A

# MICROPROCESSOR BASED DIGITAL LOGIC SIMULATOR

BY

KEVIN DRESHER

A THESIS

PRESENTED IN PARTIAL FUFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE

OF

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

AT

NEW JERSEY INSTITUTE OF TECHNOLOGY



Newark, New Jersey

1980

APPROVAL OF THESIS

A

MICROPROCESSOR BASED

DIGITAL LOGIC SIMULATOR

BY

KEVIN DRESHER

FOR

DEPARTMENT OF ELECTRICAL ENGINEERING

NEW JERSEY INSTITUTE OF TECHNOLOGY

BY

FACULTY COMMITTEE

APPROVED: _____

_____

_____

Newark, New Jersey

1980

AN ABSTRACT

A

MICORPROCESSOR BASED

DIGITAL LOGIC SIMULATOR

by

Kevin Dresher

Advisor: Dr. Robert DeLucia

Submitted in Partial Fulfillment of the Requirements for

The Degree of Master of Science in Electrical Engineering

July 1980

It is the intent of this thesis to acquaint the reader with a tool which is available for use in the digital circuit design field. The reader is now able to totally simulate via DLS the digital logic design he creates on paper before it ever takes a hardware form. The computer program accepts a detailed description of the schematic and creates timing diagrams, loading statistics, cross references, and various lists for future documentation.

The user needs no programming knowledge and will find the requirements to run a simulation with DLS extremely user oriented. The simulation descriptions and command language are tailored to logic design applications. The format is straight forward, utilizing standard English

language and logic design concepts. To code a design for simulation the designer needs only a well labeled circuit diagram, where all the inputs and outputs of each element has a label With the addition of a few simulation parameters DLS will take the network description and form a program in memory which will recreate the operations of the digital circuit.

## Dedication

I would like to thank the people that are and were close to me for threatening me with bodily injury if I did not complete this work.

TABLE OF CONTENTS

## LIST OF FIGURES

# CHAPTER 1

# WHY ANOTHER LOGIC SIMULATOR?

## 1.1 Need for Simulators

The use of computers to assist in the engineering of digital systems is not a new idea. Design automation schemes have been in existence since the first generation computers. The original computer systems were mainly concerned with production logistics such as generating wiring schedules and printed circuit board layouts. The logic design phase was performed manually, using intuition and experience based on the theories of switching circuits. When the MSI and LSI logic components were introduced, the design approach changed radically. The problem was one of sheer complexity. Since digital systems attained such a high level of sophistication, the old conventional design practices proved inadequate to handle these complexities. It therefore became essential to use the computer from the initial design stages.

This is done through the use of the process of simulation, whereby it is possible to model the behavior of a real system either mathematically or functionally.

Experience shows that simulation is one of the most powerful analysis tools available to the designer. It allows the designer to make expermental designs with systems, real or proposed, where it would otherwise be impossible or impractical to do so.

Computer-Aided Design (CAD) programs were written for the purpose of simulating proposed or experimental systems. Using CAD programs, the designer could explore new ideas and techniques. As results are achieved more rapidly, inoperative designs may be eliminated immediately while positive results are open to exploration.

## 1.2 Levels of Simulation

There are four basic levels at which digital systems can be simulated.[1] The first is known as "System Level," whereby the simulation is used to evaluate the general overall properties of a system. Elements of the system are usually complex devices, and may include buffers, memory modules, arithmetic units, and central processing units. Usually each model is characterized by a set of parameters, such as response time and capacity. System level simulation is primarly used as a means of predicting system performances.

This is followed by the type of simulation known as

---

[1] M. A. Breuer, "Recent Developments in Design Automation," Computer, May/June 1972, pp. 23-35

"Register Transfer Level." At this level data flow is specified at the register level. The simulator operates upon real data, hence the functional design of the system can be evaluated.

The third type of simulation is "Gate Level Simulation." At this level the system is described by a collection of logic gates and their interconnections. Each signal line is restricted primarly to two or three values. Time is usually quantized to the point where one unit of time corresponds to one gate delay time unit.

The final type of simulation is the "Circuit Level." A logic gate circuit may consist of some interconnection of diodes, transistors, and resistors. Here each signal line is not restricted to just two or three values but rather to a quantized interval between two voltages or current levels. In addition time is quantized to a very fine degree. Transitory behavior is usually of primary interest.

Each of the last three levels employs models which are simplifications of those of the preceding level, both in quantitative terms and in terms of behavior. The set of components represented in the circuit level model of a logic gate and the circuit's finite rate of change of state, may be simplified using a gate level model into a single two state element. The state of this element would change instantaneously at discrete time intervals. Simil-

arly sets of gates may be merged together to form elements of a register transfer level model, in which state changes may occur at varing multiples of the basic gate operation time units. Circuit, gate, and register transfer level simulation models represent progressive levels of simplification of an actual system element behavior. This can be viewed as being derived from a direct translation of its electrical characteristics.

A system level simulation model represents a level of simplification of elements of a real system derived by abstraction, rather then by synthesis. Circuit level simulation employs continous time models. This differs fundamentally from those using gate level or register transfer level which employs discrete time models.

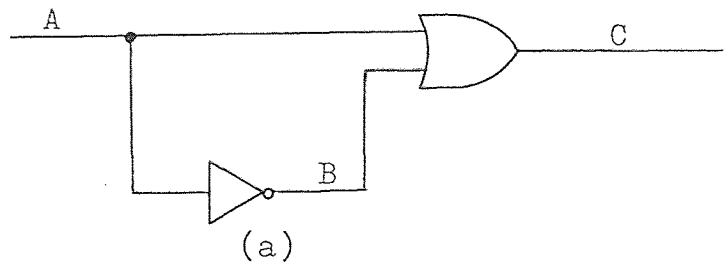
## 1.3 Gate Level Simulation

Digital Logic Simulator (DLS) is a gate level simulation program which can be used for analyzing digital logic designs. When given the initial state and the input sequence the simulator will calculate a state-time map of the logic signals.

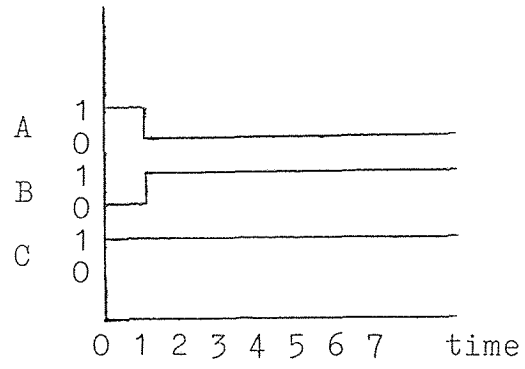
Most of the early simulators would model gates as elements having zero induced propagation delay time.[2] This

[2] M. J. Flomenhoft and B. M. Csencsits, "A Minicomputer Based Logic Circuit Fault Simulator," ASM Sigma Newsletter, Vol. 4, No. 3, 1974, pp. 15-19

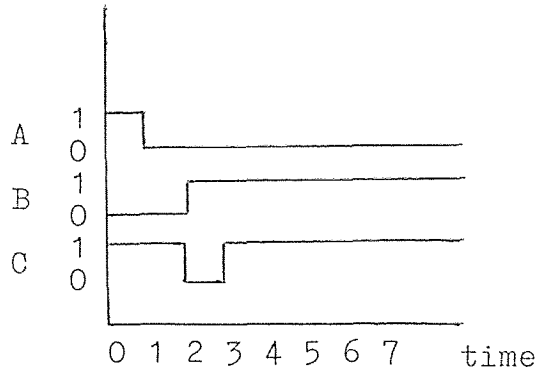
Time Delay Modeling



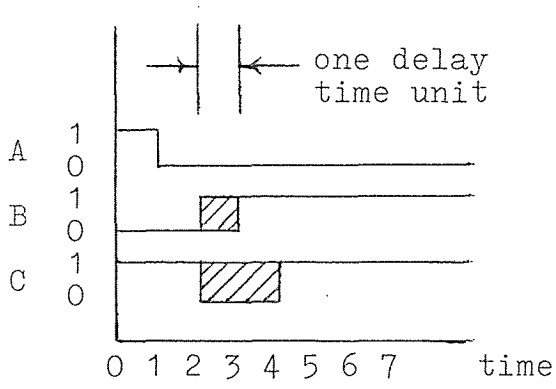

(a)




(b)




(c)




(d)

FIGURE 1-1

implies that the output logic level changes instanteously when the inputs change. An example is shown in Figure 1-1a which depicts a two gate circuit. In a zero delay simulator as the input signal (A) changes from a logic '1' to a logic '0,' the output signal (C) stays constant. This can be seen in Figure 1-1b.

