cvtdecz(Iptrtochars(LSTheadpg),LSTpage←.LSTpage+1,3);
        Lstmsg(Nptrtochars(LSTheadpg));
        Lstmsg(Crlfm);
        Lstmsg(Crlfm);
        Purgeout(Lstchn);
        end;
    novalue
    end;


comment ! OSElstline(First,Foll,Errdev)
!
! Function: output a string to the listing device, and possibly to
!   the user's terminal as well, appending a CRLF.
!
! Inputs
!   First- byte pointer to the first character of the (ASCII) string
!   Foll - byte pointer to the character just after the last character
!          of the string. This is temporarily set to zero, to make
!          the string ASCIZ, before calling Writemsg.
!   Errdev - nonzero if the string is also to be output to the terminal
!
global routine OSElstline(First,Foll,Errdev) =
    begin
    if .GBLpragflags[Prglist] or .Errdev neq 0 then
        begin
        local Sav;
        if (LSTcount←.LSTcount+1) gtr LSTlinesperpage then
            Page();
        Sav←scann(Foll);
        replacen(Foll,0);
        if .Chnistat[Lstchn,Spocd] then
            begin
            Lstmsg(.First);
            Lstmsg(Crlfm);
            if .Chnistat[Lstchn,TTYd] then Purgeout(Lstchn);
            end;
        if .Errdev neq 0 and not .Chnistat[Lstchn,TTYd] then
            Cmdmsg(.First);
```

```
        replacen(Foll,.Sav);
        end;
    novalue
    end;


!
! Object Output
!

global routine OSEobjword = novalue;


!
! Error Message Output
!

global routine OSEerrmsg(N) =
    begin
    external Errtxt;
    if .Errtxt[.N] neq 0 then
        begin
        Outs(' - ');
        Outsaz(.Errtxt[.N]);
        end;
    novalue
    end;


!
! Initialization and Finalization
!

macro Initmem (dum) =
    begin
    global GBLfreelo,GBLfreehi,Freearea[3000];
    GBLfreelo ← Freearea<0,0>;
    GBLfreehi ← (Freearea+3000)<0,0>;
    novalue
    end %;

routine Initprags =
    begin
    GBLpragflags←0;
    GBLpragflags[Prgstrap]←1;
    GBLpragflags[Prgwarn]←true;
    GBLpragflags[Prglist]←true;
    GBLpragflags[Prgobj]←true;
    novalue
    end;

macro Inittransput(dum) =
    begin
    until Initchns(0) do Initprags();
    if .Chnlstat[Lstchn,TTYd] then
        begin
        external Versid;
```

```
        Writemsg(Cmochn,Msg('?LAlgol 68 '));
        Cmdmsg(Nptrtochars(Versid));
        end;
    novalue
    end$;

macro Fintransput(dum) =
    begin
    decr I from Nichn to 0 do
        if .Chnstat[.I,Speed] then
            Close(.I);
    novalue
    end$;

external Algol68;

Greatom←0;
repeat
    begin
    local Val;
    global GBLerrs,GBLwarns,GBLprogstart,GBLpragflags;
    Initmem();
    Initprags();
    Inittransput();
    if (Val←Algol68()) lss 0 then Punt(-.Val);
    Fintransput();
    end;

novalue

end
eludom
```

```
! A68S11.B11
!

! Copyright 1977        P. Hibbard and P. Knuoven
!                       Pittsburgh, Pennsylvania

module A68S11(START = Start, NODEBUG) =
begin
!
!               Algol 68 Compiler Hydra Operating System Environment
!               ----------------------------------------------------
!

!       This module contains most of the routines which comprise the
!   Hydra OSE for the Algol 68 Compiler.  The routine Start is the
!   entry point for the Hydra load-and-go compiler system.
!
! Compilation Initialization
!       The following functions must be performed:
!       - Ascertain which output files are to be created.  Perform any
!           setup required to make subsequent output actions work correctly.
!           Set $Pragfl[Prglist] to true iff the listing file is to be
!           written and $Pragfl[Prgobj] to true iff the object file is to be
!           written.
!       - Determine what text is to be used as the source input.  Make any
!           necessary connections to allow subsequent input actions to work
!           correctly.
!       - Alter $Pragfl to reflect any requests for pragmat-controlled
!           actions.
!       - Do anything else necessary for particular system.
!       - Set Freelo and Freehi.
!
! Source Input  ($SRCchar)
!       The routine $SRCchar returns the next character from the source input.
!   If no more input exists it returns EOFchar.  The characters must be
!   encoded as 7-bit ASCII.
!
! Listing Output  ($LSTline)
!       $LSTline outputs a single line of text to the listing device and possibly
!   to the command output device as well.  It takes three parameters, an
!   N-pointer to the first character of the line, an N-pointer to the character
!   immediately following the last character of the line, and an indication of
!   whether the line should be sent to the command output device (0 implies do
!   not send it).
!
! Error Reporting  ($Errmsg)
!       $Errmsg(N) is called to output any text message associated with error
!   number N.
!
! Object Output  ($OBJword)
!       $OBJword is called to output a word of object code.
!
! Compilation Finalization
!

switches nolist;
  require KERNAL.REQ [N811HY97];
```

```
  require CDCODE.REQ [N811HY97];
  require STKPAG.REQ [N811HY97];
  require RTSSTR.REQ [N811HY97];
  require RTS   .REQ [N811HY97];
  require PMCALL.REQ [N810PM99];
  require TTCALL.REQ [N810PM99];

  require B11PRM.REQ;
  require COMMON.REQ;
  require IOMACS.REQ;
switches list;

undeclare
        $Errmsg,
        $LSTline,
        $OBJword,
        $SRCchar;

external
        Closedev,
        IOinit,
        Opendev,
        Outasciz,
        Outm,
        Outcrlf,
        Versid,

        Globlog,
        Sixcmd;

bind
        FatHarderr      = FatUser-0,
        FatSignal       = FatUser-1,
        FatObjovfl      = FatUser-2,

        CMIchan         = 0,
        CMOchan         = 1,
        LPTchan         = 2,

        RPSdata1        = 4,
        RPSdata2        = 5,
        RPSdata3        = 6,
        RPSio           = 7,

        LNSsixl2        = 1,
        LNSsys          = 2,
        LNSuser         = 3,
        LNSport         = 4,    % Parameter %
        LNSterm         = 5,    % Parameter %
        LNSsource       = 6,    % Parameter %
        LNSoptions      = 7,    % Parameter %
        LNSobject       = 8,    % Parameter %
        LNSpages        = 13,
        LNSprodt1       = 16,
        LNSprodt2       = 17,
        LNSprodt3       = 18,
        LNSerrmsg       = 19,
```

```
        LNSdata1        = 31,
        LNSdata2        = 32,
        LNSdata3        = 33,
        LNStemp         = 34,

        CPSdata1        = 4,
        CPSdata2        = 5,
        CPSdata3        = 6,
        CPSerrmsg       = 7,
        CPSxseg         = 21,
        CPSyseg         = 22,
        CPSsource       = 23,

        IOpage          = #200000^RPSio;

own
Charvect CMObuf[80],
Charvect LPTbuf[136],
        Debcom          = 0,
        IOcurpg         = 0,
        IOinptr         = IOpage,
        LSTchan         = CMOchan,
        LSTcount        = 54,
        LSTdev,
Vector  LSThead[17]     = asciz '   Algol 68 on C.mmp     Page XXX',
        LSTpgno         = 0,
        SRCpage         = CPSsource;

bind
        LSTheadpg       = LSThead+30;


routine interrupt Herr =
    begin
    Outs('?G?G?G?GHard Error at Compiler PC=#'); Outoct(.OldPC); Eolerr();
    Outs('   with ERRCODE=#'); Outoct(.$ERRCODE); Eolerr();
    if .Debcom neq 0 then
        Sixcmd(#1000);
    signal FatHarderr
    end;


routine interrupt Hsig =
    begin
    local T;
    T=.R0;
    Outs('?G?GSignal #'); Outoct(.T<0,15>); Outs(' at Compiler PC=#'); Outoct(.OldPC); Eolerr();
    Outs('   with SIGDATA=#'); Outoct(.$SIGDATA); Eolerr();
    if .Debcom neq 0 then
        Sixcmd(#1001);
    signal FatSignal
    end;


routine IOload(CPSslot) =
    begin
    if .CPSslot neq .IOcurpg then
```

```
        begin
        IOcurpg←.CPSslot;
        $RRLOAD(RPSio,.CPSslot);
        end;
    novalue
    end;


routine Punt(N) =
    begin
    if .Debcom neq 0 then
        Sixcmd(#777);
    Outs('PUNT! '); Outdec(.N);
    Eolerr();
    novalue
    end;


!
! Source Input
!

global routine $SRCchar =
    begin
    local Val;
    if .IOinptr eql IOpage+#20000 then          ! ???? This may lose if EOF occurs at boundary
        (SRCpage←.SRCpage+1; IOinptr←IOpage);
    IOload(.SRCpage);
    Val←scanN(IOinptr); incp(IOinptr);
    if .Val eql 0 then
        if (Val←scanN(IOinptr)) neq 0
            then incp(IOinptr)
            else Val←EOFchar;
    .Val
    end;


!
! Listing Output
!

global routine Reg0 $LSTline(First,Foll,Errdev) =
    begin
    if .$Pragfl[Prglist] or .Errdev neq 0 then
        begin
        if (LSTcount←.LSTcount+1) gtr 54 then
            begin
            LSTcount←0;
            if .LSTchan neq CMOchan then
                begin
                Outasciz(.LSTchan,Msg('?L'));
                Outasciz(.LSTchan,Versid);
                Cvtdecz(LSTheadpg,LSTpgno←.LSTpgno+1,3);
                Outasciz(.LSTchan,LSThead);
                Outcrlf(.LSTchan);
                Outcrlf(.LSTchan);
```

```
            end;
        end;
    Outm(.LSTchan,.First,.Foll);
    Outcrlf(.LSTchan);
    if .LSTchan neq CMOchan and .Errdev neq 0 then
        begin
        Outm(CMOchan,.First,.Foll);
        Outcrlf(CMOchan);
        end;
    end;
novalue
end;


!
! Object Output
!

bind
        Display     = #2000,
        Tranvect    = #100000,
        Xpage       = #120000,
        Ypage       = #140000;

macro SelectX(dum) = IOload(CPSxseg) $;

macro SelectY(dum) = IOload(CPSyseg) $;

global routine RegO $OBJword(Loc,Dtype,Dvalue) =
    begin
    local Doffset;
    bind Vector Basetable = uplit( 0, 0, 0, 0, Xpage, Ypage, Display, Tranvect );
    if .Loc gequ Ybase
        then begin
            Loc←.Loc-Ybase;
            SelectY();
            end
        else begin
            Loc←.Loc-Xbase;
            SelectX();
            end;
    if .Loc gequ #10000 then signal FatObjovfl;
    Doffset←(case .Dtype of
                set
                  .Dvalue;
                  .Dvalue;
                  .Dvalue;
                  .Dvalue;
                  .Dvalue@Zadrinc - Xbase@Zadrinc;
                  .Dvalue@Zadrinc - Ybase@Zadrinc;
                  .Dvalue@Zadrinc;
                  .Dvalue@2@Zadrinc
                tes);
    IOpage[.Loc]←.Basetable[.Dtype] + .Doffset;
    novalue
    end;
```

```
!
! Error Message Output
!

global routine Reg0 $Errmsg(N) =
    begin
    external Errtxt;
    IOload(CPSerrmsg);
    if .Errtxt[.N] neq 0 then
        begin
        Outs(' - ');
        Outsaz(.Errtxt[.N]);
        end;
    novalue
    end;


!
! Initialization and Finalization
!

