UNIVERSITY OF ILLINOIS

DIGITAL COMPUTER

LIBRARY ROUTINE Aux. Q 3 - 217

TITLE	Complete Circuit Analyzer (DOI only)
TYPE	Complete Program
NUMBER OF WORDS	Main Program: 161 Function Compiler: 124 D. O. I.: 25 $E(I_k)$ Routine: $11 + 4n + k$ where n is the number of nodes in the circuit and k is the total number of integers and + signs used to describe the circuit.
TEMPORARY STORAGE	0 to 9; 161 to $161 + n$; $172 + 4n + k$ to 874.
DURATION	Depends on circuit to be analyzed.
MAXIMUM NUMBER OF STATES PER CYCLE	$\left[\frac{703 - 4n - k}{6}\right]$ where [a] denotes greatest integer in a.
NOTE	An understanding of the method of using the program may be obtained without reading sections III, IV, and IX.

I. PURPOSE This routine analyzes a complete circuit for speed independence, the circuit being defined by the logical equations for each decision element and the specification of the initial state. In addition to this the user supplies a stop control on the program and a word describing what information is to be punched during the analysis.

The routine is limited in two ways: the circuits to be analyzed must contain no more than 39 nodes and secondly they must be of a class so that the maximum number of states per cycle is less than the number defined above. However in all cases tested so far these limitations have been overcome by suitably recombining elements and analyzing different recombinations [cf. 1]. The time taken to analyze a circuit with respect to one initial state depends on the circuit itself and is quite indeterminate, but only for extreme cases will it

be longer than ten minutes. The user of the program is seriously asked to consider what he feels should be added or eliminated from the program and to inform the laboratory of all suggestions.

II. INTRODUCTION A decision element is an element with an arbitrary number of input lines and one output line where the signals on all lines are taken only to possess two values, 0 and 1. To each decision element there corresponds a Boolean function of the input signals and possibly the output signal itself. We specify the state of the element by listing a set of signals on the input and output lines. The element is then said to be in equilibrium if the computed value of the function equals the signal assigned to the output line; otherwise it is said to be excited. An element is always said to act in such a way as to place itself in equilibrium and will only act if excited; but no restriction is placed on the time it takes to act.

An interconnection of decision elements at points called nodes has the property that each node has at most one output line feeding it and feeds at least one input line. Since no assumptions are made for the relative speeds of the decision elements, we cannot consider clocked or synchronous circuits. Hence, the above interconnection will only apply to asynchronous circuits.

A complete circuit is an asynchronous circuit such that every node has one and only one decision element feeding it. Any asynchronous circuit can be considered to be a complete circuit by attaching fed-back delay elements (elements whose input equals its output) to those nodes which have no decision elements feeding them (see Section I). Since every node in a complete circuit is fed by a decision element we shall label each node by the decision element feeding it.

An immediate state of a complete circuit is defined by listing the value of the signal (either 0 or 1) at each node in the circuit. Since every node is fed by one decision element the immediate state (abbreviated as I-state) determines uniquely those nodes which are excited. An I-state S is an equilibrium state if every decision element is in equilibrium. An I-state B is said to directly follow an I-state A if B results from A by no more than one decision element acting and if that decision element was excited when the

circuit was in I-state A. B is said to follow A if there exists a sequence of states $A = A_1, A_2, A_3, \ldots A_k = B$ such that each state $A_{i+1} (i = 1 \text{ to } k - 1)$ directly follows the state A_i. It is important to note that there might be states A and B where neither A follows B nor B follows A.

A cumulative state (abbreviated C-state) is defined with respect to some initial I-state, I_0, by listing the number of changes that have occurred at each node since the circuit was last placed in I_0.

A complete circuit is said to be speed independent with respect to an I-state S if for every I-state which follows S no decision element passes from an excited state to an equilibrium state without acting.

If now we extend the definitions of follows and directly follows to the C-states and use "follows" as a partial ordering relation, then it can be shown that the set of C-states of a speed-independent circuit forms a semi modular lattice [cf. 1]. Using this fact it then can be shown that there are only two sub-classes of speed independent circuits defined by the types of lattices of the C-states.

A speed independent circuit is said to be totally sequential with respect to an initial state S if for any two C-states A and B which follow S, either A follows B or B follows A holds. This is equivalent to saying that for any I-state T which follows S at most one decision element is excited; hence at most one I-state may directly follow T.

A speed independent circuit is said to be distributive with respect to an initial state S if for any three C-states A, B and C which follow S with C directly following A and B, then there exists a C-state D which follows S such that both A and B directly follow D. Since a C-state may directly follow itself, it is clear that a totally sequential circuit is distributive. If one tries to apply this definition to the I-states then in general it will not work, but the cases where it fails are those circuits which eventually break up into at least two independent parts. That is the circuit behavior at a particular set of the nodes is independent of the circuit behavior at the remaining nodes. We shall define the three classes as totally sequential, distributive but not totally sequential, and semi modular but not totally sequential or distributive.

III. METHOD Given a complete circuit with n nodes labeled from 1 to n then an I-state I_k will be denoted by a vector

$$I_k = (i_k^1, i_k^2, i_k^3, \cdots i_k^n)$$

where the i_k^j are the values of the signals either 0 or 1 at the nodes. The set of excited decision elements $E(I_k)$ will be denoted by a vector

$$E(I_k) = (e_k^1, e_k^2, e_k^3, \cdots e_k^n)$$

where $e_k^j = 1$ if the j^{th} node is excited, otherwise $e_k^j = 0$. The notation $E(I_k)$ formalizes the fact that the set of excited nodes is uniquely determined by the I-state I_k. These numbers are stored in the machine as words with the j^{th} component in the $(2^{-j})^{th}$ position, their sign always being positive.

The memory is divided into two sections A and B: one containing the list of current states and excited nodes during a cycle, the other containing a list of states which directly follow the current states along with their excited nodes. In addition these lists contain a third word for each state which is used for testing distributivity and will be described later. The routine begins by selecting a state I_k in the current list and its excited nodes $E(I_k)$. It then forms a new state by selecting an excited node appearing in $(e_k^1 e_k^2 \cdots e_k^n)$, say it is e_k^j. It forms a new state I_k^* by allowing this node to pass to equilibrium hence

$$I_k^* = (i_k^1, i_k^2, \cdots i_k^{j-1}, \overline{i_k^j}, i_k^{j+1} \cdots i_k^n)$$

It then examines the list of directly following states to see if I_k^* has already been generated. If it had not been generated it plants it in the new list and enters the function routine to calculate $E(I_k^*)$ and stores the excited nodes corresponding to I_k^*. (If it was in the list it would already have done so). Next it forms $E(I_k)^*$ which is

$$E(I_k)^* = (e_k^1, e_k^2, \cdots e_k^{j-1}, 0, e_k^{j+1}, \cdots e_k^n)$$

It now is able to make a test for speed independence which is that no other node in $E(I_k)^*$ except possibly the j^{th} node passes into

equilibrium. This is equivalent mathematically to:

$$E(I_k)^* \subseteq E(I_k^*) \quad \text{(componentwise)}$$

If $E(I_k)^* \not\subseteq E(I_k^*)$ the circuit has violated speed independence and the routine jumps to a failure stop (see "non-programmed stops") otherwise it adjusts the third word for I_k^* (see distributive test) and returns to I_k and picks the next excited node in the sequence e_k^{j+1} e_k^{j+2} ... e_k^n. It continues this process until all the states which directly follow I_k have been exhausted. It then goes to the next state in the current list, repeats the process on this state and continuing in this manner it finally exhausts the current list. An overwrite test is made to see if the directly following list overwrites its memory capacity.

IV. SPECIAL TESTS Initially a switch had been set placing the circuit in the totally sequential class. If after exhausting any list (including the first list of just the initial state itself) it did not generate more than one state in the list that directly follows the current list, then the circuit remains totally sequential and no distributive test need be made. However, the first time more than one state is generated the circuit is no longer totally sequential, and a distributive test must be made on the next set of directly following states. It need not make the test when it first branches since trivially all those states arose from one state.

We now define a set of causation nodes for a directly following state $C(I_k^*)$:

$$C(I_k^*) = (c_k^1, c_k^2, c_k^3 \cdots c_k^n)$$

Let $$I_k^* = (i_k^1\ i_k^2\ i_k^3 \cdots i_k^n)$$

then $c_k^j = 1$ if there exists a state I_m in the current list where

$$I_m = (i_m^1\ i_m^2\ i_m^3 \cdots i_m^{j-1}\ \overline{i_m^j}\ i_m^{j+1} \cdots i_m^n)$$

and where I_k^* directly follows I_m.

