Department of Applied Analysis
and Computer Science
Technical Report CSTR 1006
July, 1970

THE DIALATOR FILE SYSTEM
PROGRAMMER'S GUIDE

by

Doron J. Cohen, Paul M. Fawcett
Eric G. Manning & Larry Smith

Faculty of Mathematics

University of Waterloo
Waterloo, Ontario
Canada

Department of Applied Analysis
and Computer Science
Technical Report CSTR 1006
July, 1970

THE DIALATOR FILE SYSTEM
PROGRAMMER'S GUIDE

by

Doron J. Cohen, Paul M. Fawcett
Eric G. Manning & Larry Smith

Other manuals in this series are:

1. TRAIZE User's Guide
2. FAUST User's Guide
3. General Programmer's Guide
4. TRAIZE Programmer's Guide
5. FAUST Programmer's Guide
6. File System User's Guide

DIALATOR SYSTEM - PROGRAMMER'S GUIDE - FILE SYSTEM

This is a programmer's guide to the DIALATOR file system. The File System User's Guide should be read before this guide is studied.

The guide is laid out as follows:

SECTION 1. RECDIR and RECFILE

As explained in the FILE SYSTEM USER'S GUIDE, the file system is divided into two parts - the file directory and the record file.

The file directory is called RECDIR and is described under the name DIR in SYSLIB. It contains three structures, DIR_PRM, DKWT, and RECS.

DIR_PRM contains the directory parameters. AVA is a pointer to the head of a linked list of unused nodes. RG# indicates the number of regions used by the record file. ST# indicates the number of structures available. ND# is the number of nodes in the tree. DN# is the number of nodes desired in the tree. KW# is the number of keywords. At the moment this amounts to the structure names. POB is a variable used to point to the predecessor node of a given node when searching the tree.

DKWT contains a list of the keywords KEW and with each an associated number. In the case of the structure names it is the structure number.

RECS is a vector of structures, each structure containing a node of the tree. RNM is the node name. SIZ is the size of the record associated. If there is no record associated SIZ is zero. STB is the starting byte of the associated record. If there is no record STB is -1. SRG is the starting region number of the record, if none SRG is -1. TYP is the type of the structure in which the associated record is stored

if any, if none TYP is zero. SIS is a pointer to the sister node, and is set to -1 if there is no sister node. DAU is a pointer to the daughter node and is set to -1 if there is no daughter node.

The record file is called RECFILE and is described in SYSLIB under the name FILE. FILE describes the record file and the buffer used when reading or writing in the record file.

The buffer, BUF is 1800 words in length, or 7200 bytes. This corresponds to 1 region for the record file, also for the direct access device on which the file is stored. BUR is a buffer region pointer. D is the dimensions vector used in allocating the structures. ALC is not implemented. RECFILE is the actual record file, keyed according to regions of 7200 bytes.

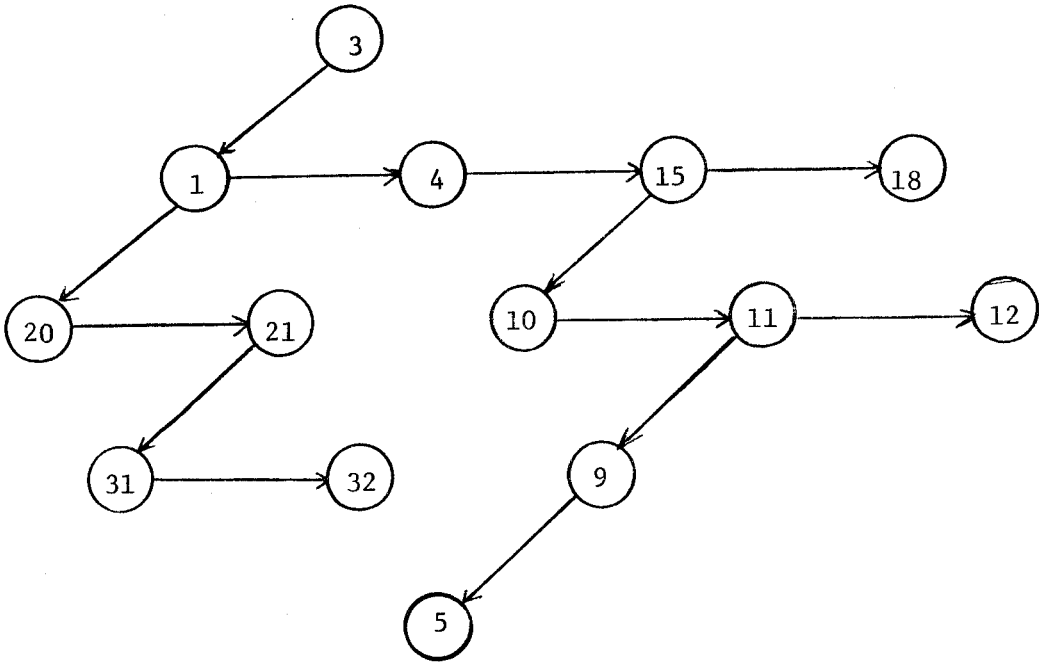
SECTION 2. Recursive PL/1 routines.

