Department of Applied Analysis
and Computer Science
Technical Report CSTR 1005
July, 1970

THE DIALATOR SYSTEM
FAUST 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 1005
July, 1970

THE DIALATOR SYSTEM
FAUST PROGRAMMER'S GUIDE

by

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

We wish to thank the Defence Research Board of Canada, the Northern Electric Research & Development Laboratories, Ottawa, and the Faculty of Mathematics of the University of Waterloo for financial, technical and moral support which made this system of programs possible.

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. File system User's Guide
6. File system Programmer's Guide

PROGRAMMER'S GUIDE - DIALATOR SYSTEM - FAUST *

This is a programmer's guide to program FAUST of the DIALATOR system. This guide should not be studied before the FAUST User's Guide has been read.

The guide is laid out as follows:

SECTION 1. Program FAUST

SECTION 2. Subroutine ALLF

SECTION 3. Subroutine DIMUY

SECTION 4. Subroutine FLEV2

SECTION 5. Subroutine INIFL

SECTION 6. Subroutines RSTS, SET & SETS

SECTION 7. Subroutine SORTRE

SECTION 8. Subroutine TREE

---

* The General Programmer's Guide should be read before this manual is studied.

SECTION 1. FAUST

Those external subroutines called by FAUST which are not explained in this guide, are described in the FILE or TRAIZE Programmer's Guides.

Description of variables:

DFLT is the default value for the faults to be simulated, if a RUN statement is executed. Its value is taken from the control card. The user may override this value by specifying in the RUN statement, 'FAULTS=NO', 'FAULTS=ALL', or 'FAULTS=recname', where recname is the name of some selected faults structure.

NSU is the next syntactic unit, or word, filled by SCAN.

P is the delimiter pointer returned by SCAN.

N# is the number of leads in the circuit.

M# is the number of faulty machines considered for the circuit.

O# is the number of outputs of the circuit.

I# is the number of inputs of the circuit.

F# is the number of feedbacks of the circuit.

N#8 is the number of leads rounded to the nearest byte.

CPUINTVL is the CPU time elapsed since the last CPUTIME call.

CPUTOTAL is the CPU time elapsed since the first CPUTIME call.

FL is the first level of the tree to be printed.

HS is the root node of the tree on the first level. If HS is zero the entire first level is printed; otherwise a subtree with the above rootnode is printed.

LL is the last level of the tree to be printed.

MAXIT is the maximum number of iterations allowed in DIMUY, in trying to stabilize a machine with a feedback loop.

The CPUTIME subroutine is initialized by calling it with CPUINTVL set to -260400000.

A control card is sought and read by CCINT.

CPUTIME calls are placed throughout FAUST to give the user an indication of how much CPU time is required for a certain portion of the simulation.

The circuit description is fetched by FCID. This routine first looks in core and then in the file for the circuit.

The circuit catalogue is fetched to give single fault type definitions and to give a faults list for the valid lead functions.

If FLOP is 'get', FLEV2 is called to interpret the selected faults definition statements. The selected faults structure is stored in the file if DISP is other than 'NO'.

If FLOP is 'all' and $FMCS has not been allocated then ALLF is called to calculate the single faults. The structure is then stored in the file.

If LIST is 'yes', the circuit description is printed by PCID.

The circuit parameters are filled from $CIDS except for M#. This is filled according to the FLOP value.

$LVBS is allocated after D is filled from N# and M#. The LVBS table and activity vector are initialized to zeroes.

The FAUST operator table is fetched into core to make available the valid FAUST instruction words.

N# is rounded to the nearest byte. Since there are five buffer rows used with the LVBS table, five is added to N#.

M#L allows a maximum of 80 elements of a row in the LVBS table to be printed on a single line.

Description of variables of the innerblock:

LLIST is a bit vector, with length equal to the number of leads rounded to the nearest byte. A '1' bit in position I indicates the Ith lead is to be processed, otherwise this lead is bypassed.

TLIST is a bit vector, with length equal to the number of leads rounded to the nearest byte. A '1' bit in position I indicates the Ith lead is being traced.

ACT is a label vector which designates the actions to be taken according to the FAUST instruction word picked up by SCAN.

OP holds NSU and is an argument of LUPTA.

LN holds NSU and is an argument of LOUP.

BOPR is a bit string indicating a boolean operation for the library function BOOL (See PL/1 reference manual).

RECNAME is a temporary name for any of the structures.

BIT indicates the ACTIVE and UNACTIVE option with a '1' and a '0' respectively.

TRC indicates the trace routine is active and inactive by a '1' and a '0' respectively.

STANO is the number of the FAUST statement being processed.

UB is the upper bound of the diagnostic tree.

We initialize TLIST,LLIST and BOPR to zeroes. UB has a default value of

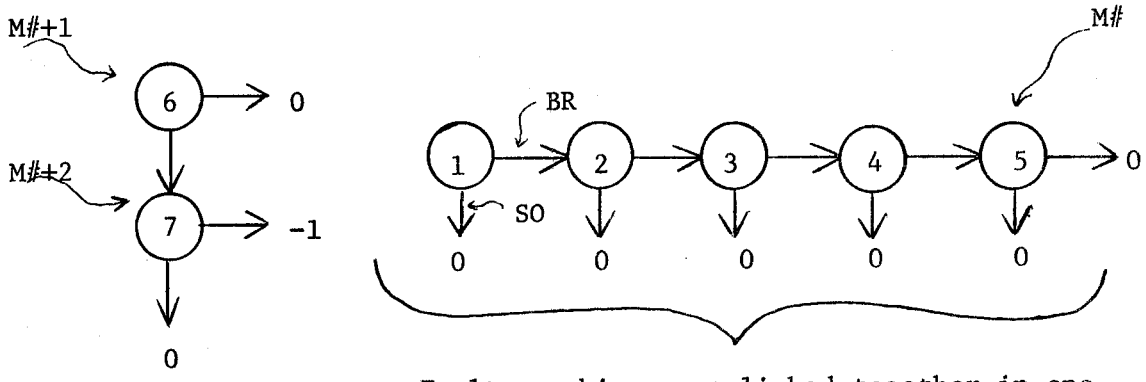
3 x number of faulty machines.

TRC is '0' to indicate that the trace routine is inactive by default.

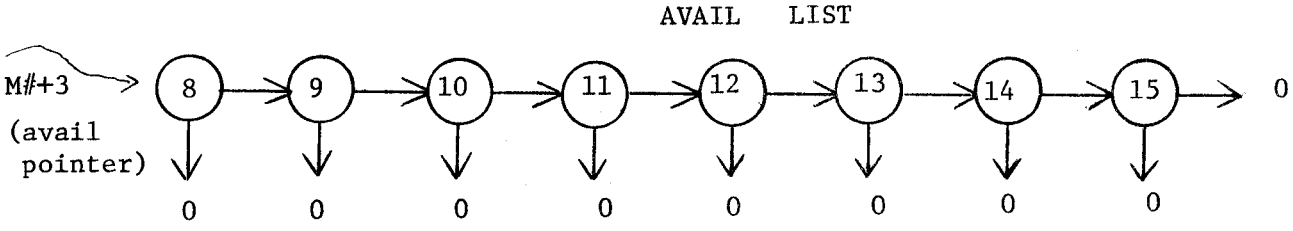
INITRE:

$DIATR is allocated after D is initialized. The parameters of $DIATR are filled in. The output vector is initialized to zero. The avail pointer is set to M#+3. All nodes are linked together by brother links. This chain is split by setting BR(M#), BR(M#+1), and BR(UB) to zero. SO(M#+1) points to M#+2 and BR(M#+2) is set to -1.