In actuality, this circuit design would have an inherent race condition. One of the two signals being fed into the OR gate will have a propagation delay time longer than the other.

One of the goals for creating DLS was to develop a method of simulation where such hazards could be observed and corrected. DLS has two modes of operation which can show the presence of a race condition. In the first mode, each gate has a single time unit delay before the output changes corresponding to changes of the inputs. Figure 1-1c shows that when the input to the NOT gate changes from a logic '1' to a logic '0' the output signal (B) of the NOT gate is delayed for one time unit before it changes from a logic '0' to a logic '1.' This means that for one time unit both inputs to the OR gate will be at a logic '0' producing a logic '0' on the output. In the next time frame the NOT gate has propagated its signal through the gate producing a logic '1' on one of the inputs of the OR gate which produces a logic '1' on the output.

There is a difference between the simulation of a zero

and a one gate delay circuit simulation. The first simulation had a constant logic '1' on the output where the latter one had a period of time where the output dropped to a logic '0.' In digital circuit design this would be known as a glitch. Using the simulator the designer would be able to see the existence of this hazardous condition and go back to modify the circuit to remove the glitch from the design.

The second mode of DLS uses what is known as a three value simulator.[3] Whenever a signal tries to change its logic level, it enters a transition state. This is a third logic state where the state is neither a logic '1' or a logic '0,' it is unknown. Figure 1-1d shows that when the output of the NOT gate tries to change its logic level, it enters the transition state for one time unit. In the next time frame the output goes to the correct logic level. The transition state that the NOT gate produced is passed to the OR gate which produces an unknown output. The output of the OR gate will have two transition states due to the fact that in time frame two both inputs were at a logic '0.' As the output attempts to reach a logic '0' it is forced into the transition state for one time unit. In the third time frame one of the inputs is in the transition state which keeps the output in the transition state, the glitch.

---

[3]J. S. Jephson, R. P. McQuarrie, and R. E. Vogelsberg, "A Three-Value Computer Design Verification System," IBM System Journal, Vol. 8, No. 3, 1969, pp. 178-189

Finally by the fifth time frame all the signals have settled out. When the results are viewed the fact would be noted that the final output had two time units in which the output is unknown. This occurrence creats a condition that is in all probability hazardous to the operation.

## 1.4 DLS a Microprocessor Based Program

One of the big differences between DLS and other simulators is that it has been implemented on a microprocessor based computer system. Most standard high-level languages, such as Fortran and Basic, are oriented to numerical computations and consequently are extremely inefficient when used for data processing operations. A more efficient approach is achieved through the use of a machine dictated assembly language. Data is usually stored in a tabular or list format. Thus a language capable of setting up data structures in list form that is capable of manipulating the items in the list is required.

DLS was written in assembly language for two reasons. The first is for its ease of handling list structured queues and secondly high-level languages, require large amounts of memory. One of the objectives for writing DLS was to create a system that occupied the smallest amount of memory space, making it possible to run on a small system. Even though assembly languages have the disadvant-

age of being specific to one type of computer, DLS was written for the 8080 microprocessor, an industry standard.

# CHAPTER 2

# THREE VALUE SIMULATION

## 2.1 Use of Ternary Algebra

The presence of hazards and races in combinational logic circuits may be detected by using the concept of ternary algebra.[1] In this method a third value 'X' which assumes the value between a logic '0' and a logic '1' is used to represent unspecified transition periods, initial conditions, oscillations, and don't know states. Basic logic gates can be redefined in terms of ternary functions using logic levels '0,' '1,' and 'X.' Figure 2-1 shows the truth tables for the basic gates for both two and three logic state simulations.

The using of the three value method allows hazards to be detected that normally go unnoticed in a two value simulation.[2] Figure 2-2a shows the two value simulation for several gates. When the two inputs change simultan-

---

[1] M. Yoeli and S. Rinon, "Application of Ternary Algebra to the study of Static Hazards," Journal of the Association for Computing Machinery, Vol.11, 1964, pp.84-97

[2] J.S. Jephson, R. P. McQuarrie, and R. E. Vogelsberg "A Three-Value Computer Design Verification System," IBM System Journal, Vol.8, No.3, 1969, pp.178-189

Two Value Truth Table

| | IN 1 | IN 2 | AND | NAND | OR | NOR | EXOR |
|---|---|---|---|---|---|---|---|
| ØØ: | Ø | Ø | Ø | 1 | Ø | 1 | Ø |
| Ø1: | Ø | 1 | Ø | 1 | 1 | Ø | 1 |
| Ø2: | 1 | Ø | Ø | 1 | 1 | Ø | 1 |
| Ø3: | 1 | 1 | 1 | Ø | 1 | Ø | Ø |

(a)

Three Value Truth Table

| | IN 1 | IN 2 | AND | NAND | OR | NOR | EXOR |
|---|---|---|---|---|---|---|---|
| ØØ: | Ø | Ø | Ø | 1 | Ø | 1 | Ø |
| Ø1: | Ø | 1 | Ø | 1 | 1 | Ø | 1 |
| Ø2: | Ø | X | Ø | 1 | X | X | X |
| Ø3: | 1 | Ø | Ø | 1 | 1 | Ø | 1 |
| Ø4: | 1 | 1 | 1 | Ø | 1 | Ø | Ø |
| Ø5: | 1 | X | X | X | 1 | Ø | X |
| Ø6: | X | Ø | Ø | 1 | X | X | X |
| Ø7: | X | 1 | X | X | 1 | Ø | X |
| Ø8: | X | X | X | X | X | X | X |

(b)