The purpose of this section is to give the programmer an idea of how a recursive procedure works.

Suppose we have a recursive procedure that is supposed to scan the tree. The skeleton of the program (without all the important validity checks) would be,

```
LOOK_AT:  PROCEDURE (P) RECURSIVE;
   DCL P FIXED BIN;
   IF DAU(P) > 0 THEN
             CALL LOOK_AT(DAU(P));
   IF SIS(P) > 0 THEN
             CALL LOOK_AT(SIS(P));
K=P; PUT SKIP EDIT('NODE',K,'HAS BEEN' 'LOOKED AT')(A,F(5),2A);
   RETURN;
   END LOOK_AT;
```

Suppose the tree is:



The following statement, CALL LOOK_AT(3), would generate this output.

```
node 32 has been looked at
node 31 has been looked at
node 21 has been looked at
node 20 has been looked at
node  5 has been looked at
node  9 has been looked at
node 12 has been looked at
node 11 has been looked at
node 10 has been looked at
node 18 has been looked at
node 15 has been looked at
node  4 has been looked at
node  1 has been looked at
node  3 has been looked at
```

In general the direction is from bottom to top and left to right.

SECTION 3. Subroutine ALOC

This subroutine allocates space for a record from the 'FREE' nodes of the tree. The 'FREE' nodes are scanned until one with a large enough number of bytes is found. The free node from which the space is taken is then reduced in size, in order to keep an accurate account of space available.

The subroutines called are FREER and SIZST.

Description of Variables:

$ is the node number of the node for which space is to be allocated for a record.

ISR is the new start region for the 'FREE' node from which space was taken.

IST is the new start byte for the 'FREE' node from which space was taken.

VS holds the total number of bytes for the offset of the new free node.

We check to see if $ is a valid node number. If so, FREER is called to place a free node next to the headfree node. This node will have the size of the old node (if there was one) at which we are making space for a record. This is convenient when we are writing over an old record of the same size or larger.

SIZST is called to determine the size of the record for which space is being allocated.

The node is checked for an illegal name and invalid size.

The list of FREE nodes is scanned to find storage space for the record. The start byte and start region of the FREE node with sufficient space assigned to the node $. The size of the FREE node is reduced. The total offset in bytes, to find the new start byte and start region for the FREE node is calculated.

If the new start byte is more than 1790, the FREE node is reduced in size to the nearest whole number of regions.

If the size of the FREE node is nil, it is returned to the unused node list and all of its parameters set to their defaults values.

The value returned is the number of regions rounded upward to the nearest integer.

SECTION 4. Subroutine CLEAN

This procedure cleans up core by freeing all the structures, designated by the non-zero bits in the parameter STS.

Description of Variables:

STS, TST are bit strings, length equal to the number of structures.

S is a bit vector defined on STS, to get at each bit.

LA is a label vector, to point to the action for each structure.

SLOOP is a do loop which examines each bit of S and if it is '1' frees the structure indicated (if it is allocated).

SECTION 5. Subroutine CNS

This subroutine constructs a new node in the directory tree.

The subroutine called is LUPTA.

Description of Variables:

$RNM is the node name of the node constructed.

$SIS is the sister link of the node constructed.

$DAU is the daughter link of the node constructed.

$ is the node number of the node constructed.

A check is made to see if there are any nodes in the unused node list. If so, the node pointed to by AVA is taken, the desired number of nodes incremented and the parameters of the new node filled in (except STB, STR and SIZ).

A check is made to see that the parameters are valid. The node number of the created node is returned.