A diagram of the tree setup at this stage looks as follows for a circuit with 5 faults and UB = 15.



Faulty machines are linked together in one subset.



The above picture may well be called the 'unborn tree' because there is actually no tree present. The tree is always generated between nodes M#+1 and M#+2. Notice that the absolute value of BR(M#+2) points to the machine list, but that BR(M#+2) points nowhere. This will be of more significance when the manufacture of the tree is examined in describing subroutine SORTRE. The avail list contains unused nodes, these nodes are taken from left to right, as needed.

GET_NEXT:

The statement counter is incremented and a new statement is scanned. The first NSU brought by SCAN must be a FAUST instruction word. LUPTA checks to see if NSU is in FAUST.$OPTAB and returns a code number if it is found, a zero if it is not found. The transfer statement takes the program to the ACT with subscript equal to the code number returned.

ACT(0):

This act indicates an invalid instruction word was found.

ACT(1): SET

This act sets the specified bits in the LVBS table to '1'. If a left parenthesis is encountered the user has specified that bits are to be set only for the machine numbers specified. $MLIST is called to get these and to modify LVBS(0) accordingly. (LVBS(0) has '1' bits for machines specified, '0' bits elsewhere). A closing parenthesis must appear. If there are no machine numbers specified, all machines are set

by default. SETS is called to set the entire LVBS(0) to '1'. BIT is set to one, by default, meaning the activity bits will be set at the appropriate time.

Subroutine $LLIST is called to get the leads specified.

The keyword 'ACTIVE' or 'UNACTIVE' may appear. If 'UNACTIVE' appears the activity bits should not be altered later, so BIT is set to zero. A semicolon must appear.

The action of setting the required bits is done by BOOL, a PL/1 library routine. It unites the rows, corresponding to the leads specified, of the LVBS table with the machine mask in LVBS(0) under the boolean operator BOPR='0111'. In effect, this function takes each row with corresponding bit in LLIST equal to '1', and sets bit I of that row if bit I in LVBS(0) is '1', otherwise the bit is left unaltered. If BIT is '1' the activity bits of the successors of the leads set are set.

ACT(2): RESET

This act resets the specified bits in the LVBS table to '0'. The format is the same as for ACT(1). This time the boolean operator is '0100' to accomplish the proper action.

ACT(3): FLIP

This act flips the specified bits in the LVBS table (1 → 0 and 0 → 1). The format is the same is the same as for ACT(1). This time the boolean operator is '0110', to accomplish the proper action.

ACT(4): TRACE

This act alters the trace pattern by adding the leads, specified, to the trace list. TRC is set to '1'. This reactivates the trace routine if it has been deactivated. If there is no lead list the trace pattern previously in use is employed. Otherwise $LLIST is called to get the leads specified. A semicolon should appear; if so, the function BOOL is called with parameter '0111'.

ACT(5): UNTRACE

This act alters the trace pattern by removing the leads specified, from the trace list. The format is the same as for ACT(4) except that TRC is set to '0'. This deactivates the trace routine if it was active.

ACT(6): SHOW

This act shows the rows of the LVBS for the specified leads. A lead list must appear. $LLIST is called to get the lead list. SHOW is called to show the LVBS rows.

ACT(7): SAVE

This act saves $LVBS and $DIATR. By default the recname includes the runname. If the user specifies a recname, he must do so in the form 'RECNAME=.....'. In that case the recname is formed from the user-supplied name. Finally, STORE is called to the two structures.

ACT(8): BACKUP

This act brings a new $LVBS and $DIATR from the file into core. The format is similar to ACT(7). The present $LVBS and $DIATR structures are freed before the new entries are fetched.

ACT(9): RUN

This act simulates the circuit by updating the LVBS table. The fault option is first set to the default value from the control card and the maximum amount of iterations set to 8 (default). If a semicolon is not found, then either one of the two keywords, FAULTS and MAXIT, or both must appear. Each must be followed by an equal sign and if both appear they must appear in the above order. DIMUY is called to update the LVBS table. If TRC is '1' then TLIST is placed in LLIST and SHOW called to show the leads specified.

ACT(10): SORTRE

This act adds a new level to the diagnostic tree and sorts the machines according to output. Note, if the outputs have already been sorted once, they are sorted within their subsets. The semicolon must be present after the first call to SCAN. SORTRE is called to perform the sort.

ACT(11):

This act prints out the diagnostic tree. There are three parameters which the user can specify. These are first set to default values. If a semicolon is not found the keywords FIRST, ROOT, LAST are looked for in that order. Each should be followed by an equal sign after which SCAN is called again to pick up the user-supplied parameter and adjust the appropriate default parameter. TREE is called to print the tree.

ACT(16): INITREE

This act initializes or reinitializes the diagnostic tree. UB is set at three times the number of faulty machines. If a semicolon

does not appear look for 'UB=' and pick up the user-supplied upper bound. Check to see that a semicolon follows and that UB is large enough. Go to INITRE.

ACT(17): INIFAULT

This act sets the stuck at faults. A semicolon must appear. INIFL is called to set the stuck at faults.

ACT(18): ACTIVATE

This act adds the specified leads to the activity vector. BIT is set to '1' in order that the activity vector will be employed. The leads to be activated are obtained by calling $LLIST. LLIST is then used to change the activity vector, @, accordingly.

ACT(19): DEACTIVATE

This act removes the specified leads from the activity vector. The format is the same as ACT(18) except that BIT is set to '0'.

ACT(12), (13), and (14):

These acts are not implemented yet.

ACT(15): END

This act signifies that the FAUST statements have terminated. LDIR is called to list the file directory. If it is in good condition it is saved by SDIR.

FLUPA:FLAD:FLUSH:

This group of statements flushes invalid statements. FLUPA handles invalid operands. FLAD handles invalid delimiters. FLUSH flushes the statement to the next semicolon.

$LLIST:

This procedure accepts a list of leads and sets the corresponding bits of LLIST to '1'. LLIST is initialized to '0'. As long as commas are found LLOOP is executed. The keyword 'ALL' is checked for and if found all of LLIST is set to '1'. Otherwise the keywords 'INPUTS', 'OUTPUTS' and 'FBKS' are looked for. If any of these are found, $REFS is used to obtain the lead numbers and the appropriate bits of LLIST are set. If the symbol '#' is found, the subsequent numbers are converted to fixed binary and the bit with that index in LLIST is set to '1'. If the NSU operand is none of the above, LUPTA is called on $SYMTA and if the operand is a valid lead name the appropriate bit is set to '1'. If the operand is blank a message is printed.