Figure 2-1

## Combinational Hazard Detection

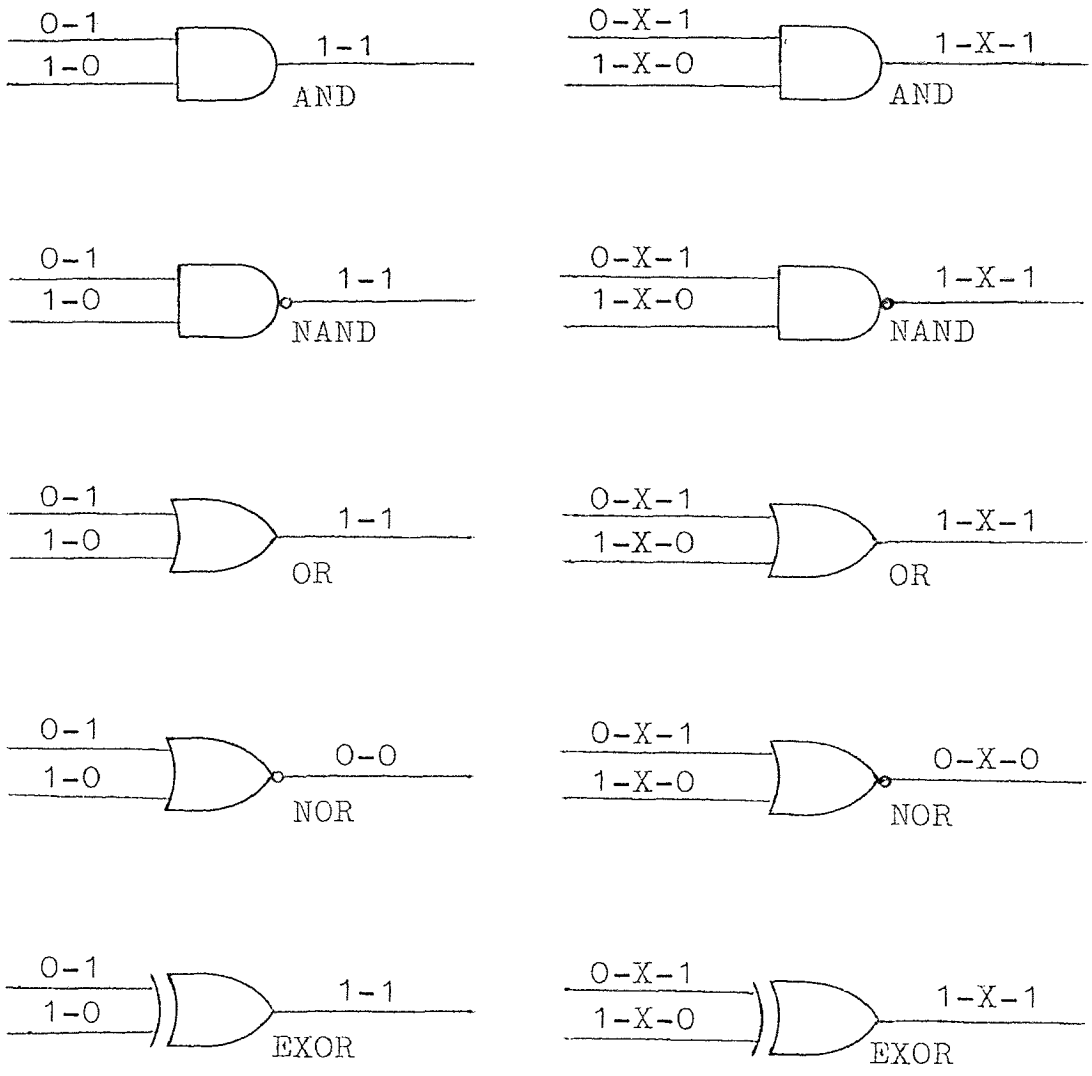



Figure 2-2

eously the output stays constant. In three value simulation when a logic level changes state first it must enter the logic 'X' state. Figure 2-2b shows that when both inputs to a gate change at the same time, for one time unit both inputs are unknown. This produces an output which is temporarily unknown. In a larger circuit design this glitch would be passed along to the rest of the circuit which could lead to a possible erroneous final output.

In addition to hazard detection the third logic level may also be used to represent "don't care" input conditions to the circuit. This makes it possible to cut down on the amount of test data required to check a given circuit. For example if it were required to simulate the reset logic of a basic register circuit. Normally this would have to be performed by applying the reset logic to the input repetitively and checking that for every possible combination of input bits the output of the register always goes to a logic '0.' This would require $2^n$ simulation runs, where n is the number of bits in the register. By initally setting all of the bits in the register to the logic 'X' state and then simulating the reset logic, it is possible to determine in one simulation run those stages which do not get reset to a logic '0' state.[2]

---

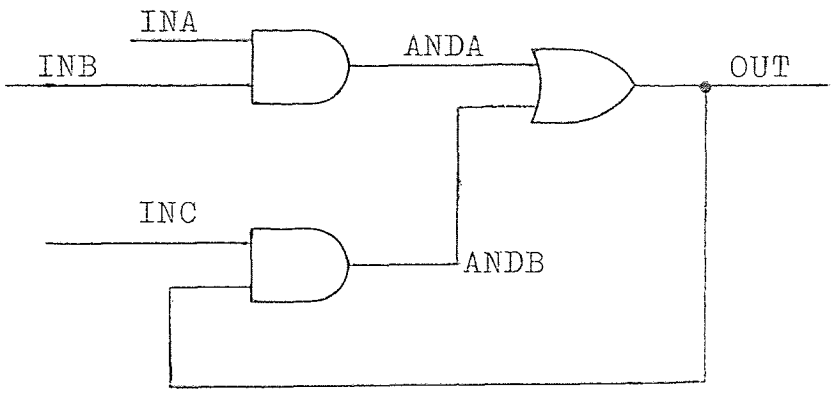
[2]Ibid., pp.179

## 2.2 Propagation Hazard Example

Figure 2-3a is a logic circuit which was simulated by DLS. The circuit consists of two AND gates and one OR gate. The output of the OR gate is fed back to one of the AND gates to form a type of latch. Figure 2-3b is the print-out of the DLS simulation operated in the normal mode. Time frame 0 shows that when the three inputs are unknown the output is unknown. In time frames 1, 2, 3, 4, and 5 the circuit is put through several different test patterns. A problem occurs when the inputs (INA and INB) change their values from time frame 5 to frame 6. This simultaneous change is detected as a possible hazard to the circuit. Due to the creation of the feedback path in the circuit, the glitch is transferred through the OR gate and then back to one of the inputs. This means that the glitch causes the circuit to settle in the unknown state.

As a verification of the results DLS is rerun using the trace mode this time. Figure 2-3c is the DLS trace mode results. The critical point is time frame 6 where the two inputs change simultaneously. INA changes from a logic '1' to a logic 'X' then to the final logic '0' value. On the other hand INB changes from a logic '0' to a logic 'X' and settles to a logic '1.' For one time unit both inputs to the AND gate are unknown. This glitch is fed into the OR gate which will produce a logic 'X' which feeds

## Digital Latch With Hazard Example




(a)

| | INA | INB | INC | ANDA | ANDB | OUT |
|---|---|---|---|---|---|---|
| ØØ: | X | X | X | X | X | X |
| Ø1: | X | Ø | Ø | Ø | Ø | Ø |
| Ø2: | 1 | 1 | 1 | 1 | 1 | 1 |
| Ø3: | 1 | Ø | 1 | Ø | 1 | 1 |
| Ø4: | 1 | Ø | Ø | Ø | Ø | Ø |
| Ø5: | 1 | Ø | 1 | Ø | Ø | Ø |
| Ø6: | Ø | 1 | 1 | Ø | X | X |

(b)

| | INA | INB | INC | ANDA | ANDB | OUT |
|---|---|---|---|---|---|---|
| ØØ: | X | X | X | X | X | X |
| ØØ: | X | X | X | X | X | X |
| Ø1: | X | X | X | X | X | X |
| Ø1: | X | Ø | Ø | X | X | X |
| Ø1: | X | Ø | Ø | Ø | Ø | X |
| Ø1: | X | Ø | Ø | Ø | Ø | Ø |
| Ø1: | X | Ø | Ø | Ø | Ø | Ø |
| Ø2: | X | Ø | Ø | Ø | Ø | Ø |
| Ø2: | 1 | X | X | Ø | Ø | Ø |
| Ø2: | 1 | 1 | 1 | X | Ø | Ø |
| Ø2: | 1 | 1 | 1 | 1 | Ø | X |
| Ø2: | 1 | 1 | 1 | 1 | X | 1 |
| Ø2: | 1 | 1 | 1 | 1 | 1 | 1 |
| Ø2: | 1 | 1 | 1 | 1 | 1 | 1 |
| Ø3: | 1 | 1 | 1 | 1 | 1 | 1 |
| Ø3: | 1 | X | 1 | 1 | 1 | 1 |
| Ø3: | 1 | Ø | 1 | X | 1 | 1 |
| Ø3: | 1 | Ø | 1 | Ø | 1 | 1 |
| Ø3: | 1 | Ø | 1 | Ø | 1 | 1 |
| Ø4: | 1 | Ø | 1 | Ø | 1 | 1 |
| Ø4: | 1 | Ø | X | Ø | 1 | 1 |
| Ø4: | 1 | Ø | Ø | Ø | X | 1 |
| Ø4: | 1 | Ø | Ø | Ø | Ø | X |
| Ø4: | 1 | Ø | Ø | Ø | Ø | Ø |
| Ø4: | 1 | Ø | Ø | Ø | Ø | Ø |
| Ø5: | 1 | Ø | Ø | Ø | Ø | Ø |
| Ø5: | 1 | Ø | X | Ø | Ø | Ø |
| Ø5: | 1 | Ø | 1 | Ø | Ø | Ø |
| Ø5: | 1 | Ø | 1 | Ø | Ø | Ø |
| Ø6: | 1 | Ø | 1 | Ø | Ø | Ø |
| Ø6: | X | X | 1 | Ø | Ø | Ø |
| Ø6: | Ø | 1 | 1 | X | Ø | Ø |
| Ø6: | Ø | 1 | 1 | Ø | X | X |
| Ø6: | Ø | 1 | 1 | Ø | X | X |

(c)

Figure 2-3

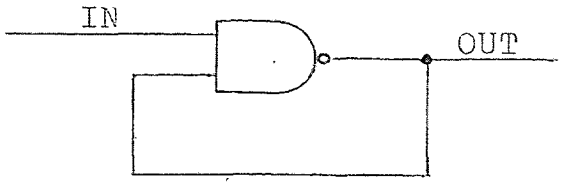
this value back to the AND gate which will the produce an output of a logic 'X.' Even though the first AND gate has by this time finished changing, the original glitch has caused the output of the circuit to become latched in the unknown state.