SECTION 6. Subroutine DELETE

This subroutine deletes a subtree of the directory tree identifying the subtree by its qualified name.

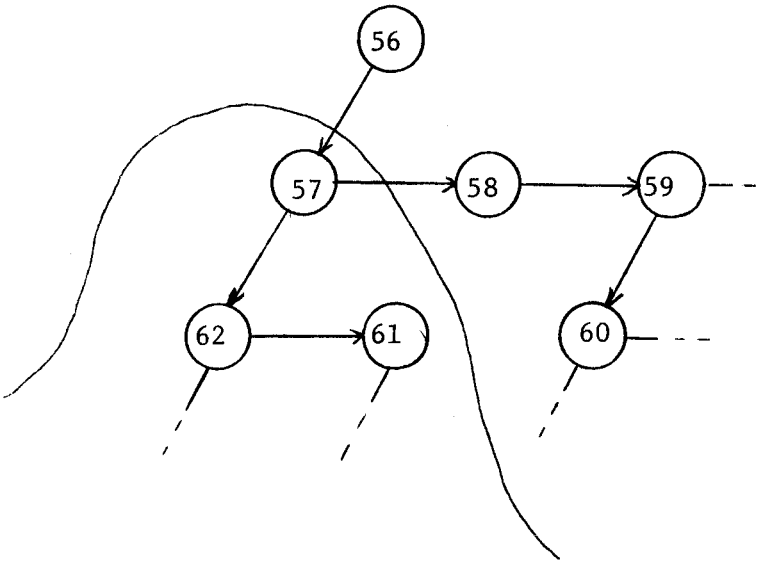
The called subroutines are LOCATE and IDLT.

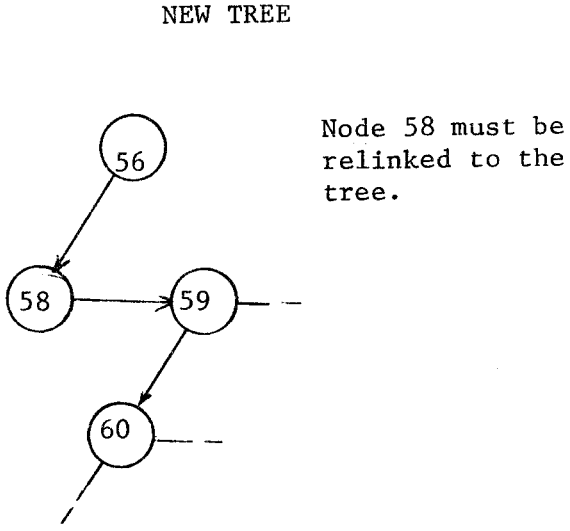
Description of Variables:

NDN, CC are qualified names indicating the subtree to be deleted.

P is the node number of the root of the subtree.

LOCATE is called to get the node number of the root of the subtree to be deleted. It also fills in POB which points to the 'mother' node of the root node. This is important, since if there are any other nodes on the same level as the root node, these nodes have to be relinked (perhaps to the mother node). For example:





The node number found by LOCATE is checked for validity. The predecessor or mother node of the root node is stored in I, then the daughter node of that node calculated. Again a validity check is done.

If the root node is the daughter node of the predecessor, the sister node of the root node becomes the new daughter node. If the root node is not, its level is scanned until it is found. It is deleted by linking its sisters together. Its sister link is set to 1. IDLT is called to delete the node and all the nodes in the subtree. The value returned is the return value of IDLT.

SECTION 7. Subroutine ENTER

This subroutine enters new nodes or a record into the directory tree.

The subroutine called is CNS.

Description of Variables:

NDN, CC are qualified names of the record to be entered.

D is the index of '.' in the qualified name.

B is the index of ' ' in the qualified name.

P is a pointer to the root of the subtree in which to enter the next node.

CC, D, B, POB, I and U are initialized. D, B and P are checked for validity. If the qualified name has at least one qualifier the first element is stored in NM. If the name is not qualified, NM receives all of the name. If there is an embedded blank, NM receives the string up to the blank (provided no period in string). If P is one, we have already added a node and we wish to add a daughter node.

If this is the last node to be added, D will be zero, so CNS is called with daughter and sister links set to -1. If it is not the last node to be added, CNS is called with the daughter link set to a call to IENTR, in order that the process may be repeated on the next part of the qualified name. The value returned by CNS is checked for validity, POB updated to point to the predecessor node if necessary. The node number of the created node is returned.