Now assume we have exhausted the current list and that a distributive test must be made. The routine then works with the directly following list but instead of using the excited nodes to obtain new states it uses the causation nodes. Since we are using $C(I_p)$ instead of $E(I_p)$ then the set of states I_p^* (which directly follow I_p using $C(I_p)$ as the excited nodes) will always be found in the list of current states. This results directly from the definition of $C(I_p)$. The routine tests for distributivity by testing for speed independence with respect to the causation nodes. That is if

$$C(I_p)^* \not\subseteq C(I_p^*)$$

the circuit is no longer distributive (provided also that the circuit is composed of not more than one independent circuit). Hence, if it fails the test the routine reclassifies the circuit as semi modular and no longer needs to make the distributive test. If not it will continue until the list of directly following states is exhausted. In either case the routine then relabels the lists by calling the present directly following list the new current list and comes to a black switch stop (see non-programmed stops).

V. PROGRAMMED AND NON-PROGRAMMED STOPS

Given a complete circuit with n nodes then there is a total of 2^n possible states that can be specified at the nodes. Hence, given an initial state, it is sufficient to generate 2^n new states to test for speed independence. However, when n gets large the time factor is prohibitive, and moreover most useful circuits do not go through all 2^n possible states. Usually the designer has an idea as to what the circuit should do and hence does not need to use this type of programmed stop (designated as a normal stop). The routine provides for two other programmed stops which are as follows:

(1) State Count Stop. This stops after a specified number of new states have been generated.

(2) Node Change Stop. This stops after a selected node has under gone a specified number of changes.

All other stops are designated as non-programmed stops and are:

(1) Failure Stop. This stops when the circuit fails the speed independence test and punches F followed by the current state, its excited nodes, the directly following state and its excited nodes. It then tests for overflow stop.

(2) Overflow Stop. This punches an "0" when the list of directly following states exceeds its memory capacity. If the circuit failed when this happened the failure might have been due to the overwriting. In some instances the compiler may also be destroyed, thus necessitating the rereading of the entire program if another circuit or state is to be analyzed. A program has been written which eliminates the distributive test and which overwrites the compiling routine so that the maximum of new state is $\frac{876-4n-k}{4}$ instead of $\frac{703-4n-k}{6}$.

(3) Equilibrium Stop. If the circuit passes to equilibrium, then the character E is punched followed by the equilibrium state.

(4) Black Switch Stop. Normally the circuit is run with the black switch down. If the switch is placed in the stop position, the program will stop at the end of a cycle. By a white switch by-pass the user may proceed to the final stop.

All of the above stops terminate in the final stop which punches the circuit type S, D, or M signifying totally sequential, distributive, or semi-modular. The routine then transfers to an OF.

VI. OUTPUT CONTROL The user may select a set of nodes such that when any node of this set changes, an output occurs giving the current state, its excited nodes, the directly following state and its excited nodes.

VII. DATA TAPE PREPARATION

There are two possible input conditions to distinguish: one is a complete circuit together with an initial state and the other is a new initial state to be used when an already existing circuit has been compiled. These are distinguished by having a character consisting of a single "fifth hole" under the reader when a new circuit is to be analyzed and otherwise having a different arbitrary 5^{th} hole character under the reader if a new initial state is

to be supplied.

The circuit designation is accomplished by first numbering the nodes. Then each nodal equation $z_i = f_i(z_1 \ldots z_k)$ is written in the following form

$$z_i = \bigvee_{j=1}^{m} g_{ij}$$ where g_{ij} is a product of the z_i's either complemented or uncomplemented. Thus the function

$$z_5 = (z_1 \bar{z}_2)(z_5 + z_7 \bar{z}_{10})$$

would be written as

$$z_5 = (z_1 \bar{z}_2)[z_5 \overline{(z_7 \bar{z}_{10})} \vee z_7 \bar{z}_{10} \bar{z}_5]$$

$$z_5 = z_1 \bar{z}_2 \bar{z}_{10} \bar{z}_5 \bar{z}_7 \vee z_1 \bar{z}_2 z_5 (\bar{z}_7 \vee z_{10})$$

$$z_5 = z_1 \bar{z}_2 \bar{z}_{10} \bar{z}_5 z_7 \vee z_1 \bar{z}_2 z_5 \bar{z}_7 \vee z_1 \bar{z}_2 z_5 z_{10}$$

The tape is now punched in the following way:

(1) The number of nodes (n) is first punched as a decimal integer terminated by a 5^{th} hole character.

(2) The n nodal equations are then punched in order from 1 to n with the following conventions:

(a) The left hand sides of the equations are omitted (i.e."z_1 =" is omitted).

(b) The term z_i is designated by the decimal integer i terminated by a space.

(c) The term $\bar{z}_i$ is designated by the decimal integer i terminated by a prime.

(d) The operation v (or) is designated by +(K).

(e) Each nodal equation is terminated by an N.

(f) Any number of 5^{th} hole characters may precede an integer but no 5^{th} hole characters can separate the digits of an integer.

(g) Any number of 5^{th} hole characters may precede or follow a + or N.

Thus several correct ways of punching the nodal equation in the example are:

(i) 1 2'10'5'7 +1 2'5 7'+1 2'5 10 N

(ii) 1 2''10' 5'''7 ' +' '1 2'5 7'+1 2'5 10 N

where if the first term were written as

(iii) 1 2'1 0'5'7 ...

it would be incorrect since a space separates the digits of the number 10.

(3) Following the nodal equations the initial state is punched with as many sexadecimal characters as are needed to describe the n nodes. Thus if we had 5 nodes and the initial state were specified as

(10111) for $(i_1\ i_2\ i_3\ i_4\ i_5)$

this would be punched as S8, the digits being grouped in sets of 4 from left to right.

The output during the running of the program will have the same number of characters and the same significance of the grouping.

(4) Following the initial state the stop information is punched. If a normal stop (after 2^n new states) is required then the character N is punched. If a state count stop is required, then the number of new states is punched as a decimal integer terminated by a 5^{th} hole character followed by an N. If a node change stop is required then the number of changes is punched as a decimal integer terminated by a 5^{th} hole character followed by the node number punched as an decimal integer terminated by a 5^{th} hole character.

(5) Following the stop control the print control constant is punched. This constant has the significance that wherever a one appears it will output when that node changes. Thus a user can, if he wishes, obtain more output by using a hand punch. For example, in a 5 node circuit if the user is only interested in changes occuring in the second node then he would punch the characters 40. If however, he wished also to examine the changes in the 1^{st} and 5^{th} nodes he would punch N8. The

termination of these last characters by a "5^{th} hole delay" character will allow the user to read in another circuit on the same tape without resetting the tape in the reader. The termination by any other 5^{th} hole character will allow the user to read in another initial state, stop control, and punch constant.

The important things to remember are:

(A) No 5^{th} hole characters may separate the digits of any integer and all integers are terminated by 5^{th} hole characters.

(B) Every nodal equation is terminated by N and every integer representing a node is terminated by either a space or prime.

(C) The list of nodal equations is preceded by the number of the equations.

(D) A "5^{th} hole only" character must be under the reader to read in a circuit. Any other 5^{th} hole character reads an initial state, stop control, and punch constant.

(E) The number of sexadecimal characters is the number required to represent n bits (n is the number of nodes)

(F) Stop constants:

N stops after 2^n states

a N stops after a states (a decimal integer)

a b stops after the b^{th} node has undergone a changes, (a,b both decimal integers)

VIII. USE The main tape is read into a cleared memory. If the sum check fails it stops on an FF order. The data tape is placed in the reader and the nodal equations, initial state, stop control and print constant are read in. One throw of the black switch will allow the circuit to be tested through one cycle. Normally the program is run in the stop disable position. The program comes to an OF when any of the programmed or non-programmed stops occur. Then depending on the 5^{th} hole character under the reader, either a new circuit or new initial state will be read in with a white switch bypass.

IX. COMPILING ROUTINE

This is a special input subroutine used to compile the

subroutine for calculating $E(I_k)$. The subroutine calculates $E(I_k)$ by the following method:

(1) First n numbers are stored where if z_i is 1 the number corresponding to z_i is negative and if z_i is 0 the number is positive.

(2) The nodal functions are calculated in order by a series of conditional transfers where if the function is 1, a 1 is added in the position corresponding to that equation. For example the set of orders:

L5	(z_1)	36 (*)	
F1	(z_2)	36 (*)	
32		26 (**)	
L5	(z_4)	36 (***)	*
L5	(z_6)	36 (***)	
L5	(equation values)	L4 (z_3 marker)	**
50		40 (equation values)	
Next equation			***

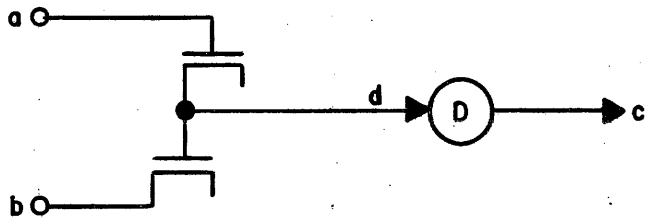
corresponds to the equation $z_3 = z_1\bar{z}_2 \vee z_4 z_6$.