macro Initcusp(dum) =
    begin
    local $StrCBERRORTRAPS Traphandlers;
    local $Rights Restrict;
    Traphandlers[$ERRPC]←Herr;
    Traphandlers[$SIGPC]←Hsig;
    $SETLCB(0,Traphandlers,$CBERRORTRAPS);
    $COPY(LNSdata1,LNSprocdt1,CPSdata1);
    $COPY(LNSdata2,LNSprocdt2,CPSdata2);
    $COPY(LNSdata3,LNSprocdt3,CPSdata3);
    $RRLOAD(RPSdata1,CPSdata1);
    $RRLOAD(RPSdata2,CPSdata2);
    $RRLOAD(RPSdata3,CPSdata3);
    $CPSLOAD(0,
        <CPSxseg,$PATH(LNSobject,1)>,
        <CPSyseg,$PATH(LNSobject,2)>);
    Restrict[$AuxRTS]←not $WritePageRTS;
    Restrict[$GenericRTS]←-1;
    incr I from 1 to $CLENGTH(LNSsource) do
        begin
        $GETCAPA(LNStemp,$Path(LNSsource,.I));
        $PUTCAPA(LNStemp,LNStemp,Restrict);
        $CPSLOAD(0, <CPSsource-1+.I,LNStemp>);
        $DELETE(LNStemp);
        end;
    novalue
    end$;

macro Initmem(dum) =
    begin
    csect global='FREE.C';
    global Freearea;
    bind Freeend = IOpage;
    $Freelo←Freearea;
    $Freehi←Freeend;
```

```
    novalue
    end$;

routine Doswitches =
    begin
    stacklocal Leng;
    Leng←$DLENGTH(LNSoptions);
    incr I from 2 to .Leng do
        begin
        stacklocal Sw;
        $GETDATA(Sw,LNSoptions,.I,1);
        if 0 lss .Sw and .Sw lss 17 then
            case .Sw of
              set
                ;
                % DEBUG %        Debcom←nonzed;
                % GHOST %        $Pragfl[Prgnoghost]←false;
                % LISTING %      $Pragfl[Prglist]←true;
                % LOWER %        $Pragfl[Prgstrop]←3;
                % MACH %         $Pragfl[Prgmach]←true;
                % NAKED %        $Pragfl[Prgnonaked]←false;
                % NODEBUG %      Debcom←zed;
                % NOGHOST %      $Pragfl[Prgnoghost]←true;
                % NOLISTING %    $Pragfl[Prglist]←false;
                % NOMACH %       $Pragfl[Prgmach]←false;
                % NONAKED %      $Pragfl[Prgnonaked]←true;
                % NOWARNINGS %   $Pragfl[Prgwarn]←false;
                % POINT %        $Pragfl[Prgstrop]←1;
                % RES %          $Pragfl[Prgstrop]←0;
                % UPPER %        $Pragfl[Prgstrop]←2;
                % WARNINGS %     $Pragfl[Prgwarn]←true
               tes;
        end;
    novalue
    end;

macro Initprags(dum) =
    begin
    $Pragfl←0;
    $Pragfl[Prgstrop]←1;
    $Pragfl[PrgWarn]←true;
    $Pragfl[Prgobj]←true;
    $Pragfl[Prglist]←true;
    Doswitches();
    novalue
    end$;

routine Inittransput =
    begin
    IOinit(LNSport);
    Opendev(CMOchan,CMObuf[0],80,3);
    $GETDATA(LSTdev,LNSoptions,1,1);
    if .LSTdev eql 0
        then $Pragfl[Prglist]←false
      elif .LSTdev eql 2
        then begin
            Opendev(LPTchan,LPTbuf[0],136,6);
```

```
            LSTchan←LPTchan;
            end;
    Outasciz(CMDchan,Versld);
    Outasciz(CMDchan,Msg(' Compiler Starting'));
    Outcrlf(CMDchan);
    novalue
    end;

routine Fintransput =
    begin
    if .LSTchan eql LPTchan then
        Closedev(LPTchan);
    Closedev(CMDchan);
    novalue
    end;


routine A68 =
    begin
    local Val;
    external Algol68;
    Initcusp();
    Initmem();
    Initprags();
    Inittransput();
    if .Debcon neq 0
        then Sixcmd(0)
        else Glotog←-1;
    if (Val←Algol68()) eql 0
        then Val←Zadrinco.$Progst+Ypage
        else begin
            if .Val lss 0 then Punt(-.Val);
            Val←0;
            end;
    $PUTDATA(LNSobject,Val,1,1);
    Fintransput();
    $BREAK(LNSterm);
    .Val
    end;

routine Start =
    begin
    external Init612,Rat612;
    csect global='GLOB.C';
    global $Errs,$Freehi,$Freelo,$Prayfl,$Progst,$Warns;
    Init612(LNSport,0,0,1,1);
    Rat612(A68(),0,0)
    end;

end
eludom
```

```
! MDSTRC.REQ
!
```



```
! required by:  CODE, CODESU, DBDEB1, LEXAN, SEMR0, SEMR2, SEMR4, SEMRSU, SRTABS

! Definitions associated with modes.

macro
        Mode = Oblock $,
        XMode = 1 $,

        Mdv         = 1,Wordf $,
          MdID      = 1,RH $,
          Mdseen    = 1,Qwrdsiz-8,1 $,
          Mddone    = 1,Qwrdsiz-7,1 $,
          Mddeproc  = 1,Qwrdsiz-6,1 $,
          Mdspecial = 1,Qwrdsiz-5,1 $,
          Mdrecur   = 1,Qwrdsiz-4,1 $,
          Mddressed = 1,Qwrdsiz-3,1 $,
          Mdio      = 1,Qwrdsiz-2,1 $,
          Mdpile    = 1,Qwrdsiz-1,1 $,

        Mdprofmd    = 2,Wordf $,

        Mdunddap    = 2,Wordf $,

        Mdunocnt    = 2,Wordf $,
        Mdunomd(n)  = 3+(n),Wordf $,

        Mdprores    = Mdprofmd $,
        Mdprocnt    = 3,Wordf $,
        Mdpreprms   = 4,Wordf $,
        Mdpreprm(n) = 4+(n),Wordf $,

        Mdrofmd     = Mdprofmd $,

        Mdevtmd     = Mdprofmd $,

        Mdround     = 2,Wordf $,
        Mdrocnt     = 3,Wordf $,

        MdstrSDB    = 2,Wordf $,
        Mdstrlen    = 3,Wordf $,
        Mdstrcnt    = 4,Wordf $,
        Mdstrflds   = 5,Wordf $,
        Mdstrfmd(n) = 5+2*(n),Wordf $,
        Mdstrflex(n)= 6+2*(n),Wordf $;


! Definitions of Mdv values for modes.

macro Mdvval(uniq,pi,dres,spc,io) =
        (pi)↑(Qwrdsiz-1) + (io)↑(Qwrdsiz-2) + (dres)↑(Qwrdsiz-3)
        + (spc)↑(Qwrdsiz-5) + (uniq) $;
```

```
bind
        Pile = 1, Simple = 0,
        Dressed = 1, Undressed = 0,
        Special = 1,
        Io = 1, Noio = 0,

        Doproc = 1^(Qwrdsiz-6);

bind Constbv
        Mdvint          = Mdvval( 0, Simple, 0        , 0      , Io   ):
        Mdvlint         = Mdvval( 1, Pile  , Undressed, Special, Io   ):
        Mdvreal         = Mdvval( 2, Pile  , Undressed, Special, Io   ):
        Mdvlreal        = Mdvval( 3, Pile  , Undressed, 0      , Io   ):
        Mdvchar         = Mdvval( 4, Simple, 0        , 0      , Io   ):
        Mdvbits         = Mdvval( 5, Simple, 0        , 0      , Io   ):
        Mdvbytes        = Mdvval( 6, Simple, 0        , 0      , Noio ):
        Mdvstring       = Mdvval( 7, Pile  , Dressed  , 0      , Io   ):
        Mdvbool         = Mdvval( 8, Simple, 0        , 0      , Io   ):
        Mdvvoid         = Mdvval( 9, 0     , 0        , 0      , Noio ):
        Mdvskip         = Mdvval(10, 0     , 0        , 0      , Noio ):
        Mdvjump         = Mdvval(11, 0     , 0        , 0      , Noio ):
        Mdvnil          = Mdvval(12, 0     , 0        , 0      , Noio ):
        Mdvout          = Mdvval(13, Pile  , Dressed  , 0      , 0    ):
        Mdvin           = Mdvval(14, Pile  , Dressed  , 0      , 0    ):
        Mdvoutb         = Mdvval(15, Pile  , Dressed  , 0      , 0    ):
        Mdvinb          = Mdvval(16, Pile  , Dressed  , 0      , 0    ):
        Mdvnumber       = Mdvval(17, Pile  , Dressed  , 0      , 0    ):
        Mdvrons         = Mdvval(18, Pile  , Dressed  , 0      , 0    ):
        Mdvunionof      = Mdvval(19, Pile  , Dressed  , 0      , 0    ):
        Mdvbnds         = Mdvval(20, Simple, 0        , 0      , Noio ):
        Mdvabsent       = Mdvval(21, Simple, 0        , 0      , 0    ):
        Mdvproc         = Mdvval(22, Pile  , Dressed  , 0      , Noio ):
        Mdvref          = Mdvval(23, Pile  , Dressed  , 0      , Noio ):
        Mdvstruct       = Mdvval(24, Pile  , Undressed, 0      , Noio ):
        Mdvrow          = Mdvval(25, Pile  , 0        , 0      , Noio ):
        Mdvevent        = Mdvval(26, Pile  , Dressed  , 0      , Noio ):
        Mdvcode         = Mdvval(27, Simple, 0        , 0      , Noio ):
        Mdvnonsimple    = Mdvval(28, Simple, 0        , 0      , Noio ):
        Mdvnonpile      = Mdvval(29, Pile  , Dressed  , 0      , Noio );

macro
        Callmode(M)     = (Tnonplain(M) eql 1) $,
        Codemode(M)     = (M[MdIO] eql Mdvcode[MdIO]) $,
        Eventmode(M)    = (M[MdIO] eql Mdvevent[MdIO]) $,
        Hipmode(M)      = (Thip(M) gtr 0) $,
        Intrealmode(M)  = (M[MdIO] leq Mdvlreal[MdIO]) $,
        Nonmode(M)      = (M[MdIO] gtr Mdvcode[MdIO]) $,
        Procmode(M)     = (M[MdIO] eql Mdvproc[MdIO]) $,
        Refmode(M)      = (M[MdIO] eql Mdvref[MdIO]) $,
        Rowmode(M)      = (M[MdIO] eql Mdvrow[MdIO]) $,
        Structmode(M)   = (M[MdIO] eql Mdvstruct[MdIO]) $,
        Undefinedmode(M)= (M[MdIO] eql Mdvabsent[MdIO]) $,
        Unionofmode(M)  = (M[MdIO] eql Mdvunionof[MdIO]) $,
        Unitedmode(M)   = (Tunited(M) gtr 0) $,
        Widenable(M)    = (M[MdIO] leq Mdvstring[MdIO]) $,
        Widentorow(M)   = (M[MdIO] geq Mdvbits[MdIO]) $,        ! Valid only if Widenable(M) is true
```

```
        Samesize(X1,X2) = ((X1) eqv (X2)) $;

external
 Reg0   Valuelength,

 Reg0   Tarith,
 Reg0   Tchars,
 Reg0   Thip,
 Reg0   Tsize,
 Reg0   Tnonplain,
 Reg0   Tunited,
        Tunion,

 Reg0   TXsize;

bind
        Xint    = 1,
        Xlint   = 2,
        Xreal   = 3,
        Xlreal  = 4,
        Xcompl  = 5,
        Xlcompl = 6,

        Xmaxsizint  = 2,
        Xmaxsizreal = 4,

        Xchar   = 1,
        Xstring = 2,
        Xchars  = (Xchars2 + Xchar) -2,

        Xskip   = 1,
        Xjump   = 2,
        Xnihil  = 3,

        Xout    = 1,
        Xin     = 2,
        Xoutb   = 3,
        Xinb    = 4,
        Xnumber = 5,
        Xrows   = 6;

!
! END OF MDSTRC.REQ
```

```
! LEXAN.BLI
!
```



```
module Lexan =
begin
!
!                                Lexical Analyzer
!                                ----------------
!
```

! The lexical analyzer converts a stream of characters into a stream of lexemes representing tokens of the ALGOL 68S language. The stream of characters is the source program and is supplied one character at a time by the system-dependent routine OSEsrcchar. The next lexeme of the program is obtained from the analyzer by a call on Lxscan. A description of the format and possible values of lexemes may be found in LXSTRC.REQ.

! A more specific list of the functions performed by the lexical analyzer is the following:

- Recognize tokens in the current stropping convention
- Construct lexemes
- Output program listing
- Ignore comments
- Process pragmats

! A source program may be represented in any of four stropping conventions. The current default is the POINT convention in which bold tokens (i.e. bold-tags and symbols represented by bold character sequences) are indicated by a prefix strop character. This strop character may be either a point (.) or an apostrophe ('). The other conventions available are UPPER, LOWER and RES. The use of the strop character is permitted in all of these. The occurrence of a strop followed by an alphabetic character always forces the following alphanumeric sequence to be bold.

! The UPPER convention uses upper case letters to indicate bold letters, while LOWER convention uses lower case letters. A strop may be used to override this as described above.

! In RES convention, typographical display features may no longer appear between the marks of a tag or the symbols of a denotation. A sequence of alphanumeric characters surrounded by disjunctors (non-alphanumeric characters or typographical display features) represents a bold symbol, if one exists with that particular spelling; otherwise it represents a tag. In RES stropping the first occurrence of a user-introduced bold-tag must be explicitly stropped.

! Known Nonconformance with the Mahsen-Boom Report
! - There is no intimidation character. In particular, there is no representation of the worthy character underscore.
! - Upper and lower case letters may be intermixed following a strop in UPPER (and LOWER); thus, .Begin is always allowed.
! - PAGE pragmat-item is not implemented.