If P is not one, we have not entered any nodes and must first find the root node, which will have node name equal to NM. We do this by scanning the level pointed to by P.

If the node is not found, we must add an extra node to the level. This is achieved by calling CNS to create a new node between the second last and last node of the level. The sister link points to the old last node of the level and the sister link of the second last node is changed to point to the new node. As before, if D is zero this is the last node to be added, otherwise there are more to be added and the daughter link is set to a call on IENTR, to repeat the action with P set to 1. POB is updated and the node number of the node just created returned.

If the root node was found in the previous search and the qualified name has no subnames, we have a duplicate name. If D is not zero, the daughter node of the root node is looked for. If it does not exist, IENTR is called to create daughter nodes. POB is updated, and the node number of the node just created returned. If the daughter node does exist, IENTR is called to checked for duplicate nodes etc. and to append daughter nodes if necessary.

At any time a negative value for Q, less than -1 indicates failure mode. If failure occurs at any time POB is set to -1.

SECTION 8. Subroutine FCID

This subroutine fetches the circuit description into core for the specified circuit.

The subroutines called are LOCATE, REDR, SIZST, CLEAN, FDIR and FETCH.

Description of Variables:

CNM is the name of the circuit whose description is wanted.

OPT, OV are bit strings indicating where the description is to be sought.

OB is a bit vector defined on OV. If OB(1) is '1', core is to be cleaned by CLEAN before the circuit description is sought. If OB(2) is '1', the description is to be looked for in the file only. If OB(3) is '1' the description is to be looked for in core only.

FLOP is the faults option. It can have the following values, 'ALL', 'GET', 'NO', 'CORE', ' ', or, some selected faults structure name.

EV contains in the end a bit pattern of the structures which have been fetched according to their associated structure numbers. If a bit is on the structure with that number indicated by position was fetched into core.

EX is a bit vector defined on EV.

N# is the number of leads in the circuit.

O# is the number of outputs of the circuit.

I# is the number of inputs of this circuit.

F# is the number of feedback leads of the circuit.

Variables are initialized. The heading is printed followed by an interpretation of the options specified. The fault option is printed. A check is made to see if the options conflict.

FDIR is called if the file directory is not in core already. The tree is scanned by LOCATE to find the circuit name. If it is found, its node number is printed. LOCATE now looks for '$CIDS' in the subtree of the circuit name. If it is found, its name, node number, starting region and starting byte (rather word) are printed. If it is not found a message is printed.

The same is done for each of the other possible structures of the circuit.

If OB(1) is on, we must clean core of all structures. EX(4) is put on if $FMCS is not wanted.

If FLOP is CORE, the faults structure is expected in core. EX(4) is set, so $FMCS will not be brought into core from the file. Also, if $SFMC is allocated, information concerning it is printed.

CORE:

If OB(2) is off, we are looking for each of the structures in core first. Before information is printed, we demand that the structure be already allocated and the corresponding bit of EV be off. If the structure is in core, but does not belong to this circuit, it is freed.

FILE:

If OB(3) is off, we look for structures in the file. The search occurs only for those structures which have the corresponding bit in EV off. Of course, if the option is 'file only' all of the bits will be off, except, possibly the fourth (it does nothing). REDR is called to get the appropriate structure from the file.

If the fault option is none of the keywords 'NO', 'ALL', 'GET', 'CORE', or ' ', it is assumed to contain the name of a $SFMC structure. An attempt is made to fetch this structure under the given name.

If EF is on, we have previously indicated a desire to have a selected faults structure which was found in core, so we wish to free $FMCS if it has been brought into core.

If EF is off, EX(4) is set or reset depending on the allocation of $FMCS.

If EX(2) is not on, FCID has failed. The value returned is EV.

SECTION 9. Subroutine FDIR

This subroutine fetches the file directory into core.

Description of Variables:

RECDIR is a stream file, to get DIR_PRM, DKWT and RECS.

We read in DIR_PRM. DKWT is allocated and each of its members fetched. ND# is updated to contain the desired node's number, N holds the present number of nodes. RECS is allocated and fetched.

If the desired number of nodes is greater than the present number, nodes with names, '.UNUSED.', are linked into an independent list. The number of these nodes is the difference between the number of nodes present and the desired number. For each of these nodes, the start byte, start region and daughter node are set to -1, and type and size are set to zero.