## 2.3 Oscillation Error Example

A simple example of an oscillating circuit is expressed in Figure 2-4a. This simple NAND gate has a problem when the input goes to a logic '1,' the output tries to go to a logic '0.' This is then fed back to the other input. Now what happenes is that the output tries to go to the logic '1' state. This circuit works fine with a logic '0' on the input but whenever it goes to any other logic value the output can not find a stable state so it oscillates.

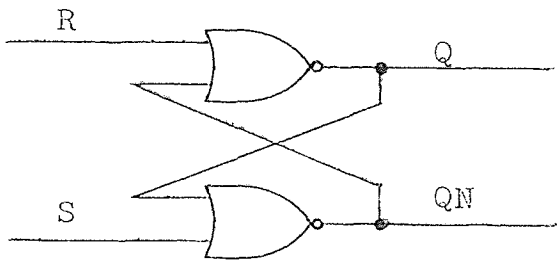
Another example is shown in Figure 2-4b. The two NOR gates are configured to form a R-S Flip Flop. Note from the results that when no initial condition is given and both inputs are at a logic '0' the output stays unknown. This is due to the fact that DLS assigns a logic 'X' to all gates prior to the start of the simulation. This circuit operates properly up to time frame 7. Here both inputs (R and S) go to a logic '1' producing outputs (Q and QN) at a logic '0.' The outputs are stable except by definition one is supposed to be the complement of the other.

## Oscillating Test Circuits




(a)

| | IN | OUT |
|---|---|---|
| ØØ: | Ø | 1 |
| Ø1: | Ø | 1 |
| Ø2: | 1 | X |
| Ø3: | 1 | X |
| Ø4: | Ø | 1 |
| Ø5: | Ø | 1 |
| Ø6: | X | X |
| Ø7: | X | X |
| Ø8: | Ø | 1 |
| Ø9: | Ø | 1 |
| 1Ø: | 1 | X |




| | R | S | Q | QN |
|---|---|---|---|---|
| ØØ: | Ø | Ø | X | X |
| Ø1: | Ø | Ø | X | X |
| Ø2: | 1 | Ø | Ø | 1 |
| Ø3: | X | Ø | Ø | 1 |
| Ø4: | Ø | Ø | Ø | 1 |
| Ø5: | Ø | 1 | 1 | Ø |
| Ø6: | Ø | Ø | 1 | Ø |
| Ø7: | 1 | 1 | Ø | Ø |
| Ø8: | Ø | Ø | X | X |
| Ø9: | 1 | Ø | Ø | 1 |
| 1Ø: | Ø | X | X | X |

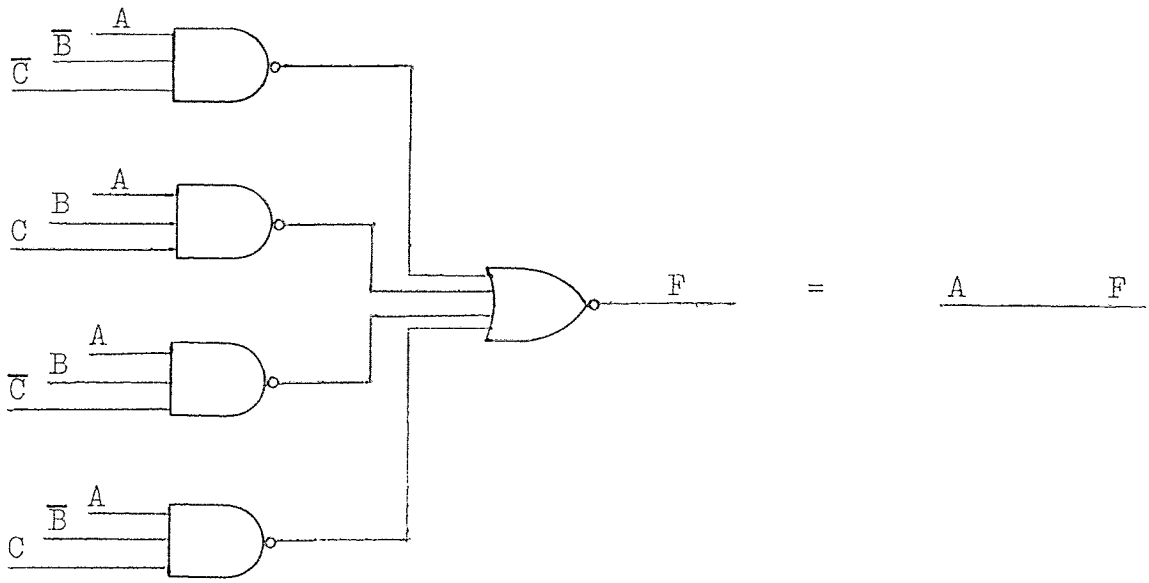
Figure 2-4

The problem occurs in this circuit when both inputs now drop from a logic '1' to a logic '0' at the same time. The circuit starts to oscillate which DLS detects in time frame 8.

## 2.4 Don't Care Example

Figure 2-5 shows a circuit derived from the equation $F=A\overline{B}\overline{C}+A\overline{B}C+AB\overline{C}+ABC$, which using Boolean Algebra can be reduced to $F=A$ . To prove this, first DLS is made to run through the nine different possible input combinations. The problem is then rerun, this time setting the values of the eliminated variables to the logic 'X' state. The two simulations produce the identical results. This example was not chosen to show reduction techniques but to show that the logic 'X' state could be used in place of don't care situations which may arise.

## Don't Care Example




| | A | B | C | F |
|---|---|---|---|---|
| ØØ: | Ø | Ø | Ø | Ø |
| Ø1: | Ø | Ø | 1 | Ø |
| Ø2: | Ø | 1 | Ø | Ø |
| Ø3: | Ø | 1 | 1 | Ø |
| Ø4: | 1 | Ø | Ø | 1 |
| Ø5: | 1 | Ø | 1 | 1 |
| Ø6: | 1 | 1 | Ø | 1 |
| Ø7: | 1 | 1 | 1 | 1 |

| | A | B | C | F |
|---|---|---|---|---|
| ØØ: | Ø | X | X | Ø |
| Ø1: | Ø | X | X | Ø |
| Ø2: | Ø | X | X | Ø |
| Ø3: | Ø | X | X | Ø |
| Ø4: | 1 | X | X | 1 |
| Ø5: | 1 | X | X | 1 |
| Ø6: | 1 | X | X | 1 |
| Ø7: | 1 | X | X | 1 |

Figure 2-5

# CHAPTER 3

# TABLE DRIVEN SIMULATION TECHNIQUES

## 3.1 Modeling Approach

A fundamental question is how a digital circuit is to represented or modeled by the computer. There are several ways to model a circuit, each have advantages and disadvantages. The method of digital circuit modeling is dependent upon the type of machine being used. Three important factors which must be considered are machine type, word length, and the number or language of the instruction set.

The simulation model is formed from the inputed source language statements which describe the digital circuit. These statements can either be interpreted directly and then executed or compiled into machine code which is executed later. Most of the earlier simulators were either interpretive or executed compiled code.[1] Current simulators however, employ some form of data structure and are table driven.