```
!
!   - Apostrophe-image symbol is not implemented.
!   - The RES convention is different.  It probably does
!       not even conform to the Report because of the restriction on the
!       use of typographical display features.  Also, applied occurences
!       of bold-tags are reserved.
!   - $ is permitted as the first character of a tag.
!

switches nolist;
  require SYSPRM.REQ;
  require COMMON.REQ;
  require IOMACS.REQ;
  require LISPKG.REQ;
  require ERRCOM.REQ;
  require ERRLEX.REQ;
  require LXTABS.REQ;
  require LXSTRC.REQ;
  require MDSTRC.REQ;
  require LXVCOM.REQ;
  require LXVSY.REQ;
  require LXTABL.REQ;
  require MDTABL.REQ;
switches list;

external
        Errchar,
        Errixptr,
        Errnonblank,
        Errptr,
        Lineno,
 Pblock Faptr;          ! Pointer to first location in freearea

forward
        InitLx,         % global %
 Reg0   Lxerr,
        Outsrc,
        Nextch,
 Reg0   Nextch,
 Reg0   Scanch,
        TietABC,
        Tbold,
!       Tbreak,
        Putchr,
        Putcvt,
        Uppercase,
        Hashin,         % global %
!       Doprag,
        Lxscan,         % global %
        Lx,
        Errorchar,
        Gotprimdon,
        Gotstrgdon,
 Reg0   Scantax,
        Hashbold,
        Gottax,
        Gotbold,
        Endoffile,
```

```
        Getopr;

own
        Char,
 Charvect Errbuf[Cbufsize],
        Index,
        Inpragment,
        Longcnt,
        Lookahead,
 Charvect Srcbuf[Cbufsize],
        Srcptr,
        Srcstch,
        Symcnt,
        Symptr,
 Bitvect Type;

macro
        Srcstat = Srcbuf[6] $;                  ! Nptr

bind
        Srcline = Srcbuf[Iptrndx(2)],           ! Iptr
        Srctext = Srcbuf[Iptrndx(8)],           ! Iptr
        Srcend  = Srcbuf[Iptrndx(Cbufsize+1)],  ! Iptr
        Errtext = Errbuf[Iptrndx(8)];           ! Iptr


comment ! InitLx
!
! Function: Perform per-compilation initialization required by
!   the Lexical Analyzer.
!
global routine InitLx =
    begin
    local Ptr;
    Index←Control;
    Inpragment←zed;
    Lookahead←0;
    Longcnt←0;

    Ptr←Iptrtochars(Errbuf);
    until .Ptr eql Errtext do replacel(Ptr," ");
    Errptr←Errixptr←.Ptr;
    Errchar←" ";
    Errnonblank←zed;

    Ptr←Iptrtochars(Srcbuf);
    replacel(Ptr,";");
    until .Ptr eql Srctext do replacel(Ptr," ");
    Srcptr←.Ptr;

    Lineno←1;
    Zerocor(HT,HTsize);
    Faptr←Lextab<0,0>;
    until .Faptr[Link] eql Minus1[Link] do
        begin
        if .Faptr[Lxv] eql Lxvtag then
            Faptr[Lxp]←0;
```

```
        Hashin();
        end;
    novalue
    end;


comment ! Lxerr(N)
!
! Function: Print error message unless currently processing inside
!   a pragment.
!
! Value: The Error lexeme.
!
! Inputs
!   N           - Error code
!   Inpragment - nonzed iff scanning pragment
!
routine Reg0 Lxerr(N) =
    begin
    if .Inpragment eql 0 then
        Error(.N);
    Lexerror
    end;


comment ! Outsrc
!
! Function: Output a line of source listing on the listing device.
!   If an error occurred in the line or the line was ignored due to a
!   previous error, then a line of error indication is also output.
!   If an error occurred in the line, then Errdev will be nonzed and thus
!   all output will go to the error device also.
!
routine Outsrc =
    begin
    Cvtdecz(Srcline,.Lineno,3);
    Lineno←.Lineno+1;
    Srcptr←ItoNptr(.Srcptr);
    incp(Srcptr);
    OSElstline(Nptrtochars(Srcbuf),.Srcptr,.Errdev);
    Srcptr←Srctext;
    if .Errnonblank neq 0 then
        begin
        Errptr←ItoNptr(.Errptr);
        incp(Errptr);
        OSElstline(Nptrtochars(Errbuf),.Errptr,.Errdev);
        if .Errchar eql " " then
            Errnonblank←zed;
        Errdev←zed;
        end;
    Errptr←Errixptr←Errtext;
    Srcstat←.Srcstch;
    novalue
    end;


comment ! Nowch
```

```
!
! Function: Get a non-control character from the input stream.
!
routine Nauch =
    begin
    local C;
    do C←OSEsrcchar() until .Chartype[.C] neq Ctctrl;
    .C
    end;


bind
    Skipnone    = EOL,
    SkipEOL     = Space,
    Skipspaces  = Errch,
    Skipinprag  = Letter;


comment ! Nextch(Level) and Scanch(Level)
!
! Function: Get the next acceptable character from the source input.
!   Level is used to indicate which characters are acceptable.  Nextch
!   always uses a new input character as the first character to consider;
!   Scanch starts with the current character.
!
! Outputs
!   Char  - the current input character
!   Type  - character type of current input character
!   Index - index of character class of current input character
!
routine Reg0 Nextch(Level) =
    begin
    do begin
        if .Index eql EOL then Outsre();
        if .Lookahead neq 0
            then (Char←.Lookahead; Lookahead←0)
            else Char←Nauch();
        Type←.Chartype[.Char];
        Index←.Type[Tpindex];
        if .Index eql EOF
            then (if .Srcptr neq Srctext then Outsrc() )
          elif .Index gtr EOL
            then begin
                if .Index eql Point then
                    if (Lookahead←Nauch()) geq "A" then
                        begin
                        Type←Ctstrop;
                        Index←Strop;
                        end;
                if .Srcptr eql Srcend then Outsrc();
                replacel(Srcptr,.Char);
                replacel(Errptr,.Errchar);
                end;
        end
      while .Index iss .Level;
    novalue
    end;
```

```
routine RegB Scanch(Level) =
    begin
    if .Index lss .Level then Nextch(.Level);
    novalue
    end;


comment ! Tdigit,Tletter,Tupper,Tlower,Tdigter,Tdigpor,
!         Tdigwor,Thexit,TletterR,Tpow10,TletABC,Tbold,
!         Tbreak(Bold)
!
! Value: true if the current character is --
!   (Tdigit)    a digit
!   (Tletter)   a letter
!   (Tupper)    an upper case letter
!   (Tlower)    a lower case letter
!   (Tdigter)   a digit or letter
!   (Tdigpor)   a digit or upper case letter
!   (Tdigwer)   a digit or lower case letter
!   (Thexit)    a hexidecimal digit
!   (TletterR)  a letter R
!   (Tpow10)    a power of ten choice
!   (TletABC)   a non-bold letter symbol
!   (Tbold)     a possible first character of bold tag
!   (Tbreak)    not allowed in tag or bold-tag
!
macro
        Tdigit(dum)     = (.Type[Tpdigit]) $,
        Tletter(dum)    = (.Type[Tpletter] neq 0) $,
        Tupper(dum)     = (.Type[Tpupper]) $,
        Tlower(dum)     = (.Type[Tplower]) $,
        Tdigter(dum)    = (.Type[Tpdigter] neq 0) $,
        Tdigpor(dum)    = (.Type[Tpdigpor] neq 0) $,
        Tdigwor(dum)    = ((.Type and Tpdigwor) neq 0) $,
        Thexit(dum)     = (.Type[Tphexit] and (TletABC() or Tdigit())) $,
        TletterR(dum)   = (TletABC() and Uppercase() eql "R") $,
        Tpow10(dum)     = (.Char eql "\" or
                                (TletABC() and Uppercase() eql "E")) $;

routine TletABC =
    case .GBLpragflags[Prgstrop] of
      set
        bool(Tletter());
        bool(Tletter());
        bool(Tlower());
        bool(Tupper())
      tes;

routine Tbold =
    if .Index eql Strop
        then true
        else case .GBLpragflags[Prgstrop] of
                set
                  % RES   %  bool(Tletter());
                  % STROP %  false;
                  % UPPER %  bool(Tupper());
```

```
                % LOWER %  bool(Tlower())
               tes;

macro Tbreak(Bold) =
    begin
    case .CBLpragflags[Prgstrop] of
      set
        % RES   %  not Tdigter();
        % STROP %  not Tdigter();
        % UPPER %  if (Bold) neq 0 then not Tdigper() else not Tdigwer();
        % LOWER %  if (Bold) neq 0 then not Tdigwer() else not Tdigper()
      tes
    end%;

comment ! Putchr and Putcvt
!
! Function: Store character in Lexeme Table entry under construction
!   at top of freearea.  Update character count.  Check if there is sufficient
!   space left in freearea.  Putcvt, in addition, first converts any
!   lower case letter to the corresponding upper case letter.
!
! Inputs
!   Symptr - Nptr to character string in new Lexeme Table entry
!   Symcnt - count of characters in current symbol
! Outputs
!   Symptr - updated after character is stored
!   Symcnt - incremented by one
!
routine Putchr =
    begin
    if modulo(.Symcnt,Qchword) eql 0 then
        begin
        Storetest(.Symptr<Addrf>);
        (.Symptr)<Wordf>←0;
        end;
    replaceN(Symptr,.Char);
    incp(Symptr);
    Symcnt←.Symcnt+1;
    novalue
    end;

routine Putcvt =
    begin
    Char←Uppercase();
    Putchr();
    novalue
    end;


comment ! Uppercase
!
! Function: Compute the upper case equivalent of the current input character.
!
routine Uppercase =
    if Tlower() then .Char and (not #40) else .Char;
```

```
comment ! Hashin
!
! Function: Search Lexeme Table for lexeme sitting at start of freearea.
!   If lexeme is already in table, then return pointer to this
!   old lexeme.  If it is not in the table and Noenter is not set and
!   we are not inside a pragment, then enter the lexeme in the table
!   and return a pointer to the new lexeme.  If lexeme is not found and
!   a new entry is not made then return zero.
!
global routine Hashin =
    begin
    local Total,Ptr;
    Total←0;
    Ptr←Nptrtochars(Faptr[Lxsym]);
    decr I from .Faptr[Lxcount]*Qchswrd-1 to 0 do
        begin
        Total←.Total+scanN(Ptr);
        incp(Ptr);
        end;
    Total←.Total and HTmask;
    do Total←.Total-HTsize until .Total lss 0;
    Total←.Total+HTsize;
    Noenter←.Noenter or .Inpragment;
    Find(HT[.Total],HTD(.Faptr[Lxcount]+1))
    end;


comment ! Doprag(N)
!
! Function: Carry out the semantics of the pragmat specified by N.
!
macro Doprag(N) =
    begin
    case (N) of
      set