The present list of unused nodes is scanned, and the last node is linked to the first node of the independent list.

If the unused list is empty, AVA is pointed to the first node of the independent list.

The sister link of the last node of the previously independent list is set to -1, to indicate the end of the available node list.

The file buffer and region pointer are initialized, as a safety precaution.

SECTION 10. Subroutine FETCH

This subroutine fetches a record from the record file.

The subroutines called are FDIR, LOCATE and REDR.

Description of Variables:

WDN, CC are qualified node names of the record to be fetched.

TND is final part of the qualified name.

P, Z is a pointer to a node in the tree.

X a the position of '.' in the qualified name.

If the file directory is not in core, FDIR fetches it.

LOCATE is called to locate the record to be fetched.

NXD:

If the qualified name has subnames, X will be greater than zero. The latter portion of the name is stored in CC. Go to NXD.

The above is repeated until the final portion of the qualified name lies in CC. If LOCATE returns a value less than one, a message is printed. Otherwise, REDR is called to read in the record from the record file.

The value returned is the value returned by REDR.

SECTION 11. Subroutine FLST.

This subroutine lists the selected faults for the specified circuit, if the structure $SFMC is in core.

Description of Variables:

REV is a vector which back links the rows of $SFMC to the lead numbers.

M# is the number of faults for the circuit.

N# is the number of leads in the circuit.

NF# is the number of faulty leads in the circuit.

JFL is the fault code number of a particular type of single fault.

IFNC is the absolute value of the lead function number.

This procedure fails if any of $LDSC, $SYMTA, $SFMC or $COPT are not allocated. N#, M#, NF# are initialized and, REV is allocated and set to zero.

A message header is printed giving information about the selected faults structure. The REV vector is filled using the pointer of $SFMC. The first machine, the fault free one, is printed. The subsequent faults are given numbers from 2 to M# using M.

There are two loops, MLOOP and ILOOP, since there may be multiple faults in which case M will not be incremented.

MLOOP:

K is set to zero; M printed.

ILOOP:

We scan each row of the selected faults table. If the number at the present location in the table is the current value of M, K is incremented. The lead number is taken from the REV vector. The fault type number is obtained from the $COPT fault definition table.

The value of K is printed, with the lead name, found using the BLINK in $SYMTA, and the fault definition. If the fault is an input fault, a check is made to see that it is not an invalid input fault. If it is valid it is printed.

We go back and check to see if M is the same as the fault number for the next fault.

Note the entire table is scanned for each fault number M. This is logical since a multiple fault may involve any subset of leads.

SECTION 12. Subroutine FREER

This subroutine frees space occupied by a record of the tree, and gives this space to a node in the 'FREE' node section.

The subroutine called is CNS.

Description of Variables:

$ is the node number of the node to be freed.

ALL is a bit indicating whether subsequent nodes are to be freed.

NM is the node name, '..FREE..'.

NM and M1 are initialized. The node number is checked for validity.

If ALL is on, we wish to free the space occupied by the daughter node and all of its sisters; plus all of their successors. (NOTE: in this system FREER has not been implemented with this option yet. This isuseful if it is necessary to free an entire subtree.)

The size of the node to be freed must be greater than zero. A check is made to see that size, start byte, and start region are valid.

If they are, a node is created between the 'headfree' node and its sister node (i.e. the new node becomes the first node in the 'free' list. It is assigned the size value had by the node to be freed, in the tree. It is also assigned the start byte and start region of the node to be freed.

The size of the node whose space was freed, is given size of zero, and its start byte and start region set to -1.

SECTION 13. Subroutine GETST

This subroutine gets a structure of an input file into core.

The called subroutines are LUPTA and SORTA.

Description of Variables:

is the number of the structure to be fetched.

INFT is the means by which the structure is to be read.

TND is the final portion of a qualified name.

NDN, CC is a qualified name of a structure.

LA is a label vector, indicating the action for each structure.

L is the line number.

The file is opened; if it has a name other than SYSIN it is given by INFT. The structure name is read from the file. The final portion of the qualified structure name is stored in TND. It must begin with '$'. The name in TND is looked for in DKWT. If # is zero, the user has left the structure number to be decided by the program. If # is not zero it is checked for validity against the number returned by LUPTA.

The dimensions vector is read from the file. These elements contain the parameters necessary to allocate the given structure.

Action now goes to the particular structure to be read. It is allocated, and the information read from the file.