$MLIST:

This subroutine translates a list of faulty machines and sets the corresponding bits in LVBS(0) to '1'. LVBS(0) is reset to '0'. As long as there are commas found, the loop is re-entered. If an invalid machine number is found, a message is printed. If there is a blank delimiter, the keyword 'TO' is expected and all the bits are set, for the range specified. The MIN function is used to insure that the range does not exceed the upper limit. For example, if M# is 13 and the user specifies 9 to 19, the bits of LVBS(0) from 9 to 13 would be set.

SHOW:

This subroutine shows the LVBS rows specified in the LLIST vector. The first put statement prints the tens digits of faulty machine numbers. The next put statement prints the heading and the units digits of the faulty machine numbers. The next two do-loops print rows of the LVBS table on the condition that the LLIST bit for the given row is '1'. For the non-buffer rows the lead name for the row is printed.

FAIL:

This message is printed if FAUST fails, and, the program is terminated. In most cases a failure message, indicating the source of the error, is printed before this error message.

SECTION 2. Subroutine ALLF

This subroutine creates the single faults structure $FMCS. ALLF accepts the circuit name as a parameter and returns the number of faulty machines.

Description of Variables:

M# is the number of single faults in the circuit.

N# is the number of leads in the circuit.

IFNC is a temporary place for the lead function number.

NOI is temporary storage for the number of input references of a given lead.

IF, a bit, is one if a fault is to be entered into table.

JFL is temporary storage for the fault type number from $COPT.FLT.

The structure $FMCS is allocated after D is initialized from $LDSC.N#. All the elements of the fault table are set to zero.

The numbers from 1 to M#, starting at one, are placed in the FM table, as faults are found in the circuit. Of course, every lead will have the two basic faults, diode stuck at zero and diode stuck at one. If the diode has a feedback line associated, it will have the third basic fault, feedback diode open. The rest of the faults, input faults, are dependent on the number of input references at each lead.

For each lead the function number is placed in IFNC. The number of input references is calculated and placed in NOI. NOI is checked with $COPT.#IN; if NOI is too large a message is printed.

The appropriate row of $COPT.FLT is scanned from left to right. IF is '1' as long as the basic faults are being dealt with i.e. JFL ≠ 3, and, is '1' while JFL is 3 and the input references have not ceased. M# is initialized to zero, and each time an entry is made to the table, M# is incremented and entered. The description of the fault is then printed. If it is an input fault, its input reference is also printed.

Suppose the Ith row of the fault table is:

25 26 27 28 29 0 0 0

and the Ith lead of the circuit has function 'OR'. By referring to the 14th row of the circuit functions catalogue, the faults of the circuits with numbers 25, 26, 27, 28, and 29 can be typed as, output stuck at zero output stuck at one, and the last three are open input diodes. Each of these faults belong to the Ith lead because of the row in which they appear.

SECTION 3. Subroutine DIMUY.

This subroutine simulates a circuit by updating the LVBS table.

The circuit fault option and the maximum number of feedback iterations allowed are passed as parameters.

There is a set of called assembler routines which manipulate bit strings. They are: SET, which sets a bit in a bit string (first parameter). The second parameter is the location in the bit string to be set.

RST resets a bit in a bit string (first parameter). The second parameter is the location.

PIK picks a single bit from location K (second parameter) in a bit string (first parameter).

SETS sets all bits in a bit string (parameter).

RSTS resets all bits in a bit string (parameter).

ANAS ands two bit strings after setting a bit at location K (third parameter) in the source string. The source string remains unaltered. First parameter is the target string; second parameter is the source string.

ORAS is the same as ANAS but the boolean function is 'OR'.

COAS copies one bit string (second parameter) into another bit string (first parameter) and sets a single bit at position K (third parameter) in the first parameter string.

STAR sets a bit at position K (second parameter) and resets a bit at position L (third parameter) in a bit string (first parameter).

ANST ands two bit strings and places the result in the first. The second parameter (source string) remains unaltered. The target string is the first parameter.

ORST is the same as ANST but the boolean operation is 'OR'.

EXST is the same as ANST but the boolean operation is 'exclusive OR'.

Description of Variables:

FM, FM1 are vectors which hold one row of $FMCS or $SFMC whichever the case may be.

F is a temporary place for the lead function number of the lead being processed.

MAXIT is the maximum number of feedback iterations allowed in attempting to stabalize a machine.

FIRE holds the single input reference lead number for the feedback node being processed.

ALLF, NOFL are bits which tell whether the faults option is 'all' or 'no'.

NF, WF are label vectors which indicate the action for each function under conditions of no faults and with faults respectively.

For each of the feedback nodes the fifth input reference is used as a counter. This is an economy move! The fifth input reference for feedback diodes has no other use. The fifth input references are set to zero.

SIM:

The leads that have activity bits '1' are processed.

AT:

The lead function number is placed in F. The activity bit for the lead being processed is set to '0'. The lead number is printed to give the user a trace of the leads 'propagated'. If the circuit being simulated has no fault, or if the selected faults do not involve the lead being processed then a branch is taken to NF(F).

If the circuit has all single faults or the lead has selected faults the appropriate row from the fault table is placed in FM and a branch taken to WF(F).

NF(0):WF(0):

An invalid function has been found, at least as far as FAUST is concerned. A message is printed.

NF(1):WF(1): INP

For inputs there are no boolean functions to be consulted, so a transfer is taken to PROP.

NF(2):WF(2): FBK

Before each iteration a check is performed to see if the LVBS row of the solitary input reference of the feedback node is identical to LVBS row of the feedback node itself. If it is, there is no unstability and no further iterations need be done. If it is not, the LVBS row of the input reference replaces LVBS(0). The iteration counter is incremented

and checked against the maximum iterations allowed. If the iterations number does not exceed the upperbound then ∅ is found by looking at the first element in $SUCS past the output reference pointer of the feedback node (Look at $SUCS to see how it is setup.) One is subtracted from the lead number obtained. This is done because, before simulation occurs again, ∅ will be incremented by one. Go to FERN.

If the iterations number exceeds the bound, a message is printed to indicate that there are unstable machines. The LVBS row of the input reference is united to the LVBS row of the feedback node by the EXST function. This picks out the unstable machines as '1's. These machines are added to the list of unstable machines which are kept in LVBS(-4), by the ORST function. (SORTRE can distinguish unstable machines from stable machines by treating LVBS(-4) as part of the output string. The list of unstable machines is printed, using the PIK function to find them in LVBS(-4).

NF(3): AND2

The LVBS rows of the two input references for the diode are 'AND'ed. The result appears in LVBS(0). Go to PROCS.

WF(3): AND2

The COAS function is used to set the bit location corresponding to an open input diode from the first input reference. ANAS is called to set the bit location corresponding to an open input diode from the second input reference, and then to AND the two LVBS's of the input references. The result appears in LVBS(0). Go to FAULTIN.

NF(4):WF(4): OR2

These two labels do the same action as NF(3) and WF(3) except, the ORST and ORAS functions replace the ANST and ANAS functions.

NF(5): XOR2

This is the same as NF(3) except, EXST replaces ANST.