(3) At the end of the routine the function $E(I_k) = I_k$ + (equation values) (digitwise) is formed and control is transferred to a fixed point in the main program.

(4) In addition to compiling the routine $E(I_k)$ the compiler plants appropriate end-test constants and addresses in the main routine.

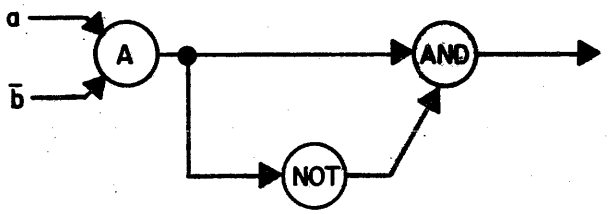
X. Miscellaneous Aspects

Sometimes it is convenient for a circuit to give a failure in a disallowed state. That is, an element might have a certain combination of inputs which gives an unpredictable output. For example, using positive logic (pos voltage = 1) the point d in the circuit is not defined for the input combination a = 1 b = 0. However, there is a nice way of taking care of this which is to include the term $\bar{c}\,a\bar{b}$ in the equation defining c.



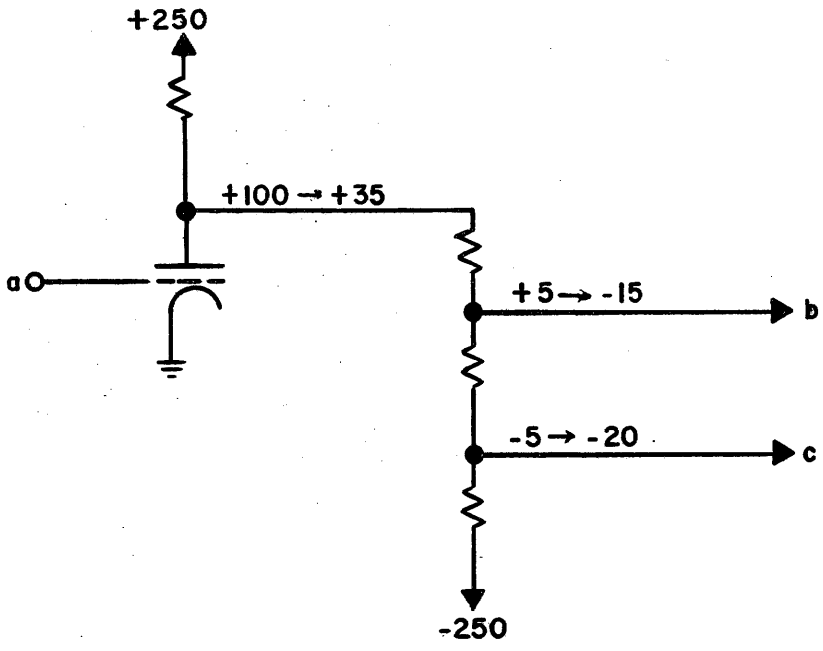
This would have the effect of making $c = \bar{c}$ when $a\bar{b} = 1$ and would cause the element to alternately flip from 1 to 0 for each cycle. (This would be approximately what happens physically to the line d).

Another way is to include the circuit below which will always fail when $a\bar{b} = 1$.



The element A (in this case an "and" element) can be expanded to include all disallowed states in a given circuit.

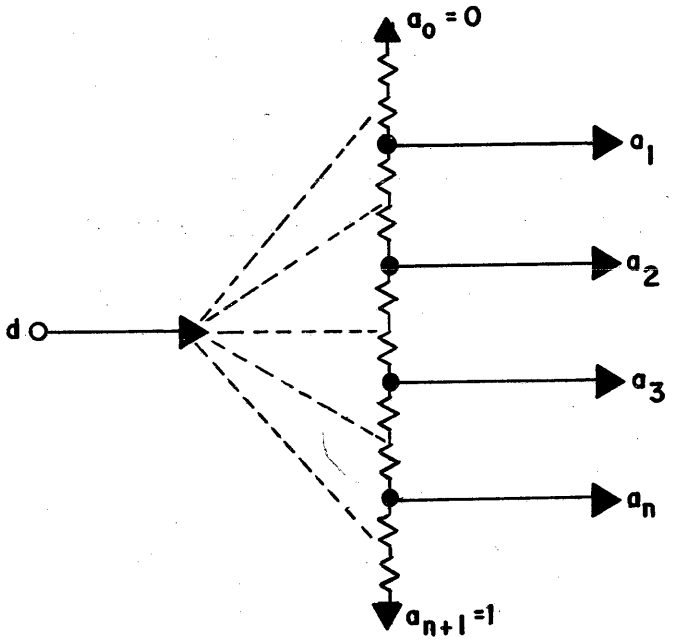
Sometimes it is convenient to represent divider chains in a circuit, for consider the following circuit:



where both lines b and c may be shunted by arbitrary capacitors. Thus it is no longer possible to say b = c as would be done normally. This "bleeder chain has the property that at no time can b have the value 1 and c have the value 0 in negative logic. Thus b can only take on the value $\bar{a}$ when c is 1. The logic for lines b and c would then be

$$b = \bar{a}c$$
$$c = (\bar{a} \vee b)$$

The general divider chain with outputs $a_1\ a_2 \ldots a_n$ pictured below

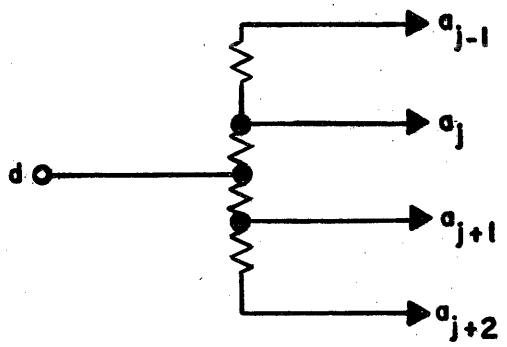


would have as the system of equations

$$a_1 = da_2 \vee a_0$$
$$a_2 = da_3 \vee a_1$$
$$a_n = da_{n+1} \vee a_{n-1}$$

where d is the driving signal. d has the property that if it is situated between

a_j and a_{j+1}, as shown in the figure,



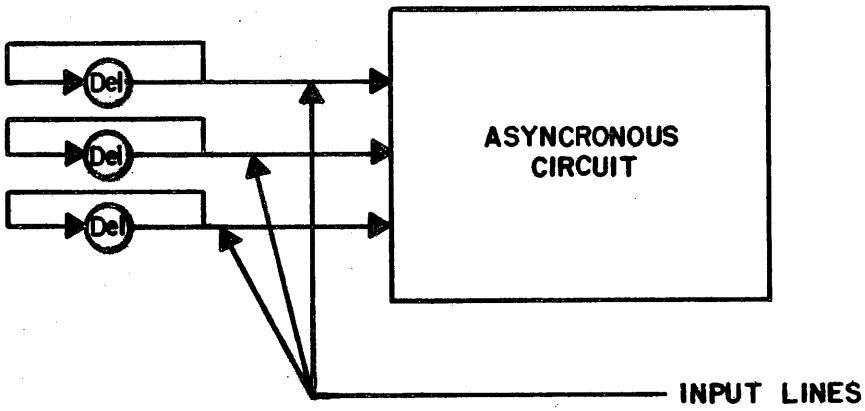
then the lowest voltage value of d = 1 must be greater than the highest voltage value at $a_{j+2} = 0$. Otherwise, it would be possible for $a_{j+1} = 1$ and $a_{j+2} = 0$. Also, the highest voltage value of d = 0 must be less than the lowest voltage value at $a_{j-1} = 1$, for if not it would be possible for $a_{j-1} = 1$ and $a_j = 0$. The divider using positive logic becomes $a_0 = 1$, $a_{n+1} = 0$,

$$a_1 = da_0 \vee a_2$$

$$a_2 = da_1 \vee a_3$$

$$a_n = da_{n-1} \vee a_{n+1}$$

A general asynchronous circuit can be handled by feeding all nodes having no decision elements attached to them with fed back delay elements. This then allows the user to attach a signal to what normally would be called the input lines.