It is important that the information appear in a certain format, since all GET instructions are edited.

SECTION 14. Subroutine IDLT

This subroutine deletes a node from the directory tree as well as all its successors. Disc space occupied by terminal nodes is returned to the free storage list.

Description of Variables:

$, R are the node numbers of the node to be deleted.

$ is checked for validity as a node number. The node name is checked for illegality.

The daughter link is stored in I. If I points to a node of the tree, IDLT is called for each node on that level.

If this node has disc space associated, a check is made of the validity of the type of node, the start byte and the start region.

The disc space is freed by linking the node, to the headfree node (node #1) and to the sister node of node #1. The appropriate parameters are adjusted and the value of R returned.

If there is no disc space associated, the node is returned to the unused node availability list by pointing AVA at this node and linking this node to the previous head node. The desired number of nodes is decreased by one and the value of R is returned.

SECTION 15. Subroutine LDIR.

This subroutine prints out the directory tree.

Description of variables:

P, S indicate the node which is to be the root node of the printed tree.

SIZE indicates the total number of bytes used.

UNUS indicates the number of unused nodes.

CNT indicates the number of nodes in the tree.

FCNT indicates the number of nodes in the free list.

The heading is printed, followed by certain parameters concerning number of nodes. This is followed by a listing of the keyword table for structures.

P is checked for validity. The tree heading and the illustrative node are printed out. LST is called to print the nodes of the tree. This is followed by a printout of more information concerning the tree: the number of nodes in the tree, the number of nodes in the free list, the total number of bytes used in records, and the number of unused nodes.

If there are any unlocatable bytes, the number is printed. If there are any unlocatable nodes the number is printed.

DUMP:

If an error has occurred, the information of RECDIR is printed out.

LST:

This routine prints out the tree from the node specified through all of its subset nodes. The variable S, here, is used to point to the column in which each node is to begin on printout.

The node number passed as a parameter is checked for validity. The node is printed. CNT is incremented, to show that another node has been printed out.

If the node is a free node, FCNT and FREE are updated. If the node has type other than zero, SIZE is updated.

If the node printed has a daughter node, S is increased and LST is called on the daughter node. S is decremented to put it back to the value it had previous to the LST call.

If the sister node exists LST is called on the sister node.

SECTION 16. Subroutine LOCATE

This procedure locates a record in the directory by its name.

Description of Variables:

NDN, CC are the record name

NM is part of the qualified name.

D is the position of '.' in the name.

B is the position of ' ' in the name.

POB is the node number of the predecessor or, the 'mother' node, for the record node.

P is a pointer to the root of the subtree in which to search for the node.

CC, D, B and POB are initialized. A validity check is made on D, B and P.

If D is greater than one, NM assumes the first part of the record name.

Otherwise, if B is greater than one, NM assumes the first part of the name up to the blank encountered, and, if B is less than one, NM assumes the entire name.

If NM is '*', the returned value is -9. The use of the '*' indicates that every node in the subtree of the qualified name is to be dealt with. In the case where the name is CIRCUIT1.$CIDS.*, the node number of $CIDS will be returned.

The node with name, NM, is searched for on the present level

of the tree. If it is not found, -2 is returned. If it is found and D is zero, we have located the record, so the node number is returned. If it is found and D is zero, we have located the record, so the node number is returned. If D is not zero, the name still has modifiers.

We travel the daughter link and if it points to a node of the tree, LOCATE is called again on the remainder of the record name and the new subtree. If the daughter link is less than one, we have failed, -2 is returned.

The POB variable is updated to point to the new predecessor, pointed to by I. The value of Q is returned, since if everything went well, we have called LOCATE again so Q will point to the node sought.

SECTION 17. Subroutines LOUP and LUPTA

These procedures implement the binary search algorithm on the sorted tables, in order to locate the member named.

Description of Variables:

TAB is the symbol table with two substructures, NAME, the symbol name and, NUMB, the code number associated with the name.

NM is the symbol to be looked for in the table.

L is the length of the table.

UB is the upper bound.

LB is the lower bound.

The basis of the method is to split the table in two each time by looking at the middle element, until the symbol required is found.

The value returned is the code number associated with the symbol name (if it is found). If not, zero is returned .

SECTION 18. Subroutine PCID

This subroutine prints the circuit description of the specified circuit.

The subroutine called is FLST.