WF(5): XOR2

This is the same as WF(3) except, both input diodes must be set before the EXST function is called.

NF(6):WF(6): NAND2.

These two labels perform the same action as NF(3) and WF(3) except the LVBS result is two's complemented.

NF(7): NOT

The LVBS of the input reference is two's complemented and stored in LVBS(0). Go to PROCS.

WF(7): NOT

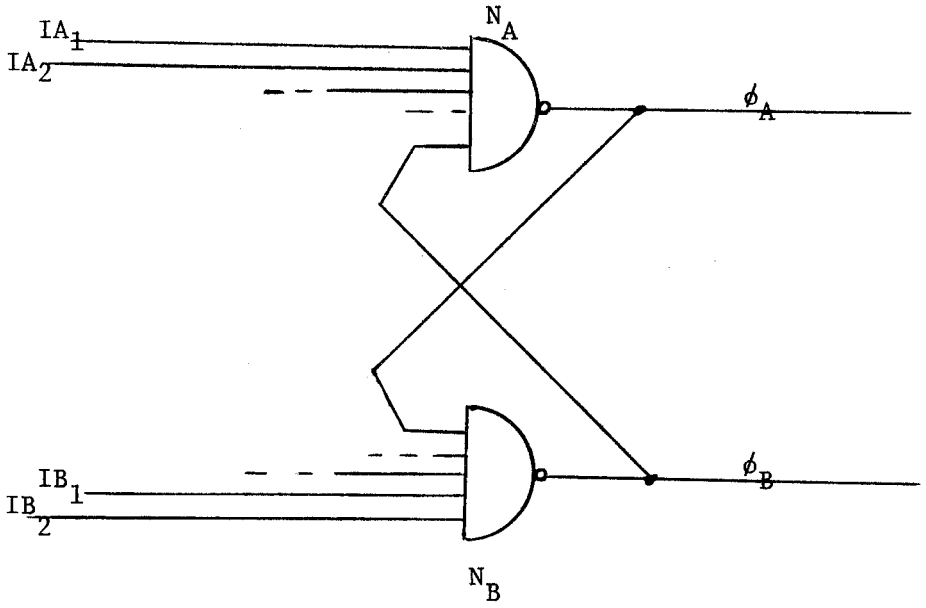
The bit location corresponding to an open input diode is set in the input reference's LVBS row and stored in LVBS(0). LVBS(0) is complemented. Go to FAULTIN.

NF(8):WF(8): NAND

These two labels are done the same as NF(6) and WF(6) except, there can be more than two input references. So the action must be repeated for each of the additional input references.

NF(9): FF1

This function simulation is of a NAND flip-flop, which consists of two NAND gates, each with an output wire acting as an input wire for the other. Pictorially it would look as follows:



Therefore $\phi_A = (IA_1 . IA_2 . \phi_B)'$

$= (IA_1 . IA_2)' + \phi_B'$

$= (IA_1 . IA_2)' + IB_1 . IB_2 . N_A$

The way the flip-flop is simulated is not as one would expect. Instead of simulating the FF1 and FF0 separately they are simulated at the same time. This is not so bad because the FF0 part would be simulated immediately after FF1 anyway. To do this two buffers had to be added. In

the end LVBS(-1) holds the results for FF0 and LVBS(0) holds the results for FF1.

Each of the input references are 'AND'ed for the two NAND gates. The results are stored in LVBS(-2) and LVBS(-3). The above equation is used to get the final result in LVBS(0) and LVBS(-1). Go to PROCS.

WF(9): FF1

The COAS function is called to set the open input diode bit. The ANAS function is called to set the open input diode bit for each of the other input references and to AND the strings. The fault vector for FF0 is placed in FM1 and, similar action is taken for it as was for FF1. The results are placed in LVBS(0) and LVBS(-1). The STAR function is called to set the open feedback diode fault. When this is done the feedback line eminating from the node in question is reset. RST is called to reset the machine corresponding to FF0 stuck at one in LVBS(0). The same is done for LVBS(-1). The equation, as above, is calculated and the results are placed in LVBS(0) and LVBS(-1). SET is called to set the machine corresponding to FF0 stuck at zero in LVBS(0). This is the same as saying, if FF0 is stuck at zero then FF1 will be stuck at one. This is why the action is performed after the equation has been calculated. The same is true for FF0, when FF1 is stuck at zero. Go to FAULTIN

NF(10):WF(10): FF0

Since FF0 has already been simulated, LVBS(-1) is stored in LVBS(0). Go to PROCS or FAULTIN respectively.

NF(11):WF(11): WOR

Place the first input references LVBS row in LVBS(0) and AND the remainder of the input reference's LVBS rows with LVBS(0).

NF(12):WF(12) NOR

These two labels do the same action as NF(8) and WF(8) except LVBS(0) is not two's complemented after the AND operations.

NF(14):WF(14): OR

These two labels perform similar to NF(13) and WF(13) except, ANST and ANAS are replaced with ORST and ORAS.

FAULTIN:

Set the stuck at faults for the particular lead.

PROCS:

We store LVBS(0) into the LVBS row of the lead that has just been processed.

PROP:

The activity bits of the successors, of the lead just processed, are set.

NOCH:

The lead counter Ø is incremented and checked against N#. If the function of the next lead is negative (e.g. FFO), the lead is part of a MACRO, so transfer to AT. Otherwise transfer to SIM.

SECTION 4. Subroutine FLEV2

This subroutine translates the selected faults definition statements of the user and builds $SFMC.

Description of Variables:

FRNM is the circuit name passed as a parameter.

CNAME is the circuit name.

TAG is a bit which indicates a multiple fault has been defined.

IF is a bit, which signifies a fault has been selected.

LNAME is a variable to contain a lead name.

NSU is the next syntactic unit filled by SCAN.

OEOF, OEOD are labels containing the failure and end of data labels in use when FLEV2 is called.

EOF, EOD are labels containing the failure and end of data labels in use for FLEV2.

M# is the number of faulty machines.

N# is the number of leads of the circuit.

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

IFNC contains a lead function number.

SFLT is a bit vector, which indicates with '1' bits which of the faults have been selected for a certain lead.

The structure $FMCS is allocated after $LDSC.N# fills D. The need for this structure may not be obvious. It is used temporarily to hold the selected faults. In the end they are transferred to $SFMC. This avoids messing with the pointers of $SFMC throughout the program.

$FMCS is initialized, the present EOF and EOD labels are saved, and temporary labels substituted. DELIS, SOUP, QUP, QUT and NC are initialized for the SCAN subroutine.

GET_NEXT:

TAG and SFLT are set to '0' to indicate, no multiple faults and, no faults selected for present lead yet.

NEXT:

SCAN brings an operand delimiter pair. Operand 'END' is checked for. No operand or invalid delimiters (blank, colon, slash are valid) are checked for. The symbol '+' is looked for, and if found, the delimiter must be a blank. Tag is set to indicate a multiple fault. Go to NEXT.

If there is no '+' sign, NSU must be a lead name, so LOUP is called to get the lead line number. If it is not valid go to FLASH. The function code number is stored in IFNC. IF is set to '0'. If a semicolon delimiter is present at this time, all faults are assumed and SFLT is set to '1'.