For compiled code simulators each source statement

---

[1] M. A. Breuer, Digital System Design Automation, California, Computer Science Press, Inc., 1975, pp. 237-242

generates a set of subroutines which perform the logical function required by each specific element. The simulated network is represented in the computer as a series of interconnected subroutines which evaluates the logical function of each element in the order in which they appear in the circuit. Starting at the input gates and proceding through the circuit, outputs of one gate acting as inputs to the succeeding gates until the final output gates are reached. The disadvantage of this approach is that for each element there could be about five to ten instructions required to perform the simulation. For a fairly large circuit the size of the compiled code would require a fair amount of memory. Another problem is that a compiled code is inherently a zero delay simulation and is extremely inflexible as to the extent of the types of different operations which can be performed during simulation.

## 3.2 Table Driven Simulation Method

In the table driven method, the parameters of each logic element in a circuit is stored in a tabular form.[2] Each entry consists of such data as logic function, propagation delay, input sources, output values, and output destination. The source language statements are translated

---

[2] M. A. Breuer, Design Automation of Digital Systems, New Jersey, Prentice-Hall, Inc., 1972, pp. 127-128

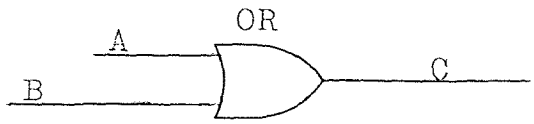
into a data structure representing the circuit. During simulation the data structure is operated on by a control program which analyzes the information in the lists in accordance with the simulator command statements to determine the flow of data and logical values in the network.

The interpreter program operates by evaluating all the elements and assessing those subroutines which are required by the program rather than having individual macros for each element. When a large circuit design is simulated the running time of the simulator could become a factor because of the sequential nature of the program and the number of instructions to be executed. In a table driven simulator for a given input pattern only a certain number of the elements will be changing their logic states. A large reduction in computation time is achieved in DLS because only those elements which are supposed to change states are evaluated.

## 3.3 Dual Table Simulation

DLS contains seven tables but the heart of the program is contingent upon two of the tables. These two tables are known as T1 and T2, contain all the logic levels of the network. Each logic level is stored in one word of memory, in the case of the 8080 microprocessor a word of memory is 8 bits in length. At the beginning of the simulation run

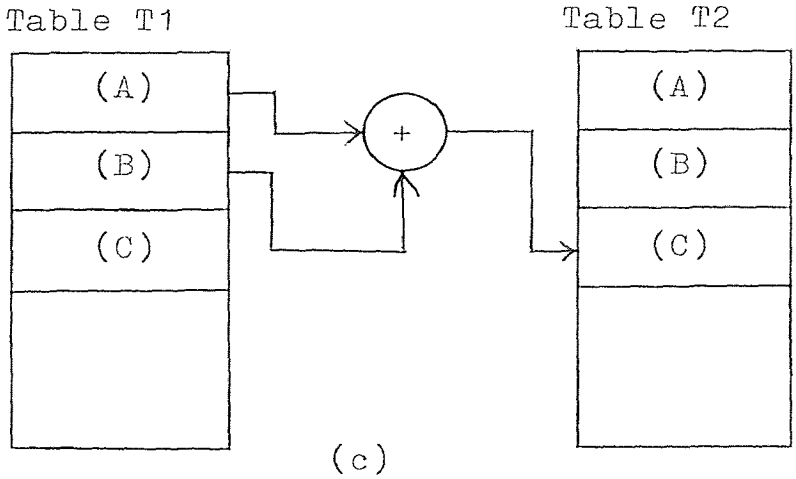
## Dual Table Operation




(a)

.OR/2. A,B,C IC=Ø

(b)




(c)

Figure 3-1

both T1 and T2 contain the same information. If no initial condition is given for each element a logic 'X' is automatically assigned to the output of that element.

The simulation is done by taking the inputs from T1, performing the logic function called for and storing the results in T2. For example, Figure 3-1a shows a single two input one output OR gate. In DLS a line of source code to describe the gate is shown by Figure 3-1b. The line tells the interpreter program the type of logic gate, the number of inputs, the input symbols, the output symbol, and any initial condition for the output symbol. The program would translate this line code and assign three words of memory for T1 and T2, for this one element. Each table would have the same logic levels assigned to them at the beginning of the simulation. During the simulation the two input values would be taken from T1, operated upon and stored in the output, located in T2, as can be seen in Figure 3-1c. At this point a comparison is made between the contents in T1 and T2. If the two tables contain the identical information then the simulated circuit is said to have reached a stable state. Disagreement indicates that some of the signals are still being propagated through the circuit.

If only one table existed there would be no way to ascertain whether the network had reached a stable state, since there would be no record of the previous state. Two

tables make it possible to check the stability of the circuit. After all logical operations were performed T1 would contain the n-th state while T2 would contain the n+1 state. When comparing the n-th and n+1 states of the network it can be determined if the network had achieved a stable state.

A clarification of this analysis may be seen in the example shown by Figure 3-2, which is a simulation run of Figure 3-1a. Assume that both inputs (A and B) are at a logic '0' and the initial condition of the output (C) is also at a logic '0.' Figure 3-2a shows that at the start of the simulation both T1 and T2 contain the same data. Assume now that one of the inputs (A) is going to change to a logic '1,' but in a three value simulation it must for one time unit be at the transition level 'X.' The 'X' value is substituted into the (A) location in T1 and T2, then the OR operation is performed as seen in Figure 3-2b. A comparison is made between T1 and T2. Since they are not the same the operation is not yet complete, so T2 is copied over into T1. The n+1 state now becomes the n-th state and a new n+1 state must be generated. Now that the input (A) has been in the transition state for the required time it now goes to a logic '1.' Another OR operation is performed as can be seen in Figure 3-2c. Again after the operation T2 is not equal to T1 so it is copied into T1 and again another OR operation is done. This time T1 is