        ! RES, POINT, UPPER, LOWER
        Newstrop←(N);
        Newstrop←(N);
        Newstrop←(N);
        Newstrop←(N);

        ! WARNINGS and NOWARNINGS
        GBLpragflags[Prgwarn]←true;
        GBLpragflags[Prgwarn]←false;

        ! LISTING and NOLISTING
        GBLpragflags[Prglist]←true;
        GBLpragflags[Prglist]←false;

        ! PAGE

      tes;
    novalue
    end$;
```

```
comment ! Lxscan
!
! Function: Scan a token from the input and return its lexeme.
!   A token consists of an optional pragment (pragmat or comment) followed
!   by a symbol.
!
global routine Lxscan =
    begin
    label L;
    local Lexeme Lex;
    while begin
        Srcstch←" ";
        Lex←Lx();
        .Lex[Lxv] eql 0
        end
      do begin
        local Newstrop;
        Inpragment←Srcstch←.Lex[Lxp];
        Newstrop←.GBLpragflags[Prgstrop];
        repeat begin
            local Lexeme Lex2;
            Scanch(Skipinprag);
            Lex2←Lx();
            if .Lex2 neq Lexerror then
                if .Lex2 eql Lexstop
                    then (Error(ELx2); exitloop)
                  elif .Lex2 eql .Lex
                    then exitloop
                  elif .Lex2[Lxv] eql Lxvprgitem and .Inpragment eql "P"
                    then Doprag(.Lex2[Lxp]);
            end;
        Inpragment←zad;
        GBLpragflags[Prgstrop]←.Newstrop;
        end;
    Liselect .Lex[Lxv] of
      nset
        Lxvlong: exitL Longsent←.Longsent+1;
        Lxvshort: exitL Longsent←.Longsent-1;
        Alielse: Longsent←0
      tesn;
    .Lex
    end;


comment ! Lx
!
! Function: Scan a symbol from the input.
!
! Value: Lexeme for the symbol.
!
routine Lx =
    begin
    local Pblock Lex;
    bind LxRouts = uplit(
        0,              ! Control
        0,              ! EOL
```

```
        0,              ! Space
        Errorchar,      ! Errch
        Getprimdon,     ! Digit
        Getprimden,     ! Point
        Getstrgdon,     ! Quote
        Getopr,         ! Punct
        Getopr,         ! Plsmin
        Gettax,         ! Letter
        Gettax,         ! Dollar
        Getbold,        ! Strop
        Endoffile,      ! EOF
        Getopr );       ! Prag

    do begin
        Scanch(Skipspaces);
        Errlxptr←.Errptr;
        Symcnt←0;
        Symptr←Nptrtochars(Faptr[Lxsym]);
        Lex←(.LxRouts[.Index])()
        end
      until .Lex neq Lexerror  or  .Inpragment neq 0;
    .Lex
    end;


comment ! Errorchar
!
! Function: Deal with situation where illegal character occurs in the source.
!
! Value: Error lexeme.
!
routine Errorchar = (Lxerr(ELx6); Nextch(Skipnone); Lexerror);


comment ! Getprimdon
!
! Function: Scan a primitive denotation from the input.
!
! Value: Lexeme for the denotation.
!
routine Getprimdon =
    begin
    local State, Mode M;
    external Lengthen, Cvtb,Cvti,Cvtll,Cvtr,Cvtlr;
    bind Vector Cvtrtn = uplit(Cvtb,Cvti,Cvtll,Cvtr,Cvtlr);

    State←(if .Index eql Point then 1 else 0);
    repeat
        begin
        Putcvt();
        Nextch(if .GBLpragflags[Prgstrop] eql 0 then Skipnone else Skipspaces);
        case .State of
          set
            ! 0: scan digits
            if .Index eql Point
                then State←1
              elif TletterR()
```

```
            then State←6
          elif Tpow10()
            then (Char←"E"; State←3)
          elif not Tdigit()
            then (M←Mdint; exitloop);

        ! 1: fixed-point-numeral must follow point in fractional-part
        if Tdigit()
            then State←2
            else return Lxerr(ELx4);

        ! 2: scan digits of fractional-part
        if Tpow10()
            then (Char←"E"; State←3)
          elif not Tdigit()
            then (M←Mdreal; exitloop);

        ! 3: check for plusminus in exponent-part
        if .Index eql Plsmin
            then State←4
          elif Tdigit()
            then State←5
            else return Lxerr(ELx4);

        ! 4: fixed-point-numeral must follow plusminus
        if Tdigit()
            then State←5
            else return Lxerr(ELx4);

        ! 5: scan fixed-point-numeral in exponent-part
        if not Tdigit()
            then (M←Mdreal; exitloop);

        ! 6: digits must follow letter-r in bits-denotation
        if Thexit()
            then State←7
            else return Lxerr(ELx4);

        ! 7: scan digits in bits-denotation
        if not Thexit()
            then (M←Mdbits; exitloop)
      tes;
    end;
  M←Lengthen(.M,.Longcnt);
  replaceN(Symptr,0);
  if (.Cvtrtn[Tarith(.M)])() gtr 8 then return Lxerr(ELx4);
  Faptr[Lxdenmd]←.M;
  Faptr[Lxv]←Lxvprimden;
  Faptr[Lxp]←8;
  Faptr[Lxcount]←Valuelength(.M)+1;
  Faptr[Lxtoken]←TkDenot;
  Hashin()
  end;


comment ! Getstrgden
!
```

```
! Function: Scan a string denotation from the input.
!
! Value: Lexeme for the string denotation.
!
routine Getstrgden =
    begin
    external Cvtstrg;
    Symptr←Nptrtochars(Fptr[Lxstrgrep]);
    repeat begin
        Srestch←"S";
        Nextch(SkipEOL);
        if .Index eql EOF
            then return Lxerr(El.x1)
          elif .Index neq Quote
            then Putchr()
            else begin
                Srestch←" ";
                Nextch(Skipnone);
                if .Index eql Quote
                    then Putchr()
                    else begin
                        Scanch(Skipspaces);
                        if .Index neq Quote
                            then exitloop;
                        end
                end
        end;
    Cvtstrg(.Symcnt);
    if .Symcnt eql 1
        then (Symcnt←0; Fptr[Lxdennd]←Mdchar; Fptr[Lxdenrep]←.Fptr[Lxstrgrep])
        else (Fptr[Lxdennd]←Mdstring; Fptr[Lxdenrep]←.Symcnt);
    Fptr[Lxv]←Lxvstrgden;
    Fptr[Lxp]←0;
    Fptr[Lxcount]←divide(.Symcnt+3*Zchswrd-1,Zchswrd);
    Fptr[Lxtoken]←TkDenot;
    Hashin()
    end;


comment ! Scantax(Bold)
!
! Function: Scan characters contained in a tag or bold-tag.
!
! Inputs
!   Bold - nonzed iff scanning bold-tag
!
routine Reg0 Scantax(Bold) =
    begin
    do begin
        Putcvt();
        Nextch(if .Bold neq 0 then Skipnone else Skipspaces);
        end
      until Tbreak(.Bold);
    Fptr[Lxp]←0;
    Fptr[Lxcount]←divide(.Symcnt+Qchswrd-1,Qchswrd);
    novalue
    end;
```

```
comment ! Hashbold
!
! Function: Scan a bold-tag from the input.
!
! Value: Lexeme for the bold-tag.
!
routine Hashbold =
    begin
    Scantax(nonzed);
    Fptr[Lxv]←Lxvtab;
    Fptr[Lxtoken]←TkBold;
    Hashin()
    end;


comment ! Gettax
!
! Function: Scan a TAX-symbol from the input.  In RES convention the
!   symbol scanned may be a TAB-symbol, otherwise it is a TAG-symbol.
!
! Value: Lexeme for the symbol.
!
routine Gettax =
    begin
    if Tbold()
        then begin
            local Val;
            if .GBLpragflags[Prgstrop] eql 0 then
                Nocntor←nonzed;
            if (Val←Hashbold()) neq 0 then
                return .Val;
            end
        else
            Scantax(zed);
    Fptr[Lxv]←Lxvtag;
    if .Inpragment eql "C"
        then Lexerror
        else begin
            Fptr[Lxtoken]←(if .Inpragment eql 0 then TkTag else TkPragitem);
            Hashin()
            end
    end;


comment ! Gotbold
!
! Function: Scan a bold-tag from the input.
!
! Value: Lexeme for the bold-tag.
!
routine Gotbold =
    begin
    local Val,Savstrop;
    Nextch(Skipnone);
    if not Tletter() then
```

```
        return Lxerr(ELx5);
    Savstrop←.GBLpragflags[Prgstrop];
    GBLpragflags[Prgstrop]←1;
    Val←Hashbold();
    GBLpragflags[Prgstrop]←.Savstrop;
    .Val
    end;


comment ! Endoffile
!
! Function: Return STOP lexeme to indicate source end-of-file has occurred.
!
routine Endoffile = return Lexstop;


comment ! Getopr
!
! Function: Scan an operator from the input.
!
! Value: Lexeme for the operator or Error lexeme.
!
routine Getopr =
    begin
    local Lxptr,S,Olds;
    Lxptr←Lexerror;
    S←0;
    do begin
        Olds←.S;
        if .Char eql .Opchtable[.S,Otchar]
            then begin
                Nextch(Skipnone);
                Lxptr←.Opchtable[.S,Otlex];
                if .Opchtable[.S,Otnext] then S←.S+Otrowsize;
                end
            else begin
                S←.S+.Opchtable[.S,Otalt];
                end;
        end
      until .S eql .Olds;
    if .Lxptr eql Lexerror
        then Lxerr(ELx3)
        else Opixtable[.Lxptr]<0,8>
    end;


global routine Lxline =
    begin
    Scanch(Skipspaces);
    novalue
    end; .


end
eludom
```

```
! SRTABS.BLI
!

! Copyright 1977        P. Hibbard and P. Knueven
!                       Pittsburgh, Pennsylvania

module SRtabs=
begin
!
!                       Semantic Tables
!
!       This module defines the entries initially contained in the
!       Mode Table, Operator Table and Symbol Table.
!
!       Kernal call tables
!           in  - %K>%  ...  %<K%
!           out - %K>%% ... %%<K%
!       Hydra tables (other than Kernal call)
!           in  - %H>%  ...  %<H%
!           out - %H>%% ... %%<H%
!
switches nolist;
  require SYSPRM.REQ;
  require COMMON.REQ;
  require LISPKG.REQ;
  require LXVCOM.REQ;
  require LXVSY.REQ;
  require LXPVAL.REQ;
  require EMIT.REQ;
  require MDSTRC.REQ;
  require STSTRC.REQ;
  require OPSTRC.REQ;
  require LXTABL.REQ;
switches list;

macro
    empty = 0:0 $,

    Defstid(offst,md,lx) =
        0, Qadrinc+uplit(0, lx), 0, Hfwords(Stbtype(Stdidty,Stbdefid), 0),
        (offst + ZRsizIRbase + ZRsizLEObase + 1), 0, md  $,

    Defstopr1(prio,std,lx) =
        0, Qadrinc+uplit(0,lx), 0, Hfwords(Stbtype(0,Stbdefprio),0),
        std, 0, prio $,

    Defstopr2(prio,std) =
        0, 0, 0, Hfwords(Stbtype(0,Stbdefprio),0),
        std, 0, prio $,

    Defdonint(val) =
        Lexm(Lxvprimdon, val), Hfwords(2,TkDonot), Mdint, val $,

    Deftag(len,str) =
        Lexm(Lxvtag, 0), Hfwords((len+Qchswrd-1)/Qchswrd, TkTag), str  $,

    Defbold(lv,lp,len,str) =
```

```
        0, Lexm(lv,lp), Hfwords((len+Qchswrd-1)/Qchswrd, TkBold), str  $,

    Defpragitem(lp,len,str) =
        0, Lexm(Lxvpragitem,lp), Hfwords((len+Qchswrd-1)/Qchswrd, TkPragitem), str  $,

    Defopr(len,str) =
        Lexm(Lxvopr,0), Hfwords((len+Qchswrd-1)/Qchswrd, TkBold), str  $,