BASIC_FAULTS:

The basic faults are those of the type: stuck at one, stuck at zero, and open feedback diode.

If the present delimiter is a slash, go to INPUT_FAULTS. If the present delimiter is a semicolon, go to MILL. SCAN is called to get the next pair. Invalid delimiters are checked for, (comma, slash, semicolon) are valid. If NSU is 'ALL', all the basic faults are selected, so all bits of SFLT are set to '1' except those which correspond to the

type 3 faults in the FLT row.

Any other basic faults will be redundant and they are printed. If NSU is not 'ALL', set IF to '1'. If NSU is 's @ 0' then K = 1. If NSU is s @ 1 then K = 2. If NSU = 'OFD' then K = 4. Finally the Kth bit location in SFLT is set. A message is printed if K is -1. Go to BASIC_FAULTS.

INPUT_FAULTS:

SCAN is called to get the next pair. IF is set to '0'. If NSU is blank and the delimiter is a semicolon go to MILL. If NSU is blank and the delimiter otherwise, a message is printed. Go to INPUT_FAULTS. If NSU is 'ALL, SFLT is set to '1' in columns which correspond to columns in FLT containing '3'. If IF is '1' we have obtained some previous input faults so a redundancy occurs. Any other faults specified before a semicolon are ignored. Go to MILL. If NSU is not 'ALL', IF is set to '1'. L points to the first position of the input fault in FLT. The input reference name is looked up to get its lead number. If the lead number is invalid, go to FINE. Otherwise, a check is made to see if the lead is indeed an input reference and if so which of the input references it is. This last step is carried out because, if the input reference is the third one it must be the third one in the fault list entered since the structure $FMCS is being used. JK gives the number of input references. To place the right fault the columns of FLT with 32 are scanned from left to right decrementing JK by 1 each time a move to right is done. When JK is less than one the position has been

found. Go to FINE. Otherwise, a message is printed. Go to FLASH.

FINE:

Set the appropriate bit of SFLT. A semicolon is looked for. If none appears and if a comma is found go to INPUT_FAULTS.

MILL:

This section fills in the appropriate row of $FMCS. IF is set to '1'. JFL contains the fault type for the particular fault being processed. If JFL is 3, the fault is an input fault. Check to see if there are any input references remaining. If the corresponding bit of SFLT is '1', this fault has been selected. If 'IF' is '1', and if there is no entry in the specific position of $FMCS, insert the number M#+1. If there is a number, a lead has been specified twice. The second specification is ignored. Also if tag is '1', M# is decremented, so the fault number will be the same as for the previous fault, Go to GET_NEXT.

FLAD:

A Message is printed for invalid delimiters.

FLASH:

The source is deleted to the next semicolon, because of an error.

FEND:

The number of faulty leads is set to zero. Each row of $FMCS is scanned for a nonzero entry. If one is found NF# is increased. Since NF# is defined, $SFMC is allocated.

COPY:

The selected faults are transferred from $FMCS to $SFMC. Set K to zero. Again a nonzero entry is sought in each row of $FMCS.FM. If one is found, the pointer K is incremented and stored in $SFMC. FP(I), where the subscript I is the lead number, also receives the new value of K. The row of $FMCS is transferred to the row in $SFMC pointed to by $SFMC.FP(I).

END_COPY:

The old labels replace EOF and EOD. The number of faults is returned.

Here is a trace for the following statements concerning CIRCUIT1 of the TRAIZE User's Guide.

```
          A1  s @ 1, s @ 0;
      +   N1  s @ 1, s @ 0/X2; END;
```

LABEL	ACTION
GET_NEXT:	TAG='0'; SFLT='0';
NEXT	SCAN → NSU='A1', P = 1 ∿ Blank
	LNAME='A1', LOUP returns 3, I = 3
	IFNC = 13; IF = '0';
BASIC_FAULTS	SCAN → NSU='s @ 1', P = 3 ∿ comma
	K=-1; IF='1'; K=2;
	SFLT(2)='1';
BASIC_FAULTS	SCAN → NSU = 's @ 0', P = 6 ∿ semicolon
	K = -1; IF='1'; K=1;
	SFLT(1)='1';

```
LABEL           ACTION
MILL            L=0; J=1; IF='1'; JFL=1;
                $FMCS.FM(3,1),M#=1;
                J=2; IF='1'; JFL=2;
                $FMCS.FM(3,2),M#=2;
                J=3; IF='1'; JFL=3; L=1;
                IF='1';
                J=4; IF='1'; JFL=3; L=2;
                IF='1';
                J=5; IF='1'; JFL=3;
                J=6; IF='1'; JFL=3;
                J=7; IF='1'; JFL=3;
GET_NEXT        TAG='0'; SFLT='0';
NEXT            SCAN -> NSU ='+', P=1  ----blank
                TAG='1';
NEXT            SCAN -> NSU ='N1', P=1 ----blank
                LNAME='N1'; LOUP returns 4;
                I=4; IFNC=8; IF='0';
BASIC_FAULTS    SCAN -> NSU ='S@1', P=3 ---comma
                K=-1; IF='1'; k=2;
                SFLT(2)='1';
BASIC_FAULTS    SCAN -> NSU ='S@0', P=4 ---slash
                K=-1; IF='1'; K=1;
                SFLT(1)='1';
INPUT_FAULTS    SCAN -> NSU ='X2', P=6  ---semicolon
                IF ='0';  IF='1';  LNAME='X2';
                LOUP returns 2; K=2;
```

LABEL	ACTION
	JK=2;
	LK=3; JK=1;
	LK=4; JK=0;
FINE	SFLT(4)='1';
MILL	L=0; J=1; IF='1'; JFL=1;
	TAG='0'; M#=0;
	$FMCS.FM(4,1), M#=2;
	J=2; IF='1'; JFL=2;
	$FMCS.FM(4,2),M#=3;
	J=3; IF='1'; JFL=3;
	L=1; IF='1';
	J=4; IF='1'; JFL=3; L=2; IF='1';
	$FMCS.FM(4,4),M#=4;
	J=5; IF='1'; JFL=3; L=3; IF='0';
	J=6; IF='1'; JFL=3; L=4; IF='0';
	J=7; IF='1'; JFL=3; L=5; IF='0';
GET_NEXT	TAG='0'; SFLT='0';
NEXT	SCAN -> NSU ='END',P=6 -----semicolon
FEND	NF#=0; I=1; J=1;
	At end of outer doloop NF#=2;
	Allocate $SFMC
COPY	After $SFMC is: FP(3)=1,FP(4)=2,(all others are zero)

	1	2	3	4	5	6	7	8
1	2	1	0	0	0	0	0	0
2	3	2	0	4	0	0	0	0

SECTION 5. Subroutine INIFL

This subroutine initializes all stuck at faults for the given circuit.

The parameter passed is the fault option.

Description of Variables:

FLOP is the fault option.

ALLF is a bit, indicating FLOP is 'all'.

FM is a vector, holding one row of $FMCS.FM or $SFMC.FM.