## OR Gate Simulation

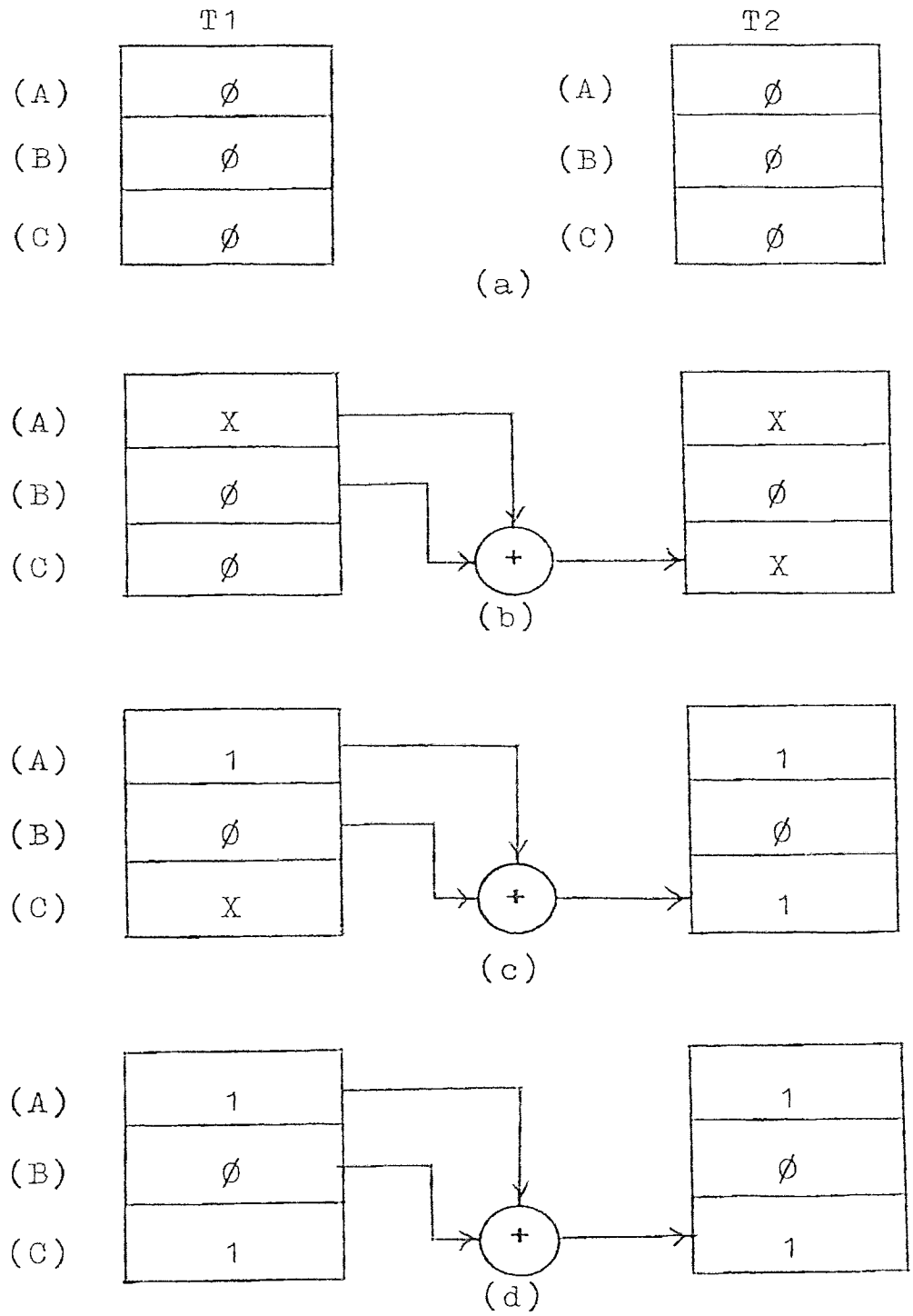



Figure 3-2

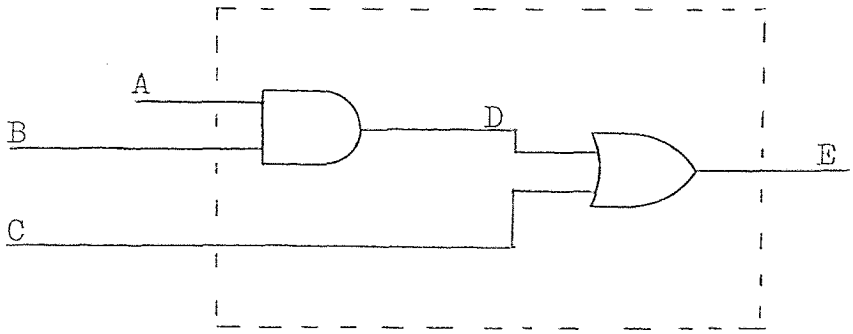
the same as T2 so the simulation update cycle is complete, all signals have been propagated through and stability in the circuit has been achieved. Using three value simulation it took two time units to produce the correct output, but it took three time units for the circuit to be considered stable in DLS.

## 3.4 Table Setups

It is the formation of the other five tables which the translator portion of DLS uses to setup the dual simulation tables. Certain information has to be extracted from the source program and broken down into the different tables. Consider Figure 3-3a which is a two element device. The enclosed area shows the portion of the circuit which will be under test. The lines extending from this area are the test inputs and the test output. Other internal signals can be monitored where applicable. To simulate this circuit using DLS the device is described by English language type statements, shown in Figure 3-3b. The program must be given the test inputs, test output, gate type, and any initial conditions.

The first thing DLS does is to scan for all symbols used in the circuit description. Figure 3-3c shows the creation of the symbol table. Each symbol, which can be up to five characters in length, is stored in the symbol

DLS Table Breakdown




(a)

```
.INPUT.   A,B,C
.AND/2.   A,B,D     IC=Ø
.OR/2.    D,C,E     IC=1
.PRINT.   A,B,C,E
```

(b)

Symbol Table

symbol

| | | |
|---|---|---|
| A | (A) | (A)' |
| B | (B) | (B)' |
| C | (C) | (C)' |
| D | (D) | (D)' |

(c)

Input Table

| |
|---|
| (A) |
| (B) |
| (C) |

(d)

( )-- memory location in table T1

( )'-- memory location in table T2

Figure 3-3

table along with its corresponding address as seen in tables T1 and T2. The symbol table is very important since all the other tables will access it to determine the locations of the symbols in table T1 and T2.

DLS then looks for certain control words for the formation of the test input table. Once DLS finds the control word, it then scans the rest of the line for symbols whose addresses can be found in the symbol table. DLS completes the operation by storing the input symbol addresses in the input table. In addition a count of the number of test inputs is maintained as shown in Figure 3-3d.

The same procedure is done in determining what points of the circuit the user wants to monitor during simulation. In this case DLS will scan for the print control word. Addresses are extracted from the symbol table and stored in the output table along with the count on the number of outputs, as seen in Figure 3-4a. For both the input and output tables, the addresses assigned are those corresponding to table T1. Since after a simulated network has reached a stable state T1 will contain the same information as T2, there would be no need to access information from T2.

The next two tables to be formed are created simultaneously. DLS scans the program looking for the logic gates. When a gate is found that gate type count will be incremented (Figure 3-4b) and then DLS will create an updating sequence table (Figure 3-4c). The update sequence for any two input

## Simulation Tables

Output Table

| |
|---|
| (A) |
| (B) |
| (C) |
| (E) |

(a)

Gate Type Table

| Gate Type | # |
|---|---|
| AND/2 | 1 |
| OR/2 | 1 |
| NAND/2 | 0 |

(b)

Update Sequence

| |
|---|
| (A) |
| (B) |
| (D)' |
| (D) |
| (C) |
| (E)' |

(c)

T1

| | |
|---|---|
| (A) | X |
| (B) | X |
| (C) | X |
| (D) | ∅ |
| (E) | 1 |

(d)

T2

| | |
|---|---|
| (A)' | X |
| (B)' | X |
| (C)' | X |
| (D)' | ∅ |
| (E)' | 1 |

(e)

( )-- memory location in T1 table

( )'-- memory location in T2 table

Figure 3-4

input device would consist of the two inputs to the gate whose addresses are located in table T1, followed by the output, whose address is located in table T2. For logic elements with four inputs and one output, the update sequence table would contain four addresses from T1 and one from T2. It should be noted that prior to simulation all symbols which were not given any initial condition are assigned a logic 'X' to their respective locations. Symbols with assigned initial conditions are inserted in both tables T1 and T2 prior to simulation.

# CHAPTER 4

# THE DLS PROGRAM

## 4.1 DLS Program Structure

The DLS simulator was written in a format kmown as a modular program. There are three distinct modules; controller/editor, compiler, and executor. Each module acts independent of each other but can not operate without the others. Parameters are not passed back and forth between modules but instead the controller will partition off blocks of memory where all the necessary information will reside. These blocks of data or tables have no fixed memory addresses. Also each table does not have any fixed size. Figure 4-1 shows how the memory would be allocated for a given simulation. The object file of DLS occupies the first 4K block of memory. The control program then partitions off the rest for the tables.

The source program which is the topological description in the DLS language is entered into the memory via the editor. As each line of data is taken in and stored in memory, the size of the source program increases. The control program will then alter where the next open source