    Defcode0(yield) = Mdvcode+Doproc, yield, 0 $,
    Defcode1(p1,yield) = Mdvcode, yield, 1, p1 $,
    Defcode2(p1,p2,yield) = Mdvcode, yield, 2, p1, p2 $,

    Defproc0(yield) = Mdvproc+Doproc, yield, 0 $,
    Defproc1(p1,yield) = Mdvproc, yield, 1, p1 $,
    Defproc2(p1,p2,yield) = Mdvproc, yield, 2, p1, p2 $,
    Defproc3(p1,p2,p3,yield) = Mdvproc, yield, 3, p1, p2, p3 $,
    Defproc4(p1,p2,p3,p4,yield) = Mdvproc, yield, 4, p1, p2, p3, p4 $,
    Defproc6(p1,p2,p3,p4,p5,p6,yield) = Mdvproc, yield, 6, p1, p2, p3, p4, p5, p6 $,

    Defref(md) = Mdvref, md $,
    Defrow(md,rcnt) = Mdvrow, md, rcnt $,
    Defunion2(m1,m2) = Mdvunionof, 2, m1, m2 $,
    Defstruct2(m1,l1,m2,l2) = Mdvstruct, 0, 0, 2, m1,l1, m2,l2 $,
    Defstruct4(m1,l1,m2,l2,m3,l3,m4,l4) = Mdvstruct, 0, 0, 4, m1,l1, m2,l2, m3,l3, m4,l4 $;

bind dummy = 0;

dataarea(Mds0)
    gbiname(Mdint,      Mdvint),
    gbiname(Mdlint,     Mdvlint),
    gbiname(Mdbits,     Mdvbits),
    gbiname(Mdbytes,    Mdvbytes),
    gbiname(Mdreal,     Mdvreal),
    gbiname(Mdlreal,    Mdvlreal),
    gbiname(Mdbool,     Mdvbool),
    gbiname(Mdchan,     Mdvnonsimple),
    gbiname(Mdchar,     Mdvchar),
    gbiname(Mdstring,   Mdvstring),
    gbiname(Mdfile,     Mdvnonsimple),
    gbiname(Mdnone,     Mdvnonsimple),
    gbiname(Mdvoid,     Mdvvoid),
    gbiname(Mdskip,     Mdvskip),
    gbiname(Mdjump,     Mdvjump),
    gbiname(Mdnil,      Mdvnil),
    gbiname(Mdout,      Mdvout),
    gbiname(Mdin,       Mdvin),
    gbiname(Mdoutb,     Mdvoutb),
    gbiname(Mdinb,      Mdvinb),
    gbiname(Mdnumber,   Mdvnumber),
    gbiname(Mdrows,     Mdvrows),
    gbiname(Mdbnds,     Mdvbnds),
    gbiname(Mdabsent,   Mdvabsent),
    gbiname(Mdrout,     Mdvproc)
dataend;

dataarea(Mds1)
```

```
    data(               0),
    gbiname(Mdcompl,    Defstruct2(Mdreal,Lexre,Mdreal,Lexim)),
    data(               0),
    gbiname(Mdlcompl,   Defstruct2(Mdlreal,Lexre,Mdlreal,Lexim)),
    data(               0),
    gbiname(Rowbool,    Defrow(Mdbool, 1)),
    data(               0),
    gbiname(Rowchar,    Defrow(Mdchar, 1)),
    data(               0),
    locname(Rowout,     Defrow(Mdout, 1)),
    data(               0),
    locname(Rowin,      Defrow(Mdin, 1)),
    data(               0),
    locname(Rowoutb,    Defrow(Mdoutb, 1)),
    data(               0),
    locname(Rowinb,     Defrow(Mdinb, 1)),
    data(               0),
    gbiname(Prcbmds,    Defproc0(Mdbmds)),                  ! Rowed mode indicant
    data(               0),
    locname(Prc2iv,     Defproc2(Mdint,Mdint, Mdvoid)),
    data(               0),
    locname(Refint,     Defref(Mdint)),
    data(               0),
    locname(Reffile,    Defref(Mdfile)),
    data(               0),
    gbiname(Refstring,  Defref(Mdstring))
dataend;

dataarea(Mds2)
    data(               0),
    gbiname(Prcrfv,     Defproc1(Reffile, Mdvoid)),         ! Reset, Close, Scratch
    data(               0),
    locname(Prcbf,      Defproc1(Reffile, Mdbool)),
    data(               0),
    locname(Prcibb,     Defproc3(Mdint,Mdint,Mdbits, Mdbool)),
    data(               0),
    locname(Codibtv,    Defcode2(Mdint,Mdbits, Mdvoid))
dataend;

dataarea(Mds3)
    data(               0),
    locname(Prcreal,    Defproc0(Mdreal)),                  ! Random
    data(               0),
    locname(Prcrr,      Defproc1(Mdreal, Mdreal)),          ! Sin, Cos, etc.
    data(               0),
    locname(Prcbtrb,    Defproc1(Rowbool, Mdbits)),         ! Bits pack
    data(               0),
    locname(Prcbyst,    Defproc1(Mdstring, Mdbytes)),       ! Bytes pack
    data(               0),
    locname(Prcrfir,    Defproc1(Refint, Mdreal)),          ! Next random
    data(               0),
    locname(Prcif,      Defproc1(Reffile, Mdint)),          ! Char number, etc.
    data(               0),
    locname(Prcchf,     Defproc1(Reffile, Mdchan)),         ! Chan
    data(               0),
    locname(Prcvout,    Defproc1(Rowout, Mdvoid)),          ! Print, Write
    data(               0),
```

```
    locname(Prcvin,     Defproc1(Rowin, Mdvoid)),           ! Read
    data(               0),
    locname(Prcvob,     Defproc1(Rowoutb, Mdvoid)),         ! Write bin
    data(               0),
    locname(Prcvib,     Defproc1(Rowinb, Mdvoid)),          ! Read bin
    data(               0),
    locname(Prcvfout,   Defproc2(Refile,Rowout, Mdvoid)),   ! Put
    data(               0),
    locname(Prcvfin,    Defproc2(Refile,Rowin, Mdvoid)),    ! Get
    data(               0),
    locname(Prcvfp,     Defproc2(Refile,Rowoutb, Mdvoid)),  ! Putbin
    data(               0),
    locname(Prcvfg,     Defproc2(Refile,Rowinb, Mdvoid)),   ! Getbin
    data(               0),
    locname(Prcvfs,     Defproc2(Refile,Mdstring, Mdvoid)), ! Make term
    data(               0),
    locname(Prcvrfrs,   Defproc2(Refile,Refstring, Mdvoid)), ! Associate
    data(               0),
    locname(Prcon,      Defproc2(Refile,Prcbf, Mdvoid)),    ! On routines
    data(               0),
    locname(Prcsn1i,    Defproc2(Mdnumber,Mdint, Mdstring)), ! Whole
    data(               0),
    locname(Prcsn2i,    Defproc3(Mdnumber,Mdint,Mdint, Mdstring)), ! Fixed
    data(               0),
    locname(Prcifsc,    Defproc3(Refile,Mdstring,Mdchan, Mdint)), ! Open
    data(               0),
    locname(Prcsn3i,    Defproc4(Mdnumber,Mdint,Mdint,Mdint, Mdstring)), ! Float
    data(               0),
    locname(Prcvf3i,    Defproc4(Refile,Mdint,Mdint,Mdint, Mdvoid)), ! Set
    data(               0),
    locname(Prcest,     Defproc6(Refile,Mdstring,Mdchan,Mdint,Mdint,Mdint, Mdint)), ! Establish
    data(               0),
    locname(Codcibtv,   Defcode0(Codibtv)),                 ! Sys trace
    data(               0),
    locname(Codp2ivv,   Defcode1(Prc2iv, Mdvoid)),          ! On tick
    data(               0),
    locname(Codpiibbv,  Defcode1(Prciibb, Mdvoid)),         ! On error
    data(               0),
    locname(Codcibtvv,  Defcode1(Codibtv, Mdvoid)),         ! On sys trace
    data(               0),
    locname(Codiv,      Defcode1(Mdint, Mdvoid))            ! Warning level
dataend;


%H>%
dataarea(Hyd0)
    gbiname(Mdtime,     Mdvnocmpile)
dataend;

dataarea(Hyd1)
    data(               0),
    locname(Reftime,    Defref(Mdtime))
dataend;

dataarea(Hyd2)
    data(               0),
    locname(Codv,       Defcode0(Mdvoid)),
```

```
    data(               0),
    locname(Codrftv,    Defcodel(Reftime, Mdvoid)),
    data(               0),
    locname(Codtmrf,    Defcodel(Mdtime, Mdreal))
dataend;
%<H%


%K>%%
dataarea(Ker0)
    gblname(Mdslot,     Mdvmexpile),
    gblname(Mdstep,     Mdvmexsimple)
dataend;

dataarea(Ker1)
    data(               0),
    locname(Mdrtsmask,  Defstruct4(Mdbits,Lexaux,Mdbits,Lexgen,Mdbool,Lexamp,Mdbool,Lextemp)),
    data(               0),
    locname(Rowint,     Defrow(Mdint, 1)),
    data(               0),
    locname(Rowstep,    Defrow(Mdstep, 1))
dataend;

dataarea(Ker2)
    data(               0),
    locname(Mdwalk,     Defref(Rowstep))
dataend;

dataarea(Ker3)
    data(               0),
    locname(Mdpath,     Defstruct2(Mdslot,Lexs,Mdwalk,Lexw))
dataend;

dataarea(Ker4)
    locname(Mdalgparm,  Defunion2(Mdpath,Mdslot)),
    locname(Mdpathslot, Defunion2(Mdpath,Mdslot))
dataend;

dataarea(Ker5)
    data(               0),
    locname(Mdarglist,  Defrow(Mdalgparm, 1)),
    data(               0),
    locname(Prcstep,    Defproc0(Mdstep)),
    data(               0),
    locname(Prcpsv,     Defproc1(Mdpathslot, Mdvoid)),
    data(               0),
    locname(Prcpsb,     Defproc1(Mdpathslot, Mdbool)),
    data(               0),
    locname(Prcpss,     Defproc1(Mdpathslot, Mdstep)),
    data(               0),
    locname(Prcspsv,    Defproc2(Mdslot,Mdpathslot, Mdvoid)),
    data(               0),
    locname(Prcpssv,    Defproc2(Mdpathslot,Mdslot, Mdvoid)),
    data(               0),
    locname(Prcpssb,    Defproc2(Mdpathslot,Mdslot, Mdbits)),
    data(               0),
    locname(Prcsbv,     Defproc2(Mdslot,Mdbits, Mdvoid)),
```