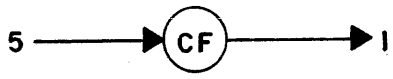
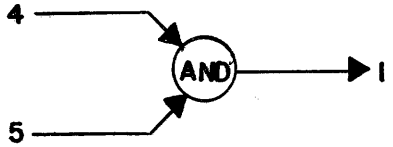
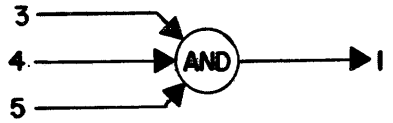
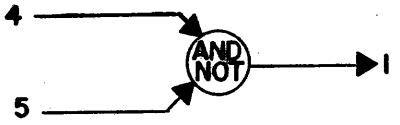
This method requires that the inputs remain fixed during the analysis. If

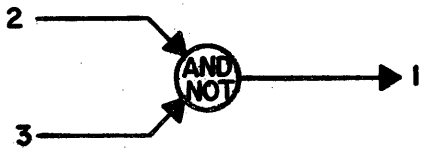
one wishes one of them to change from a 0 to 1 then that line could be tied to a "one" element (i.e. $z_1 = z_1 \vee \bar{z}_1$) which was initially placed in the 0 state.

XI. REPRESENTATION OF VARIOUS LOGICAL ELEMENTS

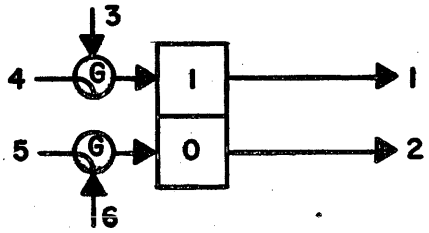
Several examples of data tape preparations for Illiac type logical elements and special logical elements are given below For construction of these elements the reader is referred to Reference 2 for Illiac types and References 3, 4, 5 for several of the special types. These elements are written with integers specifying input and output lines to indicate how the set of nodal equations would be typed on the tape. All outputs are designated by node number 1 and if needed node number 2.

ILLIAC TYPES

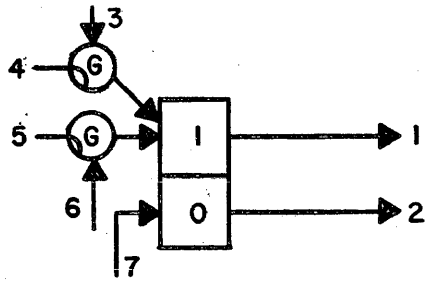
Circuit Designation	The First and Possibly the Second Nodal Equation would be Punched As:
	
Cathode Follower or Delay:	5 N
	
Not:	3'N
	
And:	4 5 N
	
3 Input And:	3 4 5 N
	
Or:	4 +5 N



And Not: 2'+3'N

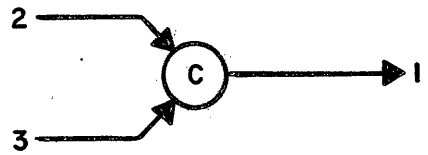


Flipflop with Two Gates: 3'4 +2'N
5 6'+1'N

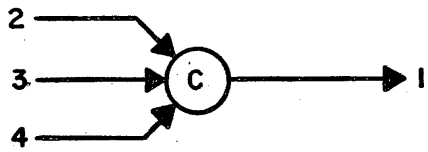


Flipflop with Clear and Two Gates: 3'4 +5 6'+2'N
7 +1'N

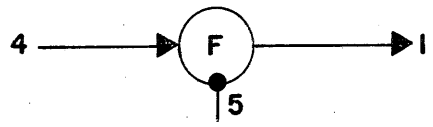
ADDITIONAL TYPES



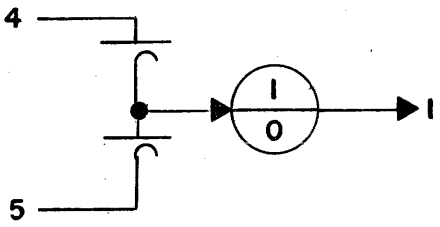
C-element: 2 3 +1 2 +1 3 N



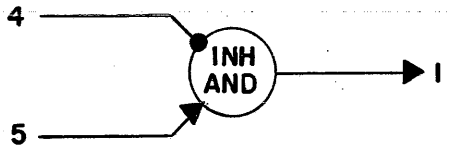
Three Input C-element: 2 3 4 +1 2 +1 3 +1 4 N



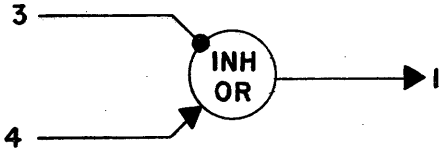
F-element: 4 5 +1 5'N



Single Output Flipflop
with Two Diode Gates: 4 5 +1 4 5'+1'4'5 N
(negative logic)



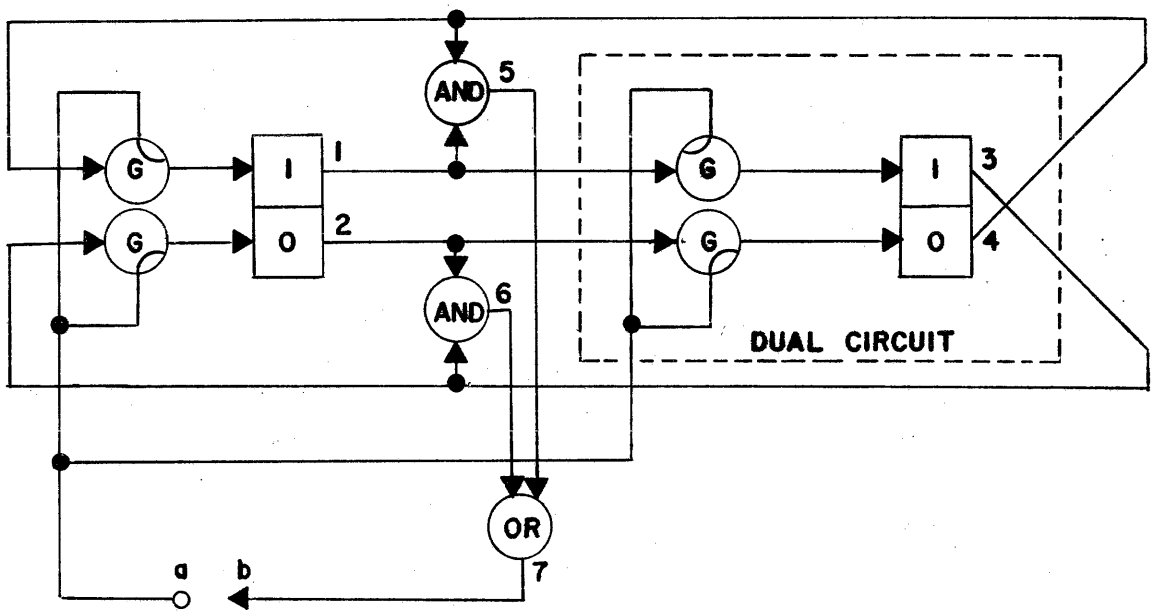
Inhibit And: 4'5 N



Inhibit-or: 3'+4 N

XII. A SPEED-INDEPENDENT COUNTER

A one state binary counter is diagrammed on Pg. 18. The node numbering is indicated at the output of each element and the flipflop and gates are included in two logical equations. The dual flipflop has the dual equations. The point a is the enabling signal and the point b is the complement of the completion signal. These were tied together to cause the counter to cycle indefinitely.



The equations for this circuit were typed as:

7	Number of Nodes
4'7 +2'N	ordinary flipflop with gates
3'7 +1'N	
1'4'+7 4'N	dual flipflop with gates
2'3'+7 3'N	
1 4 N	"and" element 5
2 3 N	"and" element 6
5 +6 N	"or" element 7
+ +	initial state
4 7	Count 4 changes in node 7
0 2	Punch when node 7 changes

The Output that followed was:	
	EXPLANATION
	Flipflops — Node 7 change
+2 02 +0 20	1011 — 1 → 0
98 02 9+ 40	1001 — 0 → 1
52 02 50 10	0101 — 1 → 0
64 02 66 80	0110 — 0 → 1
S	Totally Sequential.

The columns in the output are respectively: current state, excited nodes for current state, directly following state, and excited nodes for directly following state. Due to the print out selected the two states per output are the state where node number 7 is excited and the state that directly follows that state when node number 7 changes. The S at the end of the output indicates that the circuit is surprisingly enough, totally sequential.