For each lead the faults are taken from the appropriate structure and STAR is called to set and reset the bits for the stuck at faults. If $SFMC is used the pointer FP(I) will be zero unless there are some selected faults for the Ith lead.

SECTION 6. RSTS, SET and SETS.

These are assembler routines which manipulate bit strings.

RSTS and SETS reset and set entire bit strings to '0' and '1' respectively. The argument is the bit string.

SET sets a bit at position K in a bit string. The first parameter is the bit string; the second is the location to be set.

SECTION 7. Subroutine SORTRE

This subroutine adds a new level to the diagnostic tree and sorts the machines according to output.

The called subroutine is PIK, an assembler routine which picks a bit from a bit string (first parameter) at a given location (second parameter).

Description of Variables:

P is the node number of the present node being processed.

Q, R are the node number of the nodes last created.

Description of CNSTRE:

This subroutine constructs a node using the top element of the availability list if the list is non-empty. The values for the son and brother links are passed as parameters, $SO and $BR. These values are placed in the son and brother link locations of the node. The node number of the new node is returned.

Description of BSRTRE:

This subroutine sorts the machines appended to a single node into two groups according to there being a '1' or '0' bit in the Kth position of the output string. The parameters are the node number and the position K in the output string to be sorted. Note, if K is 0 then machines will be sorted according to LVBS(-4). Although this is not part of the output vector, LVBS(-4) distinguishes stable machines, '0''s, from unstable machines '1''s.

If $Ø is not 0 the lead numbers of the corresponding $REFS.OL ($Ø) is calculated. The son of the node $I is pointed to by I, and the bit belonging to this machine is picked out from the LVBS row of the output lead above. The appropriate bit in the output vector for this picked bit. The brother machines of I are scanned until the brother link is less than one. If the bit picked out for any of these machines differs, a new node is constructed and the same machine is removed from the other node's list and appended as a son node to the newly created node. Now J links together the old set of machines under the old node and Q links together the new set of machines under the new node. A negative brother link indicates the end of a chain. After the machines have been sorted the last brother node of the old list, BR(J), is made negative.

The routine calls itself to sort on the next bit of the output string. Also, if a new node has been created, it must also be sorted on the next bit.

Suppose the tree has just been initialized as shown in FAUST description of this manual (See SECTION D, INITRE). Assume that there are five leads in the circuit; leads 4 and 5 are outputs.

An initial sort is done on the LVBS table, before any runs, etc,

	1	2	3	4	5
-4	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

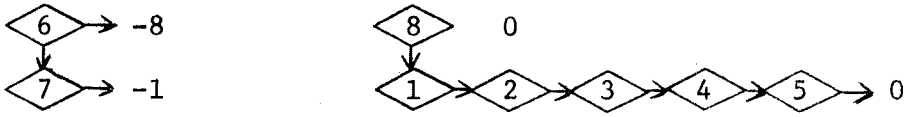
Trace of SØRTRE

ACTIØN:

P = 6; SØ(6) = 7 = M# + 2;

BR(P) = 0; call CNSTRE (1,0); BR(6) = -8;

Picture:



ACTIØN:

Call BSRTRE (8,0)

ØN = -4; I = 1; J = 1;

u = '0'; Q,R = 0; I = 2;

I = 3;

I = 4;

I = 5;

I = 0;

Call BSRTRE (8,1)

Same as above except ØN = 4

Call BSRTRE (8,2)

Same as above except ØN = 5

Return (0);

Return (0);

Return (0);

Now the tree consists of a single node with all the machines grouped under it in one subset.

Suppose SØRTRE is called again and the LVBS table looks as follows:

	1	2	3	4	5
-4	0	0	0	0	0
1	0	1	1	1	0
2	0	1	0	1	0
3	0	0	1	0	0
4	0	1	1	1	0
5	0	1	0	0	1

The trace would go as follows:

ACTIØN:

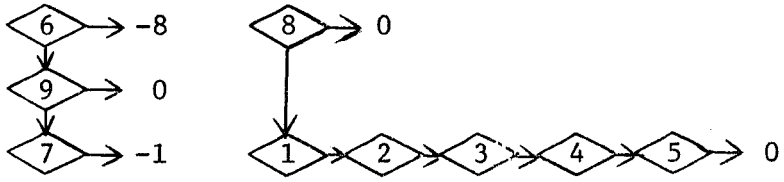
P = 6; SØ(P) = 7 = M#+2;

BR(P) = -8;

Call CNSTRE(SØ(6),0)

Q, SØ(6) = 9

PICTURE;



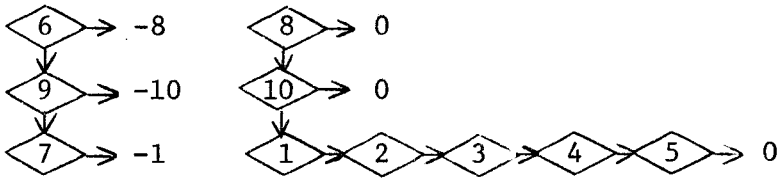
ACTIØN:

P = 8;

Call CNSTRE(SØ(8),0); R,SØ(8) = 10;

BR(9) = -10

PICTURE:



Call BSRTRE(10,0)

Since LVBS(-4) is all zeroes nothing happens and the picture remains the same.

Call BSRTRE(10,1)

ON=4; I,J=1; U='0'; Q,R=0; I=2; PIK(LVBS(4),2)='1';

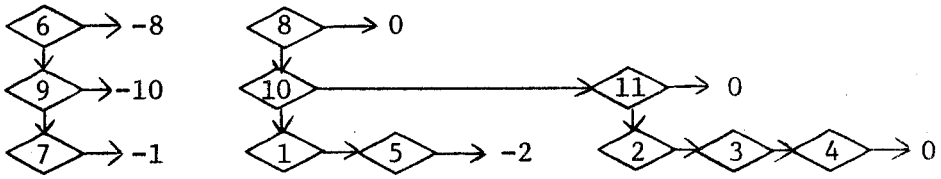
R,BR(10)=CNSTRE(2,0); Q=2; OU(11)='000';OU(11)='010';

BR(1)=3; I=3; PIK(LVBS(4),3)='1'; Q,BR(2)=3; BR(1)=4;

I=4; PIK(LVBS(4),4)='1'; Q,BR(3)=4; BR(1)=5;

I=5; PIK(LVBS(4),5)='0'; J=5; I=0; BR(5)=-2; BR(4)=0;

PICTURE:

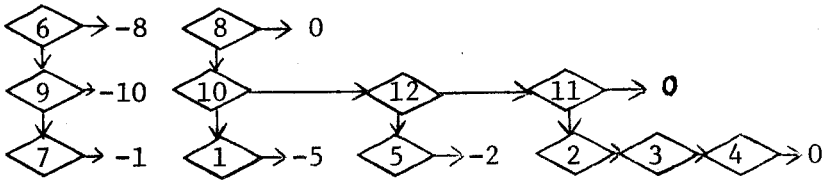


ACTION:

Call BSRTRE(8,2)

This call performs the last sorting on node number 8 since the last bit has been reached. The action is similar to the above.