Description of Variables:

CNM, NM are the circuit name.

FLOP is the faults option.

EV is a bit string, indicating with '1' bits the structure numbers which are allocated.

EX is a bit vector defined on EV.

EF is a bit, indicating a selected faults structure is allocated.

N# is the number of leads in the circuit.

O# is the number of outputs.

I# is the number of inputs.

F# is the number of feedbacks.

IFP is a selected faults pointer.

IOR is a pointer to successors list.

The EX vector is filled in according to the structures allocated at the time PCID was called. A message is printed, if either $LDSC or $CIDS is not allocated.

If FLOP is 'ALL', EF is set to '0' and a message is printed if $FMCS is not allocated.

If FLOP is 'GET' or is the name of some selected faults structure, EX(4) is set to zero, and a message is printed if EF is not '1'.

The circuit parameters are placed in N#, O#, I# and F# from $CIDS. A check is performed on all structures, to see that their parameters coincide with those of $CIDS. If any do not, a message is printed and R is set to -1.

A printout of the circuit description follows. The heading is printed. The number of leads, the number of outputs and the name of each, the number of inputs and the name of each, and the number of feedbacks and the name of each are printed.

The parameters of $CIDS are printed. If EF is '1', FLST is called to list the selected faults structure.

A description of the circuit, lead by lead, follows. For each lead is printed: its associated line number, its name, its function, its level, its input references, the number of output references, its output references and its associated fault numbers. Each portion of information is dependent, of course, on the allocation of certain structures.

LOOP:

The DO loop parameter I is printed to indicate the lead number. The corresponding symbol name is printed only if $SYMTA is allocated. The function names are printed if $LDSC is allocated. These function names are taken from $COPT if it is allocated otherwise from

$LDSC. If $LDSC is allocated the level number and the first three input references are printed and, if $SUCS is allocated the first four output references are printed preceded by the number of output references. K indicates the countdown on the output references. If $SUCS is not allocated, but $LDSC is, the output reference pointer is printed. If EF is '1', the first four elements of the row pointed to by the faults pointer are printed. If EX(4) is '1', the first four elements of the appropriate row are printed. If $REFS is allocated and the lead is an input or feedback, a message is printed.

All of the above information appears on a single line. On the next line, if $LDSC is allocated the last two possible input references are printed. If there are still some successors to be printed K will be greater than zero. As many as four will be printed starting in column 52. Again, if there have been fault numbers printed before, the last four elements of the appropriate row are printed now. If $REFS is allocated and the lead is an output or terminal, a message is printed.

If there are still output references to be printed, K will be greater than zero. The remainder is printed four per line, until K is zero.

The value returned is R.

SECTION 19. Subroutine PUTST

This routine prints out the contents of the structure specified.

The subroutine called is SIZST.

Description of Variables:

is the structure number.

LA is a label vector indicating the action dependent on the structure.

A check is made concerning the validity of the structure number. SIZST is called to calculate the size of the structure to be printed.

The action is diverted to the specific structure, in order to print headings and its contents. If the structure is not allocated a message is printed.

SECTION 20. Subroutine REDR.

This subroutine reads a record from the record file into core.

The subroutine called is MOVE which moves a fixed amount of bytes (1 region) from the buffer into the structure.

Description of Variables:

$ is the node number of the node in the directory tree indicating the record.

ALL indicates all of the successor nodes in the tree, which point to records, should be read.

$T indicates the structure number.

OF indicates the offset in bytes of the record.

L indicates the length in bytes of the portion of the record moved.

Z indicates the offset of the buffer, this is filled by MOVE.

R is the value returned.

I is the start region of the record.

J is the start byte of the record.

A validity check is performed on the node number R, $T, J, Z, I, L, and OF are initialized. If $T is zero there is no record attached, go to SOF.

S, I, and J receive their values from the nodes. These values a checked for validity. Since J is in words (4 bytes) it is multiplied by four and subtracted from 7200 to get the proper number of bytes to be moved. If L is greater than the size of the record left to be moved, L is modified.

If the buffer region is different from the starting region of the record, an initial read is done. The dimension vector D is filled from the first eight words of the buffer (this is to allow allocation).

The structure is allocated and the first part of the record moved by MOVE. Go to TROF.

TROF:

The size of the structure yet to be read is decreased by L, the amount just moved.

If there is still some record to be moved, the offset is updated, the region number is updated and L is set to 7200.