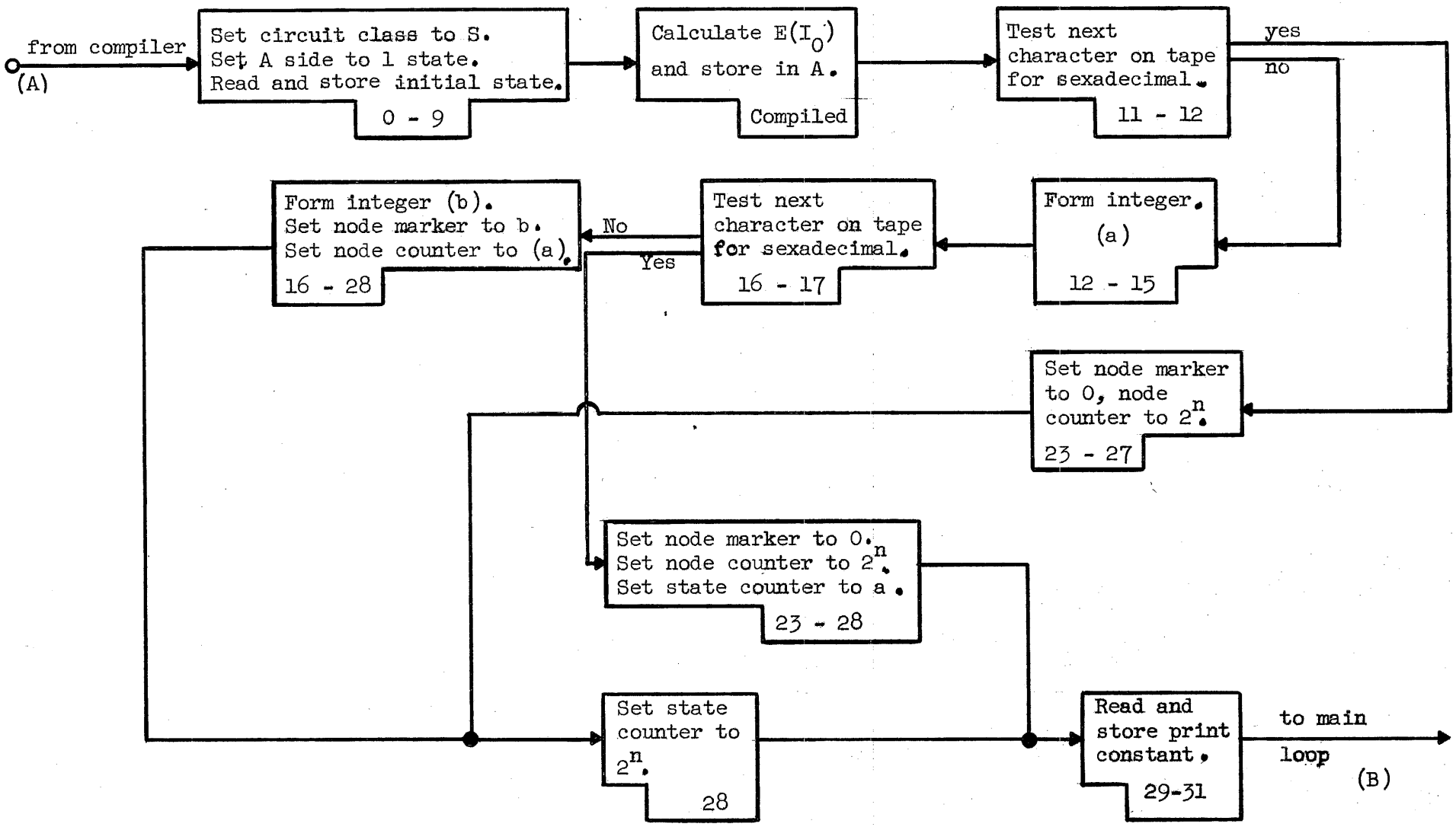
XIII. REFERENCES

1. D. E. Muller, "Theory of Asynchronous Circuits," Internal Report No. 66, University of Illinois Digital Computer Laboratory (Dec. 6, 1955).
2. Staff of the Computer Laboratory, Ordvac Manual, University of Illinois Digital Computer Laboratory (Jan. 24, 1955).
3. W. Poppelbaum, "A Fast Junction Transistor Flipflop with Stablilized Output Levels," University of Illinois Digital Computer Laboratory (Jan. 24, 1955)
4. J. M. Wier, "Some Direct Coupled Computer Circuits Utilizing NPN and PNP Transistors in Combination," Internal Report No. 65, University of Illinois Digital Computer Laboratory (Aug. 31, 1955).
5. Staff of the Computer Laboratory, "Trance Circuits," University of Illinois Digital Computer Laboratory (May 11, 1956).