PICTURE:



ACTION:

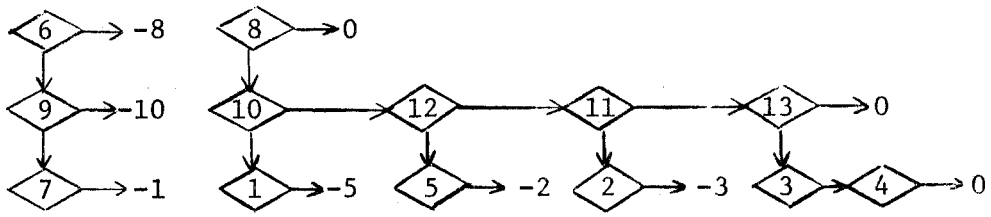
RETURN(12)

Call BSRTRE(11,2)

Call BSRTRE(11,2)

This time the sort is being done on the node created during the second call of BSRTRE. Action is similar.

PICTURE:



RETURN(13)

RETURN(13)

RETURN(0)

Each of the negative brother links does not link to any node, unless the absolute value of the link is taken. The machines have been split into four subsets. They are (1), (5), (2) and (3,4). The nodes along the left side are not part of the tree but the top and bottom, M#+1 and M#+2, are standard pointers to the first level of the tree and the set of machines respectively.

In the major subroutine (SORTRE) the left hand column of pointers is scanned to find the last level of the tree. If there are no nodes in the tree, an initial node is constructed and BSRTRE is called.

If there are nodes already present and the diagnosis is not 100%, a pointer is inserted in the lefthand column adjacent to the second last and last nodes in that column. Nodes on this new row are constructed as sorting according to output is carried out. Note that the first bit of each output vector indicates whether or not an unstable machine is present('1'). If such a machine is found we do not try to separate the machines of that subset any further.

SECTION 8. Subroutine TREE

This subroutine prints out the diagnostic tree, a section of it, or a subtree.

The parameters of the subroutine are FL, the first level of the tree to be printed, HS, the root of the subtree, and LL, the last level of the tree to be printed. If HS is zero the entire first level, FL, is printed. If HS is non zero its value is the node number within the level*, from which the subtree is generated.

Description of Variables:

ZO is the width required to print the output bit string in fixed binary. (This information determines the width of the node.)

ZOH is half of ZO.

LG is the width in columns of each node.

LGH is half of LG.

FP is a pointer to the first node of the tree. Its value should be M#+1 if the first level printed, is the first level of the tree.

HP is a pointer to the root node of the subtree. Its value is zero if no subtree is being printed.

LP is a pointer to the last node of the tree. Its value should be M#+2 if the last level printed is the default last level of the tree.

CC is a counter from the center of one node to the center of the next node.

LD is the number of columns necessary to move to the left to reach the left hand end of the node.

* If HS is 5 then the node is the fifth node from the left on level FL.

CL is a vector indicating the columns in which the center of each node should appear. If the I^{th} node should not appear CL(I) is -1 or 0.

I1 is the default value for the pointer to the first level printed.

I and I1 are initialized to M#+1, K is set to zero to indicate level 0. CL is set to zero and the pointers FP, HP, and LP are set to zero.

The tree is scanned to find the first level to be printed. If FL has a value which exceeds the number of levels of the tree a message is printed and the first level pointer will point to the last level of the tree.

If the last level parameter is less than the first level parameter, the last level printed will be the last level of the tree. This will also occur if LL has any value which is not a valid level of the tree.

If HS is greater than zero, the first level, pointed to by LP, is scanned until the HS^{th} node is found and HP is set to point to that node. If the node is not found a message is printed and the entire first level will be printed instead of the single node specified.

The width of the node is calculated along with other pertinent information. If the last level of the tree is to be printed, the machine list is split up into the subsets required. Also, the number of subsets indicates the maximum number of nodes to be allotted space on a level. By starting at node M#+2 the machine set is scanned and the first machine of each subset has the corresponding value in CL set to -1. The first machines are identified because they

are found via a negative brother link.

If the last level is not to be printed the machine list can not be consulted since it will probably have more subsets than will have been detected at the last printed level. In this case the last level to be printed is scanned and each node is set to -1.

The centre node counter cc is initialized to the center of the left most node to be printed.

If a subtree is being printed SETUP is called with the root node number pointer as argument. Otherwise SETUP is called for each node on the first level with the node number as argument .

PUTRE:

Declaration of variables for inner blocks:

CLP is a pointer to the first node of each level and the last node printed.

DGR is the diagnostic resolution for each level.

DTR is the detection for each level.

SSC is a counter of the number of nodes on each level.

P is a pointer to the node being processed now.

Q is a pointer to the node last processed.

PC is a pointer to the centre column of the node being processed.

QC is a pointer to the centre column of the node last processed.

FC is a pointer to the first column.

LC is a pointer to the last column.

IPG is a page counter.

CNT is a counter of the number of machines.

OCNT is a counter of the number of subsets at last level printed.

OMCNT is a counter of the number of machines in a subset also as counter for the number of subsets in the level above.

LW is a pointer to the left hand edge column of the node.

RW is a pointer to the right hand edge, column of the node.

Z is a constant with value 0.

R1, R2, R3, R4, R5 are the buffers for the lines required to print one level or part of a level.

PIC is a character variable to hold SSC, just before print out.

CR1, CR2, CR3, CR4, CR5 are defined on R1, R2, R3, R4, R5.

CPIC is defined on PIC.

CLP, DGR, DTR and SSC are all initialized to zero, and P takes on the value of the first level pointer FP. First the detection and diagnostic resolutions are calculated for each level.

RSL: Scale is set at 1000/M#-1 because in M# there is one good machine. Base is set at $1000/2^{O\#}-1$ because, out of all the outputs there is one good output. Per cent detection and diagnostic resolution are being calculated. '1000' is used rather than '100' to keep more accuracy; the decimal point is moved to the left one

position in the printout. OMCNT takes on the number of machines in the subset of the brother node of P.

If the current level, is the level with the machines in it. (one level below the last level of nodes), the overall detection resolution is found by multiplying SCALE by (the number of machines minus the number of machines in the subset on the far left). Remember this subset contains the good machine. The calculation tells how many machines have been separated from the good machine. CNT goes through this level and counts the number of subsets. The overall diagnostic resolution is given by SCALE times (the number of subsets minus one). One is subtracted from the number of subsets because the smallest number of subsets is one not zero. Thus if there is one subset, there is zero diagnostic resolution.

If the current level is some level of nodes in the tree, the diagnostic and detection resolution are given by the level above, that is, the DGR and DTR values at level n are dependent on the DGR and DTR values at level n-1. We store the son node of Q in LW. We find the number of machines in the subset of that node and store in CNT. Again, the good machine will be in that subset, so the DTR value is found by multiplying 1000 by (1 minus CNT over OMCNT) where OMCNT is CNT for the level above. We up date OMCNT and initialize CNT at 0 and OCNT at 1 to find DGR. CNT counts the number of nodes on the level and OCNT counts the number of subsets (a new subset is reached each time a brother link is negative). DGR is calculated by multiplying

BASE by ((number of nodes over the number of subsets) -1). We repeat the above for the son of P.