If L is greater than the size remaining, L is adjusted. The offset and the region numbers are checked. The next portion from RECFILE is read into the buffer, the buffer region is updated. The subroutine MOVE is called to move the buffer contents into the appropriate structure. This is repeated until the structure has been moved.

SOF:

If ALL is '1', REDR is called on the subset tree with the present node as root node.

SECTION 21. Subroutine SDIR.

This routine stores the directory tree back into RECDIR.

The RECDIR file is opened for output. L is initialized to 1000. This gives a line number count so that any member can be referenced. The directory parameters are saved. The keyword table is saved, L being updated before each save statement.

The actual tree information is saved, L starts at 9000 the first time, and is incremented by 1000. Thereafter L is incremented by 10,000.

SECTION 22. Subroutine SIZST

This subroutine calculates the size of the structure in core whose number is specified.

Description of Variables:

N#8, I#8, O#8, CV#8, CP8, M#8, are variables which hold a number which represent a number of whole bytes. This is used for bit strings.

LA is a label vector pointing to the appropriate action for a given structure number.

A check is made to see if the structure number is valid. For each type of structure the size is calculated if the structure is of a variable size. For those structures which have bit strings the CEIL function is used to calculate the size to the nearest whole byte.

The value returned is the size of the structure in bytes.

SECTION 23. Subroutines SORT and SORTA

These routines sort the symbol table specified alphabetically, in order that the binary search can be used. The "reverse bubble" routine is employed.

Characteristics of the sort routine:

Each time the list is scanned the scanner makes one less comparison. For example, after the first scan, the largest element is at the bottom of the list for sure. Therefore, no entry need be compared to it.

The scan is accomplished by comparing the size of successive elements. A switch is made if they are not in the proper order.

The check for completion, is achieved by a bit variable which is set to '0' before each scan. It is changed to '1' if any switches are made. Thus the bit variable will be '0' at the end of a scan if the sort is complete.

SECTION 24. Subroutine STORE

This subroutine stores a record in the record file.

The subroutines called are FDIR, WRTR, LOCATE and IENTR.

Description of Variables:

TND is the final part of the qualified name.

NDN, CC are the qualified name.

DISP is the disposition of the storing. It can be 'OLD', 'NEW', 'YES', 'NO' or ' '. If ' ' is found, 'NEW' is assumed.

P is a pointer to the node where the record is to be stored.

If the record directory is not allocated, FDIR is called.

If DISP is NO, one is returned.

CC, Z and P are initialized, P, by a call to LOCATE. The INDEX function is used to get the last part of the qualified name TND. If DISP is 'OLD' and P is negative the record was not found so a message is printed and a negative value returned.

NOT_OLD:

If DISP is 'new' or ' ', then if P is greater than zero a message is printed since the record already exists. If DISP is not 'YES', 'NEW' or ' ', a message is printed and a negative number returned. If DISP is 'YES', P less than zero, and TND is '*', then a message is printed and only one record stored. * IENTR is called to enter a new node in the tree. POB now has the predecessor of the node in which the record is to be stored.

WRTR is called, on the predecessor node.

---

* Since P was negative the node with the qualified name was not found and we cannot store the whole subtree as the '*' asks for.

SECTION 25. Subroutine WRTR

This subroutine writes a record specified by the node number, into RECFILE.

The subroutines called are MOVE and ALOC.

Description of Variables:

$ is the node number of the node in the directory indicating the record.

ALL indicates all of the successor nodes in the tree which point to records should be rewritten.

$T indicates the structure number.

OF indicates the offset in bytes of the structure.

L indicates the length in bytes of the portion of the record to be moved.

Z indicates the offset of the buffer.

R is the value returned.

I is the starting region of the record.

J is the starting byte of the record.

A validity check is made on the node number received. R, $T, J, Z, I, L and OF are initialized. IF $T is zero there is no record attached. Go to SOF.

The node number is checked to see if it is a legal node. ALOC is called to allocate space in RECFILE. S, J and OF are filled in.

This doloop writes the structures into RECFILE. L is set to the size of the portion of the record to be written. The RECFILE of the

current region is read into the buffer area to be adjusted. MOVE is called to adjust the buffer area by moving into the buffer a portion of the structure. A write statement follows, putting the buffer area back into RECFILE. This is repeated until the entire record is written.

SOF:

If ALL is '1', all the subset nodes with records attached must be rewritten.