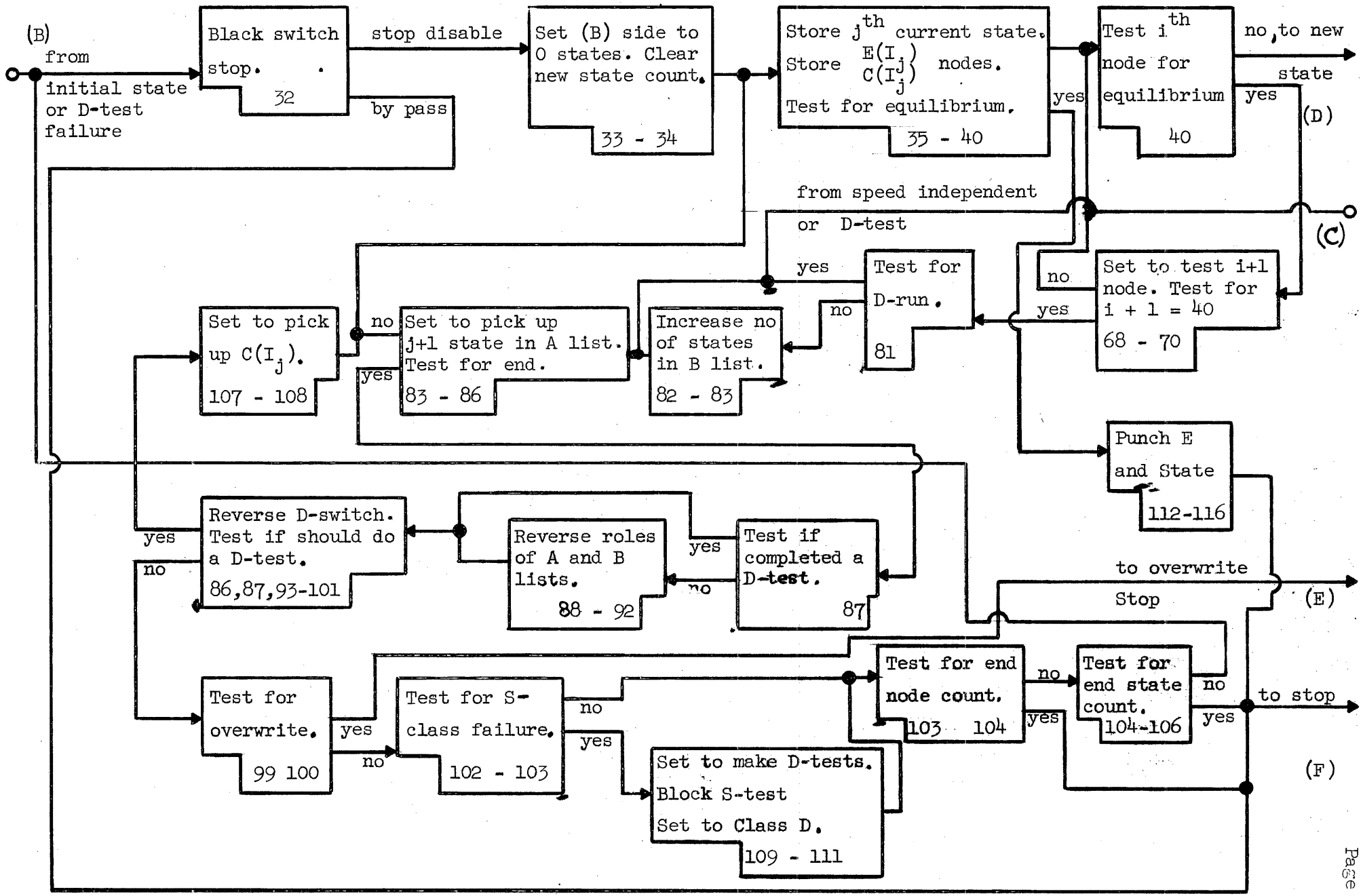
Rt: 9/23/59

DATE June 14, 1956
PROGRAMMED BY W. Scott Barthy
APPROVED BY J P Nash

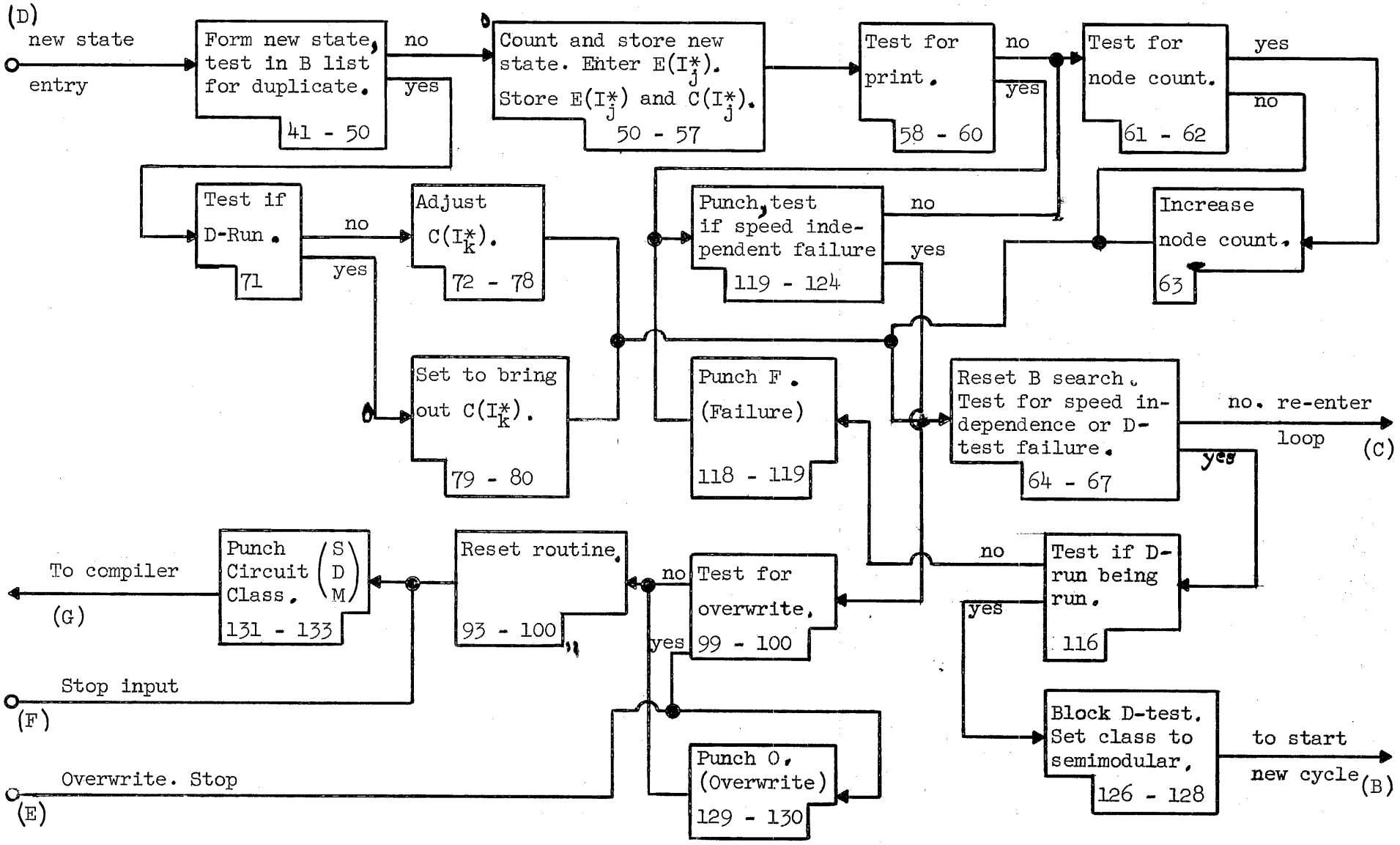
WSB/mge



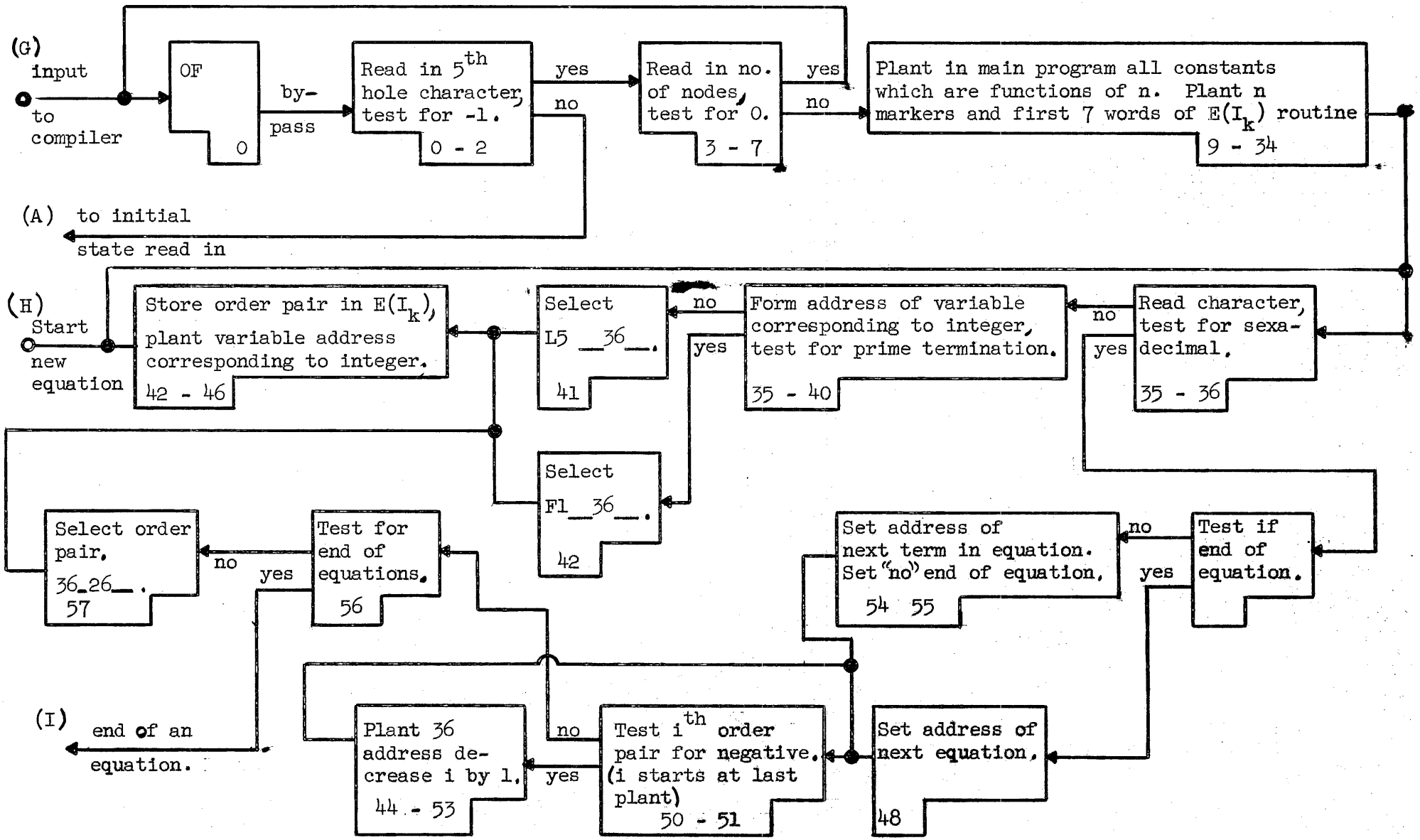
Main Program Part I (Initial State Read In)
(Relative Locations 0 - 31)