COUNT:

This procedure counts the number of machines in the subset of node P, where P is the node number passed as a parameter.

We initialize FC and LC to 0 and 110 and IPG to 1.

PASS:

We print the tree heading and initialize the five buffers rows to blank.

LOOP:

This loop prints out a level of the tree or part of a level. We set QC to zero and Q to point to the first node on the first level printed. We set P to the brother of the former node.

If P is less than M# and greater than zero, the set of machines in their subsets is to be printed. We print R4. We print the left hand heading, then call MLIST to print the first machine of each subset. We print a blank heading on the next line, followed by the second machine of any of the subsets. IF indicates whether or not there are any. We print a heading on the next line and the third machines of any subset, if any exist. We print the diagnostic resolution on the next line (note the decimal is moved one place to

the left). We repeat with the fourth machines. We print the detection resolution on the next line, and print machines if any. If there are still machines to be printed IF will be '1'; machines are printed until IF is zero. Go to NXT.

If the current level is not the bottom level, nodes must be printed. R1, R2, R3, R5 take on the value of R4. The level is scanned from left to right and each node is processed as follows: PC is set to point at the center column of the node to be printed minus the value in the first column pointer.

If PC is too high, a new page must be started. If this subset is not finished arrows and dashes are printed to indicate the link up with the next page. Go to PRT.

If PC is not too high, SSC is incremented to indicate another node is to be added to the level node count. If PC is greater than zero, the node is to be printed. If the brother link of the previous node is greater than zero this node and the last node must be connected. The next loop does that from QC + LGH to PC - LGH. The shell of the node is created next; R3 and R5 form the bottom and top respectively R1 and R2 form the sides. R4 forms the son links. The inside of the node is filled. If the node has unstable machines in subset, the letters 'AST' replace the bottom of the node. SSC is stored in PIC, and each

character placed in R1. The output vector is changed from binary to hexadecimal and put in R2. PC is stored in QC and P in Q; P is updated to the brother node. If P is not zero, we repeat for that node. After all nodes are processed, the level is printed.

MLIST:

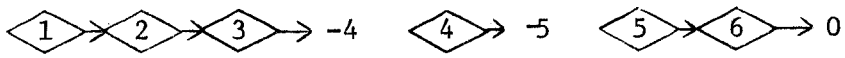
This procedure prints out the machines for each subset in specific columns. It receives the first machine in the first subset as parameter and returns a '1' if there is at least one machine in one subset left to be printed; otherwise a '0' is returned.

Each time this routine is entered the entire machine list is scanned. PC is set to point to the column in which the node is to be centered. If PC is too large, we return. If PC is greater than zero, the machine node is to be printed. If the brother link is greater than zero there is another machine to be printed so the returned value is set to '1'. The column pointer for that machine is changed to the one that was just printed, since the machines are printed one underneath the other. The column pointer for the machine just printed is set to zero, so it will not be printed again. P replaces Q and P is updated to its brother node.

If the brother node of P is negative, the column pointer for P is set to zero.

P replaces Q, and P is updated to the brother node. Notice this is done twice before the end of the outer loop if the brother of the original machine was in the same subset, which means that the column pointer of the present P will be zero if P is still in the same subset, otherwise it may have a value greater than zero.

A picture of the preceding function would look as follows for the machines below.



CL(1) = 8, CL(2), CL(3) = 0, CL(4) = 12, CL(5) = 18, CL(6) = 0

First call printout would be,

1 # # 4 # # 5

and CL(1) = 0, CL(2) = 8, CL(3) = 0, CL(4) = 0, CL(5) = 0 CL(6) = 18

Second call printout would be

2 # # 6

and CL(1), CL(2) = 0, CL(3) = 8, CL(4), CL(5), CL(6) = 0

The third call printout would be

3

NXT:

There is no more room to print the tree; a new page must be started. FC and LC are incremented by 100. This allows a ten column overlap, that is, the right most 10 columns of the first page are printed

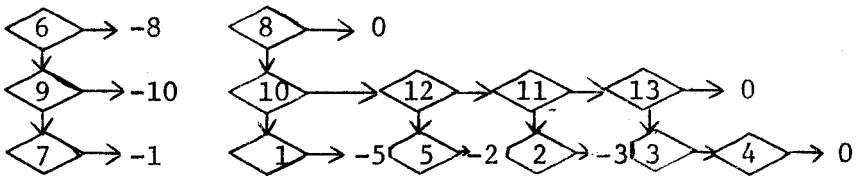
again on the second page. This makes the tree easier to read. The page counter IPG is incremented Go to pass.

SETUP:

This subroutine finds a column number for the center of each node in the tree, if the node is to be printed. If it is not to be printed its column pointer value remains at zero.

The following is a trace of SETUP for the tree developed in SECTION 7.

Tree:



Trace: values of pertinent variables.

CL(1), CL(2), CL(3), CL(5) = -1, CL(4) = 0,

LG = 6, CC = 4, M# = 5, UB = 15;

CALL SETUP(8)

I,J = 10;

CALL SETUP (10);

I,J = 1;

CALL SETUP (1);

CL(1) = -1; CL(1) = 4; CC = 4 + 6 + 1;

RETURN;

CL(1) = 4; LCC = 4; I = -5; CL(10) = $\frac{4+4}{2}$;

```
          RETURN;
CL(10) = 4; LCC = 4; I = 12;
          CALL SETUP (12);
          I, J = 5;
               CALL SETUP (5);
               CL (5) = -1; CL(5) = 11; CC = 11 + 6 + 1;
               RETURN;
          CL(5) = 11; LCC = 11; I = -2; CL(12) = (11+11)/2;
          RETURN;
CL(12) = 11; LCC = 11; I = 11;
          CALL SETUP (11);
          I, J = 2;
               CALL SETUP (2);
               CL(2) = -1; CL(2) = 18; CC = 18 + 6 + 1;
               RETURN.
          CL(2) = 18; LCC = 18; I = -3; CL(11) = (18+18)/2;
          RETURN;
CL(11) = 18; LCC = 18; I = 13;
          CALL SETUP (13);
          I, J = 3;
               CALL SETUP (3);
               CL(3) = -1; CL(3) = 25; CC = 25 + 6 + 1;
               RETURN;
```

CL(3) = 25; LCC = 25; I = 4;

CALL SETUP (4);

CL(4) = 0;

RETURN;

CL(4) = 0; I = 0; CL(13) = $\frac{25 + 25}{2}$;

RETURN;

CL(13) = 25; LCC = 25; I = 0; CL(8) = $\frac{4+25}{2}$;

RETURN;

Now the column pointers are:

CL(1) = 4, CL(2) = 18, CL(3) = 25, CL(4) = 0,

CL(5) = 11 , CL(6) = 0, CL(7) = 0, CL(8) = 14,

CL(9) = 0, CL(10) = 4, CL(11) = 18, CL(12) = 11,

CL(13) = 25, CL(14) = 0, CL(15) = 0.

Tree.