## Memory Allocation

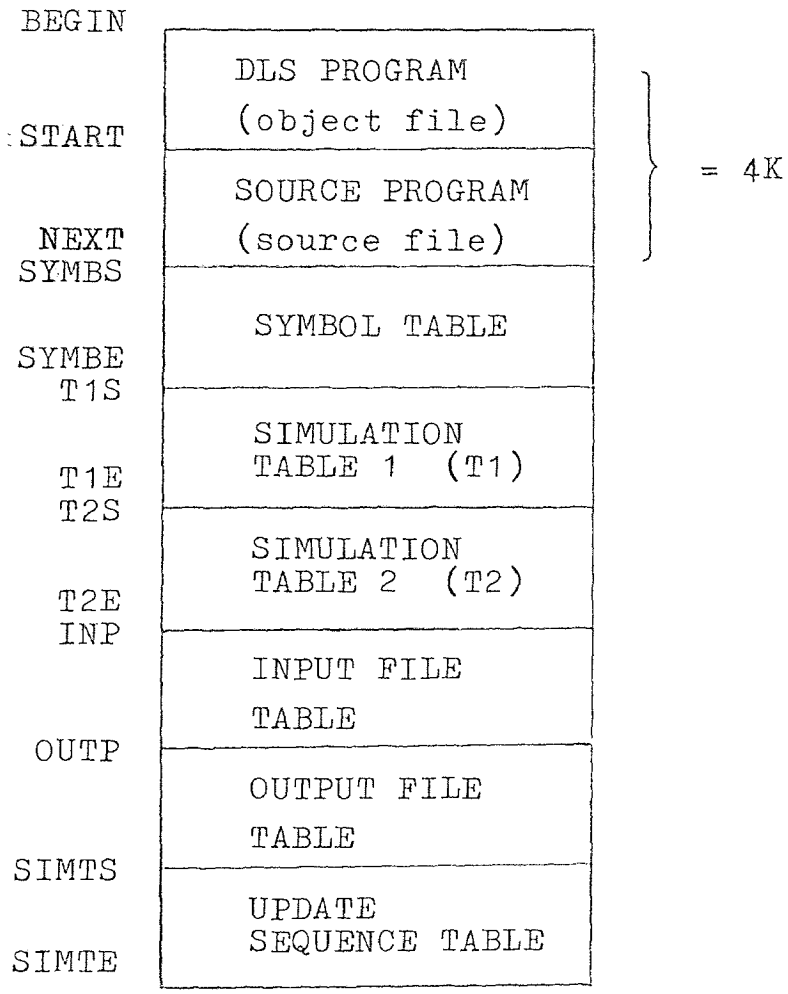



Figure 4-1

program location will be located in memory. If there are alterations in the source program any previously compiled network becomes void. This is because when the source program increases or decreases in size the other table addresses will not be altered, meaning source code information may overlap into the table area.

Once the network description is complete the compiler module will be called upon. The compiler takes the source program and breaks it down into the representing data structure. Once the compiler sets up the tables it is the function of the execution module to perform the simulation. The executer contains a simulation controller which calls upon the user to setup certain simulation parameters. Using these parameters plus the compiled tables the network can now be simulated.

## 4.2 Source Program Requirements

It is possible to define logic circuits in terms of Boolean equations but impractical for large complex circuits. To reflect the implemented configuration the equations would have to be derived directly from the actual circuit. Such an approach would be rather cumbersome. A better way would be based on an element description.[1] Each element

---

[1] H. J. Kahn and J. W. R. May, "The Use of Logic simulation in the Design of a Large Computer System," The Radio and Electronic Engineer, Vol. 43, No. 8 pp. 497-503

would have its inputs and output uniquely defined, making it easer to define complex compound modules. An element would consist of gate type, number of inputs, and the output. DLS uses this along with another parameter, the initial condition. This helps eliminate transients which would exist when the simulation first begins, since all logic elements which are not given an initial condition are put into the logic 'X' state.

DLS is slightly limited in the types of elements it can presently simulate. Figure 4-2a and 4-2b show the types of elements which DLS can handle. That which is in capitalized letters must be typed by the user, the lower case letters are where the user would put variable names, which can be up to five alphabetic characters in length. The initial condition is optional to the user and can be completely left out.

It is the users responsibility to inform DLS, within the source program, which logic variables are primary inputs and which are monitored outputs. A primary input is a variable whose logic level is not generated internally in the circuit but rather must be supplied externally by the user. They can be considered the test input paths. The monitored output points are those variables which the user wants to view during the simulation. The format for these operations is shown in Figure 4-2c.

The final requirement for DLS to operate is that the

## Command Word Format

```
.AND/2.   in1,in2,out      IC=__
.NAND/2.  in1,in2,out      IC=__
.OR/2.    in1,in2,out      IC=__
.NOR/2.   in1,in2,out      IC=__
.EXOR/2.  in1,in2,out      IC=__
```

(a)

```
.AND/4.   in1,in2,in3,in4,out    IC=__
.NAND/4.  in1,in2,in3,in4,out    IC=__
.OR/4.    in1,in2,in3,in4,out    IC=__
.NOR/4.   in1,in2,in3,in4,out    IC=__
.EXOR/4.  in1,in2,in3,in4,out    IC=__
```

(b)

.INPUT. $a1,a2,a3,\ldots,a_n$

.PRINT. $b1,b2,b3,\ldots,b_n$

(c)

```
.END.
```

(d)

Figure 4-2

last line in the program must be as shown in Figure 4-2d. This statement informs the compiler that there is no more source code to be compiled.

## 4.3 The Controller/Editor

The controller/editor module performs two duties for DLS. Its first task is to interact with the user to determine what action DLS is to perform. The second duty is to edit the source program which the user loads into the computer via a terminal.

The source program is loaded one line at a time. Each line must have a four digit identification as the first four characters. This is simular to the program language Basic. As each line comes in the source program is scanned for where the new line will go. This is done by scanning the source program for the other line indentifiers then comparing it with that of the new line. Figure 4-3 shows the flowchart depicting how the editor goes about placing a new line into memory. What must first be done is to determine if a line with the same number already exists in the source program. If it does it must first be deleted from memory. After that has been determined then the routine finds where the new line goes and puts it there.

Figure 4-4 is the controller routine flowchart. Its

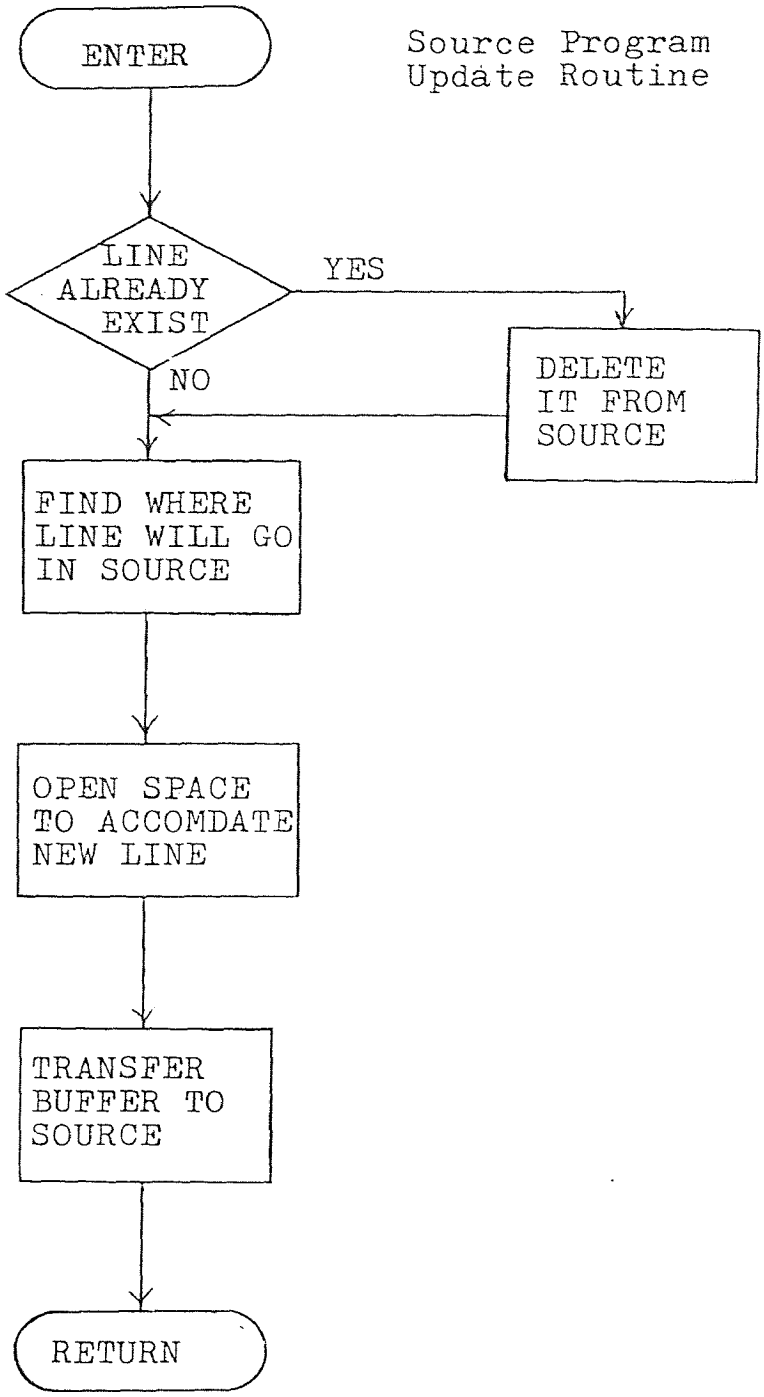

ENTER
Source Program
Update Routine
LINE
ALREADY
EXIST
YES
NO
DELETE
IT FROM
SOURCE
FIND WHERE
LINE WILL GO
IN SOURCE
OPEN SPACE
TO ACCOMDATE
NEW LINE
TRANSFER
BUFFER TO
SOURCE
RETURN

Figure 4-3

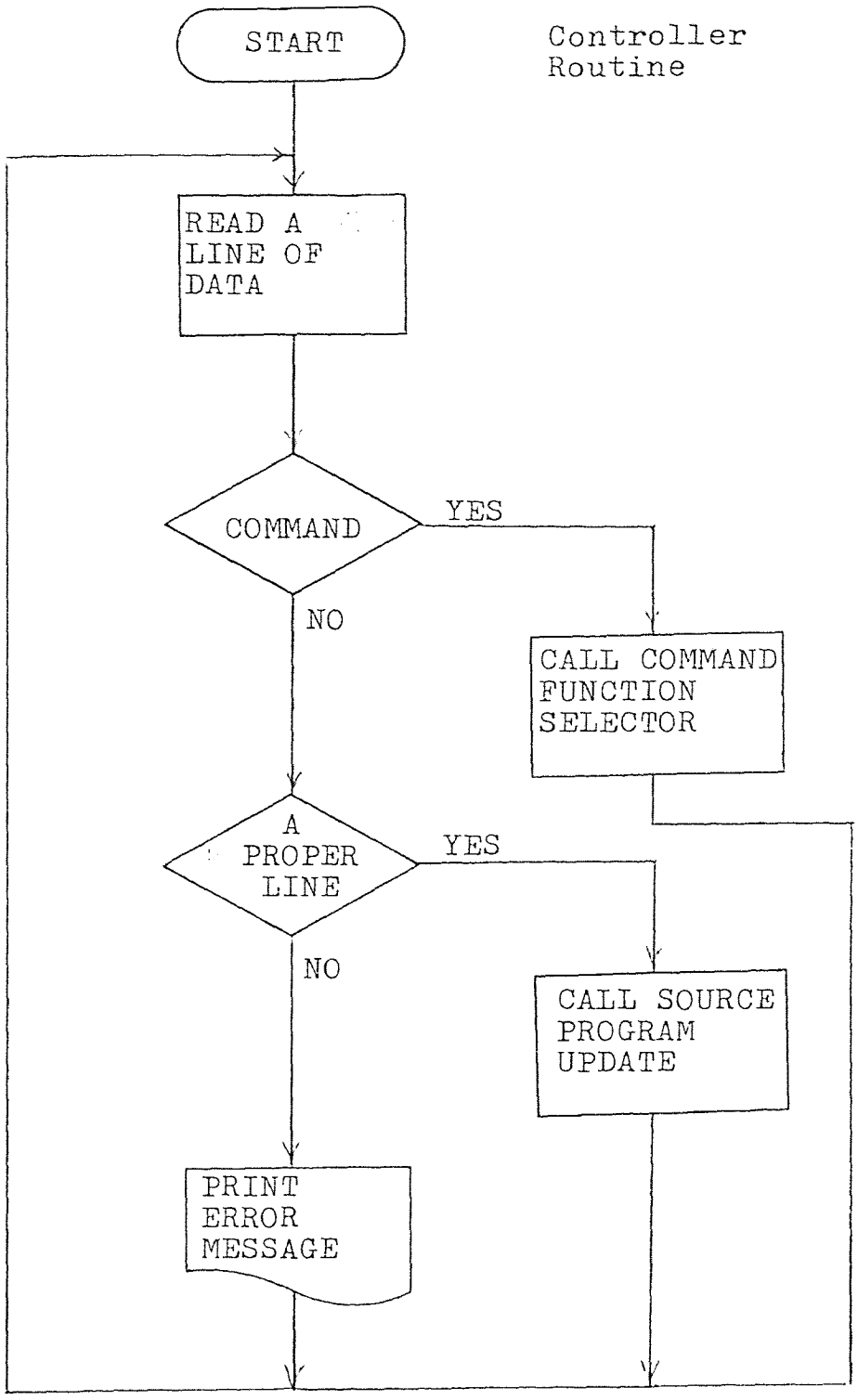

START
Controller Routine
READ A LINE OF DATA
COMMAND
YES
NO
CALL COMMAND FUNCTION SELECTOR
A PROPER LINE
YES
NO
CALL SOURCE PROGRAM UPDATE
PRINT ERROR MESSAGE

Figure 4-4

Command Function
Selection Routine

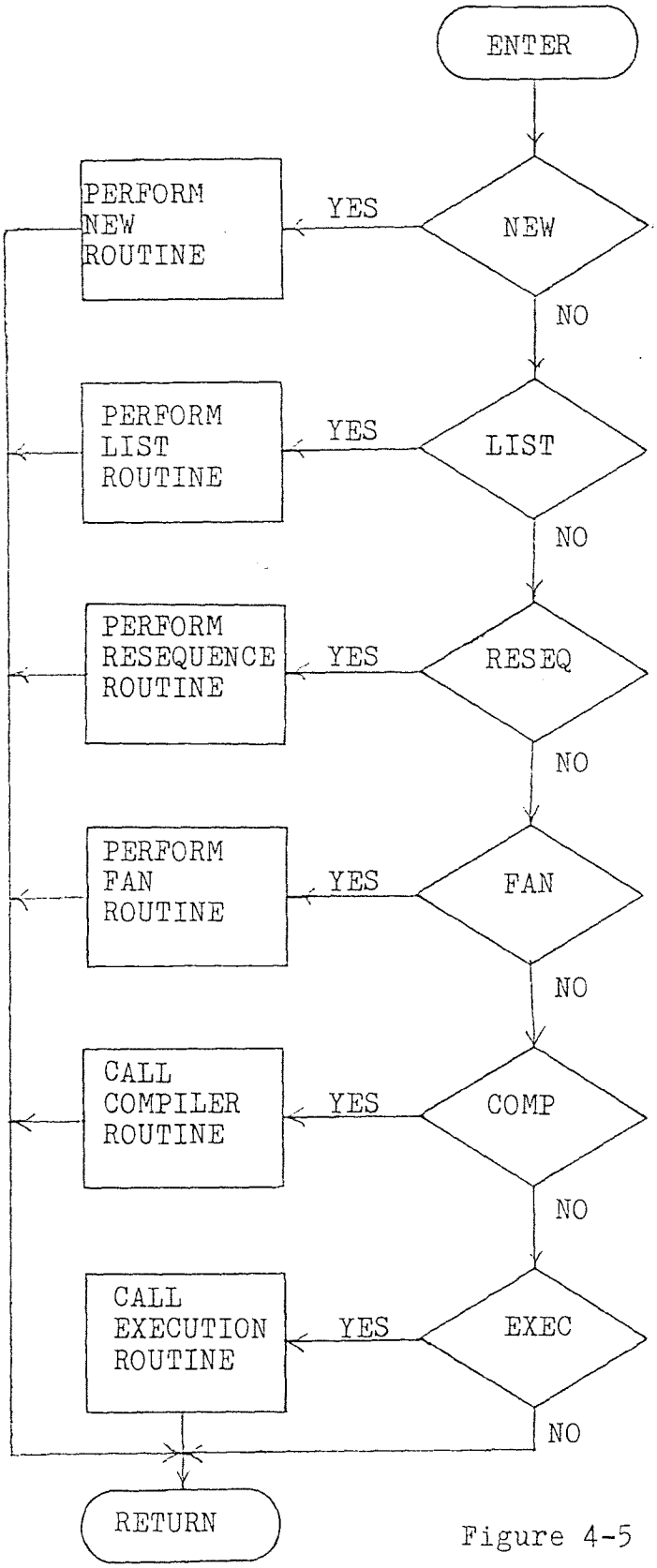



Figure 4-5

task is to get a line of information from the user and determine if it is a command or source data. If it is source data and it fits the proper format then the source program update routine will be implemented. If it is a