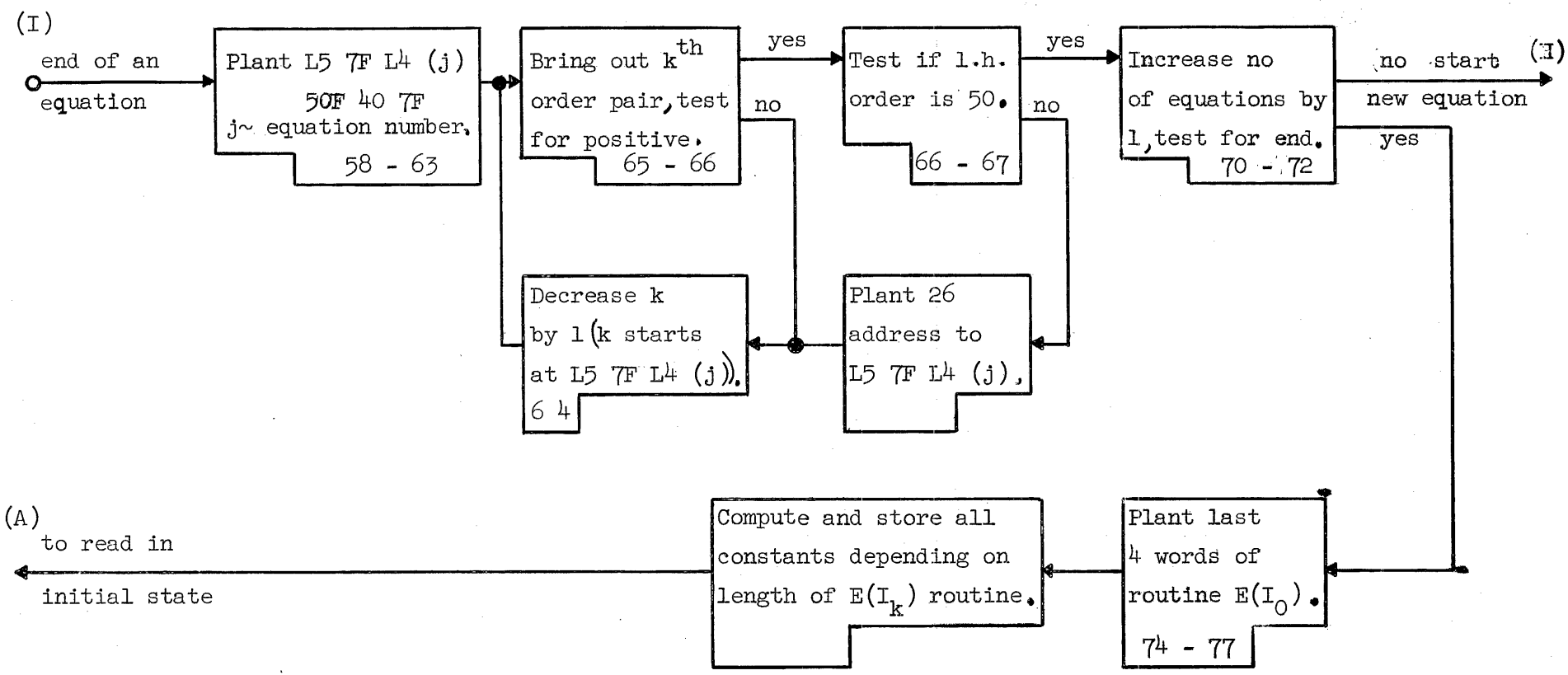
Main Program Part II
(Relative Location 0 - 150)



Main Program Part III
(Relative Locations 0 - 150)



Compiler Part I
(Relative locations 0 - 123)



Compiler Part II
(Relative locations 0 - 123)

LOCATION	ORDER		NOTES
Library Routine X1 D.O.I.			
	00 3K		
0	00 F		
	00 10F		location of main routine
1	00 F		
	00 875F		location of compiler
2	00 F		
	00 874F		memory constant
Main Routine			
	00 10K		
0	L5 21L		
	46 132L		set to punch S
1	F1 22L		
	42 134L		set S test
2	L5 135L		
	40 136L		set 80F 00F to D switch
3	F5 6L		
	42 137L		set a side to 1 state
4	81 ( )F	p	
	50 139L		clear Q
5	10 ( )F	p	
	S5 8L		read initial state
6	40 6F		
	40 ( )F	a.	
7	L5 5L		
	42 ( )F	e	set function routine
8	26 ( )F	f	to function routine
	40 ( )F	a + m/6	store excited nodes
9	L5 55L		
	42 ( )F	e	restore function routine
10	L5 146L		
	40 2F		set 2^n to 2
11	81 4F		
	L0 140L		-10

LOCATION	ORDER	NOTES PAGE 2
12	36 22L	test for 2^n state stop
	L4 140L	+ 10
13	50 139L	clear Q
	74 140L	x 10
14	00 4F	
	91 4F	read integer α
15	32 13L	store in 1
	S5 40F	
16	40 1F	
	81 4F	
17	L0 140L	- 10
	32 23L	test for α state stop
18	L4 140L	+ 10
	50 139L	clear Q
19	74 140L	x 10
	00 4F	read integer β
20	91 4F	
	36 19L	
21	S5 706F	(S)
	26 25L	
22	L5 146L	
	40 1F	set 2^n to 1
23	22 24L	
	L5 1F	set 1 to 2
24	40 2F	
	L5 15L	
25	42 46L	set node marker
	L1 1F	
26	40 141L	stop node counter
	19 ( )F	
27	00 1F	
	40 143L	store node marker

LOCATION	ORDER		NOTES PAGE 3
28	L1 2F		
	40 142L		set state counter
29	81 ( )F	p	
	50 139L		clear Q
30	10 ( )F	p	
	S5 ( )F	b_1	plant print constant
31	40 144L		
	41 9F		clear failure indicator
32	24 33L		stop
	26 131L		end
33	L5 30L		
	42 138L		set (b) side to 0
34	41 8F		clear new state
	L5 ( )F	a_1	state to 4
35	40 4F		
	L5 ( )F	$a_1+m/6$	$\binom{\text{ex}}{\text{caus}}$ nodes to 5
36	40 5F		
	40 3F		
37	L3 5F		
	36 112L		test for equilibrium
38	49 1F		set marker
	F1 1F		set marker
39	40 2F		
	L5 3F		
40	L4 3F		
	40 3F		test i^{th} node for
41	36 68L		equilibrium
	50 4F		
42	S5 67F		(D)
	J0 1F		form new state
43	S0 643F		(M)
	S0 F		
44	L4 1F		
	40 6F		

LOCATION	ORDER		NOTES PAGE 4
45	L5 46L		
	22 49L		
46	L5 6F		
	L0 ( )F	b_1	test in (b) list
47	40 F		
	L3 F		for duplicate
48	36 71L		
	F5 46L		
49	42 46L		
	L0 138L		(b) end test
50	36 46L		
	F5 8F		count new state
51	40 8F		
	26 ( )F	f	to function routine
52	00 1F	w	
	40 ( )F	$b_1+m/6$	store excited nodes
53	L5 6F		
	40 ( )F	b_1	store new state
54	L5 1F		
	40 ( )F	$b_1+m/3$	store causation node
55	F5 52L		
	42 52L		
56	F5 53L		
	42 53L		increase store
57	F5 54L		addresses
	42 54L		
58	50 144L		print constant
	J0 1F		
59	S1 F		
	36 61L		
60	93 135F		2 C. R.
	22 119L		to punch
61	50 143L		node constant
	J0 1F		

LOCATION	ORDER		NOTES · PAGE 5
62	S1 F		
	36 64L		
63	F5 141L		
	40 141L		count nodes
64	L5 30L		
	42 46L		reset (b) search
65	50 5F		
	J0 2F		
66	S5 F		
	J0 7F		speed independent
67	K0 F		or D-test
	32 116L		
68	L5 1F		
	10 1F		shift marker
69	40 1F		
	L3 3F		test for end
70	36 81L		
	22 38L		to i^{th} node.
71	L1 136L		
	36 79L		test if D run
72	L5 46L		
	L4 147L		+ m/6
73	42 77L		
	L4 147L		+ m/6
74	42 75L		
	42 76L		
75	00 1F		
	L5 ( )F		adjust causation
76	L4 1F		nodes
	40 ( )F		
77	00 1F	W	
	L5 ( )F		store$\binom{ex}{caus}$ nodes
78	40 7F		
	26 64L		to reset (b) search

LOCATION	ORDER	NOTES PAGE 6
79	L5 46L	
	L4 148L	+ m/3
80	42 77L	set to store causation
	22 77L	nodes in 7
81	L1 136L	
	32 83L	jump if testing D
82	L5 46L	
	L4 8F	set b end test constant
83	42 138L	
	F5 35L	
84	42 35L	increase addresses
	F5 34L	on (a) side
85	42 34L	
	L0 137L	test for end
86	32 34L	
	L1 136L	reverse D switch
87	40 136L	
	32 96L	jump if completed D-run
88	50 137L	
	L5 138L	interchange (a) and
89	42 137L	
	S5 F	(b) end test constants
90	42 138L	
	L5 30L	
91	50 6L	
	42 6L	reverse (a) and (b)
92	S5 F	markers
	42 30L	
93	L5 30L	
	42 46L	set b_1
94	42 53L	
	L4 147L	+ m/6
95	42 52L	b_1 + m/6
	L4 147L	+ m/6

LOCATION	ORDER	NOTES	PAGE 7
96	42 54L		b_1 + m/3
	L5 6L		
97	42 34L		a_1 set
	L4 147L		+ m/6
98	42 35L		a_1 + m/6 set
	42 8L		
99	L5 8F		
	F0 147L		- m/6 - 1
100	36 129L		jump for overwrite
	26 (101)L		
101	L1 136L		jump if D-run
	36 107L		coming up
102	L5 8F		
	L4 134L		jump if S fails
103	36 109L		
	L5 141L		test node count
104	36 131L		
	L5 8F		
105	L4 142L		
	40 142L		add states
106	36 131L		test state count
	26 32L		to start
107	L5 34L		
	L4 148L		+ m/3
108	42 35L		
	22 34L		to D-run
109	43 134L		block S test
	F1 136L		set D switch
110	40 136L		
	L5 42L		set to punch D
111	46 132L		
	22 104L		
112	92 135F		2 C.R.
	92 259F		letters