```
    data(                   0),
    locname(Prcpsrtv,       Defproc2(Mdpathslot,Mdrtsmask, Mdvoid)),
    data(                   0),
    locname(Prcpssrtv,      Defproc3(Mdpathslot,Mdslot,Mdrtsmask, Mdvoid)),
    data(                   0),
    locname(Prcpssrts,      Defproc3(Mdpathslot,Mdslot,Mdrtsmask, Mdstep)),
    data(                   0),
    locname(Prcsspsv,       Defproc3(Mdslot,Mdslot,Mdpathslot, Mdvoid)),
    data(                   0),
    locname(Cods,           Defcode0(Mdslot)),
    data(                   0),
    locname(Codist,         Defcode1(Mdint, Mdstep)),
    data(                   0),
    locname(Codriw,         Defcode1(Rowint, Mdwalk)),
    data(                   0),
    locname(Codisl,         Defcode1(Mdint, Mdslot))
dataend;

dataarea(Ker6)
    data(                   0),
    locname(Prcspsab,       Defproc3(Mdslot,Mdpathslot,Mdarglist, Mdbits)),
    data(                   0),
    locname(Prcspswal,      Defproc4(Mdslot,Mdpathslot,Mdwalk,Mdarglist, Mdbits))
dataend;
%%<K%


structure Imode[I,P,S] = (.Imode+Qadrince(.I-1))<.P,.S>;

map Imode
%H>%    Reftime:Codv:Codrftv:Codtmrl: %<H%
%K>%%   Mdwalk:Rowint:Rowstep:Mdarglist:Mdpath:Mdrtsmask:
        Prcpsv:Prcpsrtv:Prcspsv:Prcspswal:
        Prcpssv:Prcprtv:Prcpsb:Prcpssrts:Prcpss:Prcspsab:
        Prcpssb:Prcsbv:Prcstep:Prcsspsv:
        Cods:Codist:Codriw:Codisl: %%<K%
        Refint:Reffile:Refstring:
        Rowbool:Rowchar:Rowout:Rowin:Rowoutb:Rowinb:
        Mdcompl:Mdlcompl:
        Prcbuds:Prcreal:Prcrr:Prcbtrb:Prciibb:
        Prcif:Prcbf:Prcchf:Prcvob:Prcvib:Prcon:Prcsnli:Prcbyst:Prcrfir:
        Prcvf3l:Prcrfv:Prcvout:Prcvin:Prcvfout:Prcvrfrs:Prcvfs:
        Prcsn2l:Prcsn3l:Prcost:Prcvfin:Prcvfp:Prcvfg:Prcifsc:Prc2iv:
        Codibtv:Codp2lvv:Codpilbbv:Codcibtv:Codcibtvv:Codiv;


globaldata(Vector,Idata,)
        RefL, Refint[Link],Reffile[Link],Refstring[Link],
%H>%        Reftime[Link], %<H%
%K>%%       Mdwalk[Link], %%<K%
            0,
        RowL, Rowbool[Link],Rowchar[Link],Rowout[Link],Rowin[Link],
            Rowoutb[Link],Rowinb[Link],
%K>%%       Rowint[Link],Rowstep[Link],Mdarglist[Link], %%<K%
            0,
        StructL, Mdcompl[Link],Mdlcompl[Link],
%K>%%       Mdpath[Link],Mdrtsmask[Link], %%<K%
```

```
            0,
        ProcL, Prcbnds[Link],Prcreal[Link],Prcrr[Link],
            Prcbtrb[Link],Prcif[Link],Prcbf[Link],Prcchf[Link],
            Prcvf3i[Link],Prcrfv[Link],Prcvout[Link],Prcvin[Link],
            Prcvob[Link],Prcvib[Link],Prcvfs[Link],Prcvrfrs[Link],
            Prcvfout[Link],Prcvfin[Link],Prcvfp[Link],Prcvfg[Link],
            Prcon[Link],Prcsn1i[Link],Prcsn2i[Link],Prcsn3i[Link],
            Prcifsc[Link],Prcast[Link],Prcbyst[Link],Prcrfir[Link],
            Prc2iv[Link],Prciihb[Link],
%K>%%       Prcspsval[Link],Prcpsv[Link],Prcspsv[Link],Prcpsrtv[Link],
            Prcpssv[Link],Prcportv[Link],Prcpsb[Link],Prcpsrts[Link],
            Prcpss[Link],Prcspsab[Link],Prcpsnb[Link],Prcsbv[Link],
            Prcstep[Link],Prcspsv[Link], %%<K%
            0,
        CodeL, Codibtv[Link],Codcibtv[Link],Codp2ivv[Link],Codpiibbv[Link],
            Codcibtvv[Link],Codiv[Link],
%H>%        Codv[Link],Codrftv[Link],Codtmri[Link], %<H%
%K>%%       Cods[Link],Codist[Link],Codriw[Link],Codisi[Link], %%<K%
            0,
        EventL, 0,
        0);


bind    ! Indexes into the Xmodes table (see SEMR4.BLI)

        Mint = 0, Mlint = 1, Mreal = 2, Mlreal = 3, Mcompl = 4, Mlcompl = 5,
        Mchar = 6, Mstring = 7, Mbool = 8, Mbits = 9,
        Msema = 10, Mvoid = 11;


bind
    Obabs       = uplit(    Hfwords(Idmon, Pabsi), Hfwords(Mint, Mint),
                            Hfwords(Idmon, Pabsli),Hfwords(Mlint, Mlint),
                            Hfwords(Idmon, Pabsr), Hfwords(Mreal, Mreal),
                            Hfwords(Idmon, Pabslr),Hfwords(Mlreal, Mlreal),
                            Hfwords(Idmon, Pabsc), Hfwords(Mcompl, Mreal),
                            Hfwords(Idmon, Pabslc),Hfwords(Mlcompl, Mlreal),
                            Hfwords(Idmon, Pabsb), Hfwords(Mbool, Mint),
                            Hfwords(Idmon, Pabsbt),Hfwords(Mbits, Mint),
                            Hfwords(Idmon, Pabsch),Hfwords(Mchar, Mint),
                            0 ),
    Obband      = uplit(    Hfwords(Idgen, Pandb),  Mdbool, Mdbool,Mdbool,
                            Hfwords(Idgen, Pandbt), Mdbits, Mdbits,Mdbits,
                            0 ),
    Obarg       = uplit(    Hfwords(Idmon, Parg),  Hfwords(Mcompl, Mreal),
                            Hfwords(Idmon, Pargl), Hfwords(Mlcompl, Mlreal),
                            0 ),
    Obbin       = uplit(    Hfwords(Idmon, Pbin),  Hfwords(Mint, Mbits),
                            0 ),
    Obconj      = uplit(    Hfwords(Idmon, Pconj), Hfwords(Mcompl, Mcompl),
                            Hfwords(Idmon, Pconjl),Hfwords(Mlcompl, Mlcompl),
                            0 ),
    Obdiv       = uplit(    Hfwords(IdIIR, Pdiv),
                            Hfwords(IdRRR, Pdiv),
                            0 ),
    Obdoun      = uplit(    Hfwords(Idmon, Pdownsem),Hfwords(Msema,Mvoid),
                            0 ),
```

```
    Obdvab      = uplit(    Hfwords(IdRAR, Pdiv),
                            0 ),
    Obolem      = uplit(    Hfwords(Idgen, Polmbt), Mdbool, Mdbits,Mdint,
                            Hfwords(Idgen, Polmby), Mdchar, Mdbytes,Mdint,
                            0 ),
    Obonti      = uplit(    Hfwords(Idmon, Ponti), Hfwords(Mreal, Mint),
                            Hfwords(Idmon, Pontii),Hfwords(Mireal, Mint),
                            0 ),
    Obeq        = uplit(    Hfwords(IdAABNC, Peq),
                            Hfwords(IdSSB, Peqcs),
                            Hfwords(Idgen, Peqb),   Mdbool, Mdbool,Mdbool,
                            Hfwords(Idgen, Peqbt),  Mdbool, Mdbits,Mdbits,
                            Hfwords(Idgen, Peqby),  Mdbool, Mdbytes,Mdbytes,
                            0 ),
    Obgo        = uplit(    Hfwords(IdAABC, Pgo),
                            Hfwords(IdSSB, Pgocs),
                            Hfwords(Idgen, Pgobt),  Mdbool, Mdbits,Mdbits,
                            Hfwords(Idgen, Pgoby),  Mdbool, Mdbytes,Mdbytes,
                            0 ),
    Obgt        = uplit(    Hfwords(IdAABC, Pgt),
                            Hfwords(IdSSB, Pgtcs),
                            Hfwords(Idgen, Pgtby),  Mdbool, Mdbytes,Mdbytes,
                            0 ),
    Oble        = uplit(    Hfwords(IdAABC, Ple),
                            Hfwords(IdSSB, Plecs),
                            Hfwords(Idgen, Plebt),  Mdbool, Mdbits,Mdbits,
                            Hfwords(Idgen, Pleby),  Mdbool, Mdbytes,Mdbytes,
                            0 ),
    Obleng      = uplit(    Hfwords(Idmon, Plengi),Hfwords(Mint, Mint),
                            Hfwords(Idmon, Plengr),Hfwords(Mreal, Mreal),
                            Hfwords(Idmon, Plengc),Hfwords(Mcompl, Mcompl),
                            0 ),
    Oblevel     = uplit(    Hfwords(Idmon, Plevint),Hfwords(Mint, Msema),
                            Hfwords(Idmon, Plevsem),Hfwords(Msema, Mint),
                            0 ),
    Oblt        = uplit(    Hfwords(IdAABC, Plt),
                            Hfwords(IdSSB, Pltcs),
                            Hfwords(Idgen, Pltby),  Mdbool, Mdbytes,Mdbytes,
                            0 ),
    Obiwb       = uplit(    Hfwords(IdIBR, Piwbm),  Mdabsent,
                            Hfwords(IdIBR, Piwb),   Mdint,
                            Hfwords(Idmon, Piwbmstr),       Hfwords(Mstring,Mint),
                            0 ),
    Obmdab      = uplit(    Hfwords(IdIAI, Pmod),
                            0 ),
    Obmnab      = uplit(    Hfwords(IdRAR, Psub),
                            Hfwords(IdIAI, Psub),
                            0 ),
    Obminus     = uplit(    Hfwords(IdAAA, Psub),
                            Hfwords(Idmon, Pnegi), Hfwords(Mint, Mint),
                            Hfwords(Idmon, Pnegli),Hfwords(Mlint, Mlint),
                            Hfwords(Idmon, Pnegr), Hfwords(Mreal, Mreal),
                            Hfwords(Idmon, Pnegir),Hfwords(Mireal, Mireal),
                            Hfwords(Idmon, Pnegc), Hfwords(Mcompl, Mcompl),
                            Hfwords(Idmon, Pnegic),Hfwords(Micompl, Micompl),
                            0 ),
    Obmod       = uplit(    Hfwords(IdIII, Pmod),
```

```
                                0 ),
    Obno        = uplit(        Hfwords(IdRRRNC, Pno),
                                Hfwords(IdSSB, Pnocs),
                                Hfwords(Idgen, Pnob),   Mdbool, Mdbool,Mdbool,
                                Hfwords(Idgen, Pnobt),  Mdbool, Mdbits,Mdbits,
                                Hfwords(Idgen, Pnoby),  Mdbool, Mdbytes,Mdbytes,
                                0 ),
    Obnot       = uplit(        Hfwords(Idmon, Pnotb), Hfwords(Mbool, Mbool),
                                Hfwords(Idmon, Pnotbt),Hfwords(Mbits, Mbits),
                                0 ),
    Obodd       = uplit(        Hfwords(Idmon, Podd),  Hfwords(Mint, Mbool),
                                Hfwords(Idmon, Poddl), Hfwords(Mlint, Mbool),
                                0 ),
    Obor        = uplit(        Hfwords(Idgen, Porb),  Mdbool, Mdbool,Mdbool,
                                Hfwords(Idgen, Porbt),  Mdbits, Mdbits,Mdbits,
                                0 ),
    Obovab      = uplit(        Hfwords(IdIAI, Pover),
                                0 ),
    Obover      = uplit(        Hfwords(IdIII, Pover),
                                0 ),
    Obplab      = uplit(        Hfwords(IdRRR, Padd),
                                Hfwords(IdIAI, Padd),
                                Hfwords(IdSCS, Pcat+4), Mdstring,
                                Hfwords(IdSCS, Pcat+3), Mdchar,
                                0 ),
    Obplto      = uplit(        Hfwords(IdCSS, Pplustocs),
                                0 ),
    Obplitm     = uplit(        Hfwords(IdIIC, Plitm),
                                0 ),
    Obplus      = uplit(        Hfwords(IdRRR, Padd),
                                Hfwords(IdSSS, Pcat),
                                Hfwords(Idmon, Pnoop), Hfwords(Mint, Mint),
                                Hfwords(Idmon, Pnoop), Hfwords(Mlint, Mlint),
                                Hfwords(Idmon, Pnoop), Hfwords(Mreal, Mreal),
                                Hfwords(Idmon, Pnoop), Hfwords(Mlreal, Mlreal),
                                Hfwords(Idmon, Pnoop), Hfwords(Mcompl, Mcompl),
                                Hfwords(Idmon, Pnoop), Hfwords(Mlcompl, Mlcompl),
                                0 ),
    Obrepr      = uplit(        Hfwords(Idmon, Prepr), Hfwords(Mint, Mchar),
                                0 ),
    Obroun      = uplit(        Hfwords(Idmon, Proun), Hfwords(Mreal, Mint),
                                Hfwords(Idmon, Prounl),Hfwords(Mlreal, Mlint),
                                0 ),
    Obshl       = uplit(        Hfwords(Idgen, Pshl),   Mdbits, Mdint,Mdbits,
                                0 ),
    Obshr       = uplit(        Hfwords(Idgen, Pshr),   Mdbits, Mdint,Mdbits,
                                0 ),
    Obshrt      = uplit(        Hfwords(Idmon, Pshrti),Hfwords(Mlint, Mint),
                                Hfwords(Idmon, Pshrtr),Hfwords(Mlreal, Mreal),
                                Hfwords(Idmon, Pshrtc),Hfwords(Mlcompl, Mcompl),
                                0 ),
    Obsign      = uplit(        Hfwords(Idmon, Psgni), Hfwords(Mint, Mint),
                                Hfwords(Idmon, Psgnli),Hfwords(Mlint, Mint),
                                Hfwords(Idmon, Psgnr), Hfwords(Mreal, Mint),
                                Hfwords(Idmon, Psgnlr),Hfwords(Mlreal, Mint),
                                0 ),
    Obtimes     = uplit(        Hfwords(IdRRR, Pmul),
```

```
                                Hfwords(Idgen, Pmulic), Mdstring, Mdchar,Mdint,
                                Hfwords(Idgen, Pmulci), Mdstring, Mdint,Mdchar,
                                Hfwords(Idgen, Pmulis), Mdstring, Mdstring,Mdint,
                                Hfwords(Idgen, Pmulsi), Mdstring, Mdint,Mdstring,
                                0 ),
    Obtmab      = uplit(        Hfwords(IdRAR, Pmul),
                                Hfwords(IdIAI, Pmul),
                                Hfwords(IdSCS, Pmulsi), Mdint,
                                0 ),
    Obup        = uplit(        Hfwords(Idmon, Pupsem),Hfwords(Msema, Mvoid),
                                0 ),
    Obup1       = uplit(        Hfwords(IdAIA, Pexp),
                                0 ),
    Obup2       = uplit(        Hfwords(IdAIA, Pexp),
                                0 ),
    Obupb       = uplit(        Hfwords(IdIBR, Pupbm),  Mdabsent,
                                Hfwords(IdIBR, Pupb),   Mdint,
                                Hfwords(Idmon, Pupbmstr),       Hfwords(Mstring, Mint),
                                0 );


datapsect(Symtab)

 ! Environment enquiries
    data(       Defstid( 0, Mdint,   quoted Deftag( 6, 'MAXINT'))),
    data(       Defstid( 1, Mdreal,  quoted Deftag( 7, 'MAXREAL'))),
    data(       Defstid( 2, Mdreal,  quoted Deftag( 9, 'SMALLREAL'))),
    data(       Defstid( 3, Mdint,   quoted Deftag(10, 'MAXABSCHAR'))),

 ! Operations associated with BITS values
    data(       Defstid( 4, Prcbtrb, quoted Deftag( 8, 'BITSPACK'))),

 ! Operations associated with BYTES values
    data(       Defstid( 5, Prcbyst, quoted Deftag( 9, 'BYTESPACK'))),

 ! Standard mathematical constants and functions
    data(       Defstid( 6, Mdreal,  quoted Deftag( 2, 'PI'))),
    data(       Defstid( 7, Prcrr,   quoted Deftag( 4, 'SQRT'))),
    data(       Defstid( 8, Prcrr,   quoted Deftag( 3, 'EXP'))),
    data(       Defstid( 9, Prcrr,   quoted Deftag( 2, 'LN'))),
    data(       Defstid(10, Prcrr,   quoted Deftag( 3, 'COS'))),
    data(       Defstid(11, Prcrr,   quoted Deftag( 6, 'ARCCOS'))),
    data(       Defstid(12, Prcrr,   quoted Deftag( 3, 'SIN'))),
    data(       Defstid(13, Prcrr,   quoted Deftag( 6, 'ARCSIN'))),
    data(       Defstid(14, Prcrr,   quoted Deftag( 3, 'TAN'))),
    data(       Defstid(15, Prcrr,   quoted Deftag( 6, 'ARCTAN'))),
    data(       Defstid(16, Prcrfir, quoted Deftag(10, 'NEXTRANDOM'))),

 ! Channels
    data(       Defstid(17, Mdchan,  quoted Deftag(14, 'STANDINCHANNEL'))),
    data(       Defstid(18, Mdchan,  quoted Deftag(15, 'STANDOUTCHANNEL'))),
    data(       Defstid(19, Mdchan,  quoted Deftag(16, 'STANDBACKCHANNEL'))),

 ! Files and associated operations
    data(       Defstid(20, Prcchf,  quoted Deftag( 4, 'CHAN'))),
    data(       Defstid(21, Prcvfs,  quoted Deftag( 8, 'MAKETERM'))),
    data(       Defstid(22, Prcon,   quoted Deftag(16, 'ONLOGICALFILEEND'))),
```

```
    data(       Defstid(23, Prcon,   quoted Deftag(17, 'ONPHYSICALFILEEND'))),
    data(       Defstid(24, Prcon,   quoted Deftag( 9, 'ONPAGEEND'))),
    data(       Defstid(25, Prcon,   quoted Deftag( 9, 'ONLINEEND'))),
    data(       Defstid(26, Prcest,  quoted Deftag( 9, 'ESTABLISH'))),
    data(       Defstid(27, Prcifsc, quoted Deftag( 4, 'OPEN'))),
    data(       Defstid(28, Prcvrfrs,quoted Deftag( 9, 'ASSOCIATE'))),
    data(       Defstid(29, Prcrfv,  quoted Deftag( 5, 'CLOSE'))),
    data(       Defstid(30, Prcrfv,  quoted Deftag( 7, 'SCRATCH'))),
    data(       Defstid(31, Prcif,   quoted Deftag(10, 'CHARNUMBER'))),
    data(       Defstid(32, Prcif,   quoted Deftag(10, 'LINENUMBER'))),
    data(       Defstid(33, Prcif,   quoted Deftag(10, 'PAGENUMBER'))),
    data(       Defstid(34, Prcrfv,  quoted Deftag( 5, 'SPACE'))),
    data(       Defstid(35, Prcrfv,  quoted Deftag( 7, 'NEWLINE'))),
    data(       Defstid(36, Prcrfv,  quoted Deftag( 7, 'NEWPAGE'))),
    data(       Defstid(37, Prcvf3i, quoted Deftag( 3, 'SET'))),
    data(       Defstid(38, Prcrfv,  quoted Deftag( 5, 'RESET'))),

! Conversion routines
    data(       Defstid(39, Prcsn1i, quoted Deftag( 5, 'WHOLE'))),
    data(       Defstid(40, Prcsn2i, quoted Deftag( 5, 'FIXED'))),
    data(       Defstid(41, Prcsn3i, quoted Deftag( 5, 'FLOAT'))),

! Formatless transput
    data(       Defstid(42, Prcvfout,quoted Deftag( 3, 'PUT'))),
    data(       Defstid(43, Prcvfin, quoted Deftag( 3, 'GET'))),

! Binary transput
    data(       Defstid(44, Prcvfp,  quoted Deftag( 6, 'PUTBIN'))),
    data(       Defstid(45, Prcvfg,  quoted Deftag( 6, 'GETBIN'))),

! Particular preludo
    data(       Defstid(46, Refint,  quoted Deftag(10, 'LASTRANDOM'))),
    data(       Defstid(47, Prcreal, quoted Deftag( 6, 'RANDOM'))),
    data(       Defstid(48, Refile,  quoted Deftag( 7, 'STANDIN'))),
    data(       Defstid(49, Refile,  quoted Deftag( 8, 'STANDOUT'))),
    data(       Defstid(50, Refile,  quoted Deftag( 9, 'STANDBACK'))),
    data(       Defstid(51, Prcvout, quoted Deftag( 5, 'PRINT'))),
    data(       Defstid(51, Prcvout, quoted Deftag( 5, 'WRITE'))),
    data(       Defstid(52, Prcvin,  quoted Deftag( 4, 'READ'))),
    data(       Defstid(53, Prcvob,  quoted Deftag( 8, 'WRITEBIN'))),
    data(       Defstid(54, Prcvib,  quoted Deftag( 7, 'READBIN'))),

    data(       Defstid(55, Codp2ivv,quoted Deftag( 6, 'ONTICK'))),
    data(       Defstid(56, Codpiibbv,quoted Deftag( 7,'ONERROR'))),
    data(       Defstid(57, Codcibtvv,quoted Deftag(10,'ONSYSTRACE'))),
    data(       Defstid(58, Codcibtv,quoted Deftag( 8, 'SYSTRACE'))),
    data(       Defstid(59, Codiv,   quoted Deftag(12, 'WARNINGLEVEL'))),
%H>%
    data(       Defstid(60, Mdchan,  quoted Deftag(13, 'CONSINCHANNEL'))),
    data(       Defstid(61, Mdchan,  quoted Deftag(14, 'CONSOUTCHANNEL'))),
    data(       Defstid(62, Mdchan,  quoted Deftag(16, 'FIXEDPAGECHANNEL'))),
    data(       Defstid(63, Mdchan,  quoted Deftag(14, 'VARPAGECHANNEL'))),
    data(       Defstid(64, Codiv,   quoted Deftag( 7, 'SUTRACE'))),
    data(       Defstid(65, Codrftv, quoted Deftag(11, 'GETPROCTIME'))),
    data(       Defstid(66, Codv,    quoted Deftag(13, 'STARTPROCTIME'))),
    data(       Defstid(67, Codtmrl, quoted Deftag( 6, 'MUSECS'))),
    data(       Defstid(68, Mdchan,  quoted Deftag(14, 'SOSFILECHANNEL'))),
```

```
%<H%
%K>%%
    data(       Defstid(69, Prepssrts,quoted Deftag(11,'$APPENDCAPA')),
    data(       Defstid(70, Prespsab,quoted Deftag( 5, '$CALL')),
    data(       Defstid(71, Prepss,  quoted Deftag( 8, '$CLENGTH')),
    data(       Defstid(72, Prepssb, quoted Deftag( 8, '$COMPARE')),
    data(       Defstid(73, Presbv,  quoted Deftag( 8, '$CONTROL')),
    data(       Defstid(74, Prepsv,  quoted Deftag( 7, '$DELETE')),
    data(       Defstid(75, Prespsv, quoted Deftag( 8, '$GETCAPA')),
    data(       Defstid(76, Prepssrtv,quoted Deftag(12,'$INTERCHANGE')),
    data(       Defstid(77, Prestep, quoted Deftag(10, '$LNSLENGTH')),
    data(       Defstid(78, Prepsv,  quoted Deftag( 9, '$MAKEPAGE')),
    data(       Defstid(79, Prepsv,  quoted Deftag(14, '$MAKEUNIVERSAL')),
    data(       Defstid(80, Presspsv,quoted Deftag( 6, '$MERGE')),
    data(       Defstid(81, Prepsv,  quoted Deftag( 2, '$P')),
    data(       Defstid(82, Prepssrtv,quoted Deftag(5, '$PASS')),
    data(       Defstid(83, Prepssrts,quoted Deftag(11,'$PASSAPPEND')),
    data(       Defstid(84, Prepsb,  quoted Deftag(13, '$PCONDITIONAL')),
    data(       Defstid(85, Prepssrtv,quoted Deftag(8, '$PUTCAPA')),
    data(       Defstid(86, Prepsrtv,quoted Deftag(13, '$SETCHKRIGHTS')),
    data(       Defstid(87, Prepssv, quoted Deftag( 7, '$SWITCH')),
    data(       Defstid(88, Prespsv, quoted Deftag( 5, '$TAKE')),
    data(       Defstid(89, Prespsval,quoted Deftag(9, '$TYPECALL')),
    data(       Defstid(90, Prepsv,  quoted Deftag( 7, '$UPDATE')),
    data(       Defstid(91, Prepsv,  quoted Deftag( 2, '$V')),
    data(       Defstid(92, Prepsv,  quoted Deftag( 7, '$VACATE')),
    data(       Defstid(93, Prepsv,  quoted Deftag( 5, '$VALL')),
    data(       Defstid(94, Cods,    quoted Deftag( 8, '$GETSLOT')),
    data(       Defstid(95, Codist,  quoted Deftag( 9, '$MAKESTEP')),
    data(       Defstid(96, Codrim,  quoted Deftag( 9, '$MAKEWALK')),
    data(       Defstid(97, Codisl,  quoted Deftag( 8, '$USESLOT')),
    data(       Defstid(98, Codpiibbv,quoted Deftag(8, 'ONSIGNAL')),
    data(       Defstid(99, Mdslot,  quoted Deftag( 8, 'NULLSLOT')),
%%<K%

! Operators
    data(               Defstopri( 0, Obabs,    quoted Defopr( 3, 'ABS')),
    data(               Defstopri( 0, Obarg,    quoted Defopr( 3, 'ARG')),
    data(               Defstopri( 0, Obbin,    quoted Defopr( 3, 'BIN')),
    data(               Defstopri( 0, Obconj,   quoted Defopr( 4, 'CONJ')),
    data(               Defstopri( 0, Obdown,   quoted Defopr( 4, 'DOWN')),
    data(               Defstopri( 0, Obenti,   quoted Defopr( 6, 'ENTIER')),
    data(               Defstopri( 0, Obleng,   quoted Defopr( 4, 'LENG')),
    data(               Defstopri( 0, Oblevel,  quoted Defopr( 5, 'LEVEL')),
    data(               Defstopri( 0, Obnot,    quoted Defopr( 3, 'NOT')),
    data(               Defstopri( 0, Obodd,    quoted Defopr( 3, 'ODD')),
    data(               Defstopri( 0, Obrepr,   quoted Defopr( 4, 'REPR')),
    data(               Defstopri( 0, Obroun,   quoted Defopr( 5, 'ROUND')),
    data(               Defstopri( 0, Obshrt,   quoted Defopr( 7, 'SHORTEN')),
    data(               Defstopri( 0, Obsign,   quoted Defopr( 4, 'SIGN')),
    data(               Defstopri( 0, Obup,     quoted Defopr( 2, 'UP')),
    data(               Defstopri( 1, Obdvab,   quoted Defopr( 5, 'DIVAB')),
  gbinamo( Oprdvab,     Defstopr2( 1, Obdvab)),
    data(               Defstopri( 1, Obmnab,   quoted Defopr( 7, 'MINUSAB')),
  gbinamo( Oprmnab,     Defstopr2( 1, Obmnab)),
    data(               Defstopri( 1, Obmdab,   quoted Defopr( 5, 'MODAB')),
```

```
 gbiname( Oprmdab,      Defstopr2( 1, Obmdab)),
    data(               Defstopr1( 1, Obovab,   quoted Defopr( 6, 'OVERAB'))),
 gbiname( Oprovab,      Defstopr2( 1, Obovab)),
    data(               Defstopr1( 1, Obplab,   quoted Defopr( 6, 'PLUSAB'))),
 gbiname( Oprplab,      Defstopr2( 1, Obplab)),
    data(               Defstopr1( 1, Obplto,   quoted Defopr( 6, 'PLUSTO'))),
 gbiname( Oprplto,      Defstopr2( 1, Obplto)),
    data(               Defstopr1( 1, Obtmab,   quoted Defopr( 7, 'TIMESAB'))),
 gbiname( Oprtmab,      Defstopr2( 1, Obtmab)),
    data(               Defstopr1( 2, Obor,     quoted Defopr( 2, 'OR'))),
    data(               Defstopr1( 3, Oband,    quoted Defopr( 3, 'AND'))),
    data(               Defstopr1( 4, Obeq,     quoted Defopr( 2, 'EQ'))),
 gbiname( Opreq,        Defstopr2( 4, Obeq)),
    data(               Defstopr1( 4, Obne,     quoted Defopr( 2, 'NE'))),
 gbiname( Oprne,        Defstopr2( 4, Obne)),
    data(               Defstopr1( 5, Obge,     quoted Defopr( 2, 'GE'))),
 gbiname( Oprge,        Defstopr2( 5, Obge)),
    data(               Defstopr1( 5, Obgt,     quoted Defopr( 2, 'GT'))),
 gbiname( Oprgt,        Defstopr2( 5, Obgt)),
    data(               Defstopr1( 5, Oble,     quoted Defopr( 2, 'LE'))),
 gbiname( Oprle,        Defstopr2( 5, Oble)),
    data(               Defstopr1( 5, Oblt,     quoted Defopr( 2, 'LT'))),
 gbiname( Oprlt,        Defstopr2( 5, Oblt)),
 gbiname( Oprplus,      Defstopr2( 6, Obplus)),
 gbiname( Oprminus,     Defstopr2( 6, Obminus)),
    data(               Defstopr1( 7, Obelem,   quoted Defopr( 4, 'ELEM'))),
 gbiname( Oprtimes,     Defstopr2( 7, Obtimes)),
 gbiname( Oprdiv,       Defstopr2( 7, Obdiv)),
    data(               Defstopr1( 7, Obmod,    quoted Defopr( 3, 'MOD'))),
 gbiname( Oprmod,       Defstopr2( 7, Obmod)),
    data(               Defstopr1( 7, Obover,   quoted Defopr( 4, 'OVER'))),
 gbiname( Oprover,      Defstopr2( 7, Obover)),
 gbiname( Oprup1,       Defstopr2( 8, Obup1)),
 gbiname( Oprup2,       Defstopr2( 8, Obup2)),
    data(               Defstopr1( 8, Oblwb,    quoted Defopr( 3, 'LWB'))),
    data(               Defstopr1( 8, Obupb,    quoted Defopr( 3, 'UPB'))),
    data(               Defstopr1( 8, Obshl,    quoted Defopr( 3, 'SHL'))),
    data(               Defstopr1( 8, Obshr,    quoted Defopr( 3, 'SHR'))),
 gbiname( Oprplitm,     Defstopr2( 9, Obplitm)),

    data(       -1)
dataend;


! Think about doing IM and RE


undeclare Lexstop, Lexstart, Lextab;

globaldata(,Lexstart,)  Lexm(Lxvstart,empty));
globaldata(,Lexstop,)   Lexm(Lxvstop,empty));

globaldata(,Lextab,)

    0, Deftag( 0, empty),
    0, Deftag( 2, 'IM'),
    0, Deftag( 2, 'RE'),
```

```
    0, Deftag( 4, 'STOP'),
    0, Defdonint( 1),
%K>%%
    0, Deftag( 1, 'S'),
    0, Deftag( 1, 'W'),
    0, Deftag( 7, '$AUXRTS'),
    0, Deftag(11, '$GENERICRTS'),
    0, Deftag(12, '$AMPLIFYFLAG'),
    0, Deftag(13, '$TEMPLATEFLAG'),
%%<K%

    Defbold(Lxvat,       0,          2, 'AT'),
    Defbold(Lxvbegin,    0,          5, 'BEGIN'),
    Defbold(Lxvprimdr,   Mdbits,     4, 'BITS'),
    Defbold(Lxvothdr,    Mdbool,     4, 'BOOL'),
    Defbold(Lxvby,       0,          2, 'BY'),
    Defbold(Lxvprimdr,   Mdbytes,    5, 'BYTES'),
    Defbold(Lxvcase,     0,          4, 'CASE'),
    Defbold(Lxvothdr,    Mdchan,     7, 'CHANNEL'),
    Defbold(Lxvothdr,    Mdchar,     4, 'CHAR'),
    Defbold(Lxvprco,     "C",        2, 'CO'),
    Defbold(Lxvcode,     0,          4, 'CODE'),
    Defbold(Lxvcodop,    0,          6, 'CODEOP'),
    Defbold(Lxvprco,     "C",        7, 'COMMENT'),
    Defbold(Lxvprimdr,   Mdcompl,    5, 'COMPL'),
    Defbold(Lxvdo,       0,          2, 'DO'),
    Defbold(Lxvelif,     0,          4, 'ELIF'),
    Defbold(Lxvelse,     0,          4, 'ELSE'),
    Defbold(Lxvend,      0,          3, 'END'),
    Defbold(Lxvesac,     0,          4, 'ESAC'),
    Defbold(Lxvevent,    0,          5, 'EVENT'),
    Defbold(Lxvexit,     0,          4, 'EXIT'),
    Defbold(Lxvext,      0,          3, 'EXT'),
    Defbold(Lxvboolden,  A68false,   6, 'FALSE'),
    Defbold(Lxvfi,       0,          2, 'FI'),
    Defbold(Lxvothdr,    Mdfile,     4, 'FILE'),
    Defbold(Lxvfor,      0,          3, 'FOR'),
    Defbold(Lxvfrom,     0,          4, 'FROM'),
    Defbold(Lxvgo,       Mdjump,     2, 'GO'),
    Defbold(Lxvgoto,     Mdjump,     4, 'GOTO'),
    Defbold(Lxvheap,     Genheap,    4, 'HEAP'),
    Defbold(Lxvif,       0,          2, 'IF'),
    Defbold(Lxvin,       0,          2, 'IN'),
    Defbold(Lxvprimdr,   Mdint,      3, 'INT'),
    Defbold(Lxvidty,     Idtyis,     2, 'IS'),
    Defbold(Lxvidty,     Idtyisnt,   4, 'ISNT'),
    Defbold(Lxvloc,      Genlocref,  3, 'LOC'),
    Defbold(Lxvlong,     0,          4, 'LONG'),
    Defbold(Lxvmode,     0,          4, 'MODE'),
    Defbold(Lxvmodule,   0,          6, 'MODULE'),
    Defbold(Lxvnewmode,  Mdvnewpile, 7, 'NEWPILE'),
    Defbold(Lxvnewmode,  Mdvnewsimple,9,'NEWSIMPLE'),
    Defbold(Lxvnil,      Mdnil,      3, 'NIL'),
    Defbold(Lxvod,       0,          2, 'OD'),
    Defbold(Lxvof,       0,          2, 'OF'),
    Defbold(Lxvop,       0,          2, 'OP'),
    Defbold(Lxvouse,     0,          4, 'OUSE'),
```

```
    Defbold(Lxvout,      0,          3, 'OUT'),
    Defbold(Lxvpar,      0,          3, 'PAR'),
    Defbold(Lxvprco,     "P",        2, 'PR'),
    Defbold(Lxvprco,     "P",        7, 'PRAGMAT'),
    Defbold(Lxvprio,     0,          4, 'PRIO'),
    Defbold(Lxvproc,     0,          4, 'PROC'),
    Defbold(Lxvprimdr,   Mdreal,     4, 'REAL'),
    Defbold(Lxvref,      0,          3, 'REF'),
    Defbold(Lxvhoap,     Consecloc,  6, 'SECLOC'),
    Defbold(Lxvothdr,    Mdsema,     4, 'SEMA'),
    Defbold(Lxvshort,    0,          5, 'SHORT'),
    Defbold(Lxvskip,     Mdskip,     4, 'SKIP'),
    Defbold(Lxvothdr,    Mdstring,   6, 'STRING'),
    Defbold(Lxvstruct,   0,          6, 'STRUCT'),
    Defbold(Lxvthen,     0,          4, 'THEN'),
    Defbold(Lxvto,       0,          2, 'TO'),
    Defbold(Lxvboolden,  A68true,    4, 'TRUE'),
    Defbold(Lxvvoid,     Mdvoid,     4, 'VOID'),
    Defbold(Lxvwhile,    0,          5, 'WHILE'),

%H>%
    Defbold(Lxvothdr,    Mdtime,     8, 'PROCTIME'),
%<H%
%K>%%
    Defbold(Lxvothdr,    Mdpath,     4, 'PATH'),
    Defbold(Lxvothdr,    Mdrtsmask, 10, 'RIGHTSMASK'),
    Defbold(Lxvothdr,    Mdslot,     4, 'SLOT'),
    Defbold(Lxvothdr,    Mdstep,     4, 'STEP'),
    Defbold(Lxvothdr,    Mdwalk,     4, 'WALK'),
%%<K%

    Defpragitem(0,  3, 'RES'),
    Defpragitem(1,  5, 'POINT'),
    Defpragitem(2,  5, 'UPPER'),
    Defpragitem(3,  5, 'LOWER'),
    Defpragitem(4,  8, 'WARNINGS'),
    Defpragitem(5, 10, 'NOWARNINGS'),
    Defpragitem(6,  7, 'LISTING'),
    Defpragitem(7,  9, 'NOLISTING'),
    Defpragitem(8,  4, 'PAGE'),
    -1  );


end
eludom
```