LOCATION	ORDER		NOTES PAGE 8
113	92 194F		E
	92 707F		nos.
114	93 963F		
	L5 4F		
115	00 1F		
	82 ( )F	p	punch E-state
116	26 131L		end
	L1 136L		test for D-run
117	32 126L		
	40 9F		set failure switch
118	92 135F		2 C. R.
	92 898F		F
119	93 963F		
	L5 4F		
120	00 1F		
	82 ( )F	p	
121	F5 119L		
	42 119L		to next output
122	L0 145L		
	36 119L		
123	L5 16L		reset
	42 119L		
124	L5 9F		
	36 61L		normal punch jump
125	L5 128L		
	42 100L		
126	26 93L		to reset
	43 136L		Block D-test
127	L5 43L		
	46 132L		set to punch M
128	26 93L		
	00 130L		
129	92 135F		2 C. R.
	92 2F		0

LOCATION	ORDER		NOTES PAGE 9
130	F5 125L		
	42 100L		reset 100L
131	92 135F		2 C. R.
	92 259F		letters
132	92 ( )F		type
	92 707F		nos.
133	92 135F		2 C. R.
	26 S4		* to compiler
134	LL 4095F		
	LL 4094F		
135	80 F		
	00 F		D set constant
136	80 F		
	00 F		D switch
137	N1 8F		
	L5 ( )F		(a)end test constant
138	75 6F		
	L0 ( )F		(b)end test constant
139	00 F		
	00 F		0
140	00 F		
	00 10F		
141	00 F		
	00 F		node counter
142	00 F		
	00 F		state counter
143	00 F		
	00 F		node marker
144	00 F		
	00 F		punch marker
145	13 963F		
	L5 8F		end test for punch
146	00 F		
	00 F		2^n

LOCATION	ORDER	NOTES PAGE 10
147	00 F	
	00 F	m/6
148	00 F	
	00 F	m/3
149	74 F	constants for
	40 ( )F	
150	00 F	function routine
	00 151L	
Compiler		
	00 875K	
0	0F F	
	91 4F	read in 5th hole character
1	40 F	
	L7 F	test for 5th hole delay
2	36 S3	
	41 F	clear 0
3	81 4F	
	L0 123L	-80F 001F
4	36 L	to 0FF if 0
	L4 123L	+ 80F 001F
5	50 F	
	74 97L	x 10
6	00 4F	
	91 4F	
7	32 5L	
	S1 3F	store -n in 1
8	40 1F	
	75 98L	x $(2^{-19} + 2^{-39})$
9	S5 F	
	L4 99L	+ $(\alpha 2^{-19} + \alpha 2^{-39})$
10	42 114L	α + n
	42 26L	location of 1st marker
11	42 149S3	variable end test plant
	S4 F	α + 2n

LOCATION	ORDER	NOTES PAGE 11
12	42 100L	end test for marker
	46 8S3	start of function routine
13	42 51S3	
	42 30L	first word store
14	L4 98L	$+ 2^{-19} + 2^{-39}$
	46 106L	$\alpha + 2n + 1$
15	L4 98L	$+ 2^{-19} + 2^{-39}$
	46 105L	$\alpha + 2n + 2$
16	42 104L	
	42 107L	
17	S5 F	
	L4 7L	
18	10 2F	
	50 F	
19	00 2F	
	42 115S3	store punch
20	42 120S3	and read in constant
	00 20F	in main
21	46 29S3	program
	46 30S3	
22	46 4S3	
	46 5S3	
23	L1 1F	
	42 24L	
24	F5 F	
	00 ( )F	plant 2^n
25	40 146S3	
	49 2F	
26	L5 2F	
	40 ( )F	
27	10 1F	
	40 2F	plant markers
28	F5 26L	
	42 26L	

LOCATION	ORDER	NOTES PAGE 12
29	L0 100L	
	36 26L	
30	L5 (101)L	
	40 ( )F	
31	L5 30L	
	L4 98L	plant 1st 7
32	46 30L	
	42 30L	words of fcn,
33	L0 108L	routine
	36 30L	
34	L5 30L	
	42 42L	set start
35	42 44L	
	81 4F	
36	L0 97L	- 10
	36 47L	
37	L4 97L	
	50 F	read integer
38	74 97L	
	00 4F	
39	91 4F	
	36 38L	
40	L0 112L	
	36 42L	test for prime
41	L5 109L	L5 36
	22 42L	
42	L5 110L	F1 36
	40 ( )F	store orderpair
43	00 20F	
	L5 111L	164
44	S4 116L	
	46 ( )F	store node location

LOCATION	ORDER	NOTES PAGE 13
45	F5 42L	
	42 42L	increase address
46	42 44L	
	22 35L	to next term
47	F0 F	- 1
	36 54L	test for end of function
48	50 44L	
	L5 44L	
49	F0 F	-1
	42 50L	
50	42 52L	
	L5 ( )F	
51	36 56L	
	K5 F	set 36 transfers
52	32 52L	
	42 ( )F	
53	L5 50L	
	26 49L	
54	F5 44L	
	40 2F	set end of function
55	50 2F	
	22 48L	to plant 36 at end
56	F1 2F	
	36 58L	jumps if end of function
57	L5 113L	
	22 42L	plant 36 26
58	L5 44L	
	42 61L	set end of function
59	F5 44L	
	42 44L	
60	42 42L	
	42 62L	
61	L5 114L	
	40 ( )F	store L5 7F L4(marker)

LOCATION	ORDER	NOTES PAGE 14
62	L5 115L	
	40 ( )F	store 50 F 40 7F
63	41 2F	clear 2
	L5 61L	
64	F0 F	-1
	42 65L	
65	42 68L	
	L5 ( )F	
66	L4 108L	+ 75 - -
	36 69L	
67	L4 115L	+ 50 - -
	36 70L	plant 26
68	L5 61L	transfers
	42 ( )F	
69	L5 68L	
	26 64L	
70	F5 114L	
	42 114L	marker increase
71	F5 1F	
	40 1F	count functions
72	36 73L	
	26 45L	to next function
73	F5 62L	
	42 74L	
74	L5 (116)L	
	40 ( )F	
75	L5 74L	
	L4 98L	$2^{-19}+2^{-39}$
76	46 74L	
	42 74L	plant last 4
77	L0 120L	words of function
	36 74L	routine
78	L5 44L	
	46 74L	reset initial

LOCATION	ORDER	NOTES PAGE 15
79	L5 86L	and final store
	46 30L	
80	L5 74L	
	42 1F	store a_1
81	F0 F	-1
	42 7S3	plant last word of
82	42 9S3	function routine
	L5 121L	875
83	L0 1F	
	40 2F	store 875 - a_1
84	51 2F	
	00 1F	
85	66 122L	
	10 1F	
86	S5 101L	
	40 2F	m/6 to 2
87	40 147S3	store m/6
	L4 2F	
88	40 148S3	store m/3
	L5 1F	
89	42 6S3	
	42 34S3	store a_1
90	L4 2F	+ m/6
	42 8S3	
91	42 35S3	a_1 + m/6
	L4 2F	
92	L4 2F	
	42 30S3	
93	42 46S3	store b_1
	42 53S3	
94	L4 2F	
	42 52S3	store b_1 + m/6
95	L4 2F	
	42 54S3	store b_1 + m/3
96	26 S3	to program
	00 F	waste

LOCATION	ORDER	NOTES PAGE 16
97	00 F	
	00 10F	
98	00 1F	
	00 1F	
99	00 151S3	α
	00 151S3	α
100	75 2F	
	40 ( )F	marker end test
101	L5 6F	
	40 F	
102	41 7F	
	L5 F	
103	L4 F	
	40 151S3	
104	40 F	
	F5 ( )F	1^{st} 7 words of
105	42 ( )F	
	L0 149S3	function routine
106	32 ( )F	
	L5 150S3	
107	50 F	
	42 ( )F	
108	75 108L	
	00 F	end test for 1^{st} 7 words
109	L5 F	plant
	36 F	z_i
110	F1 F	
	36 F	$\bar{z}_i$
111	00 150S3	α - 1
	00 F	
112	00 F	
	00 13F	
113	36 F	
	26 F	end of term

LOCATION	ORDER	NOTES PAGE 17
114	L5 7F	
	L4 ( )F	end of function
115	50 F	
	40 7F	
116	50 6F	
	S5 F	
117	J0 7F	
	S0 F	last 4 words of
118	L4 7F	
	S0 F	function routine
119	40 7F	
	22 F	
120	75 120L	
	00 F	end test last 4 words
121	00 F	
	00 1S5	memory constant
122	00 F	
	00 6F	
123	80 F	
	00 1F	
	00 161K	
Library Routine X - 7 Sum Check		
	20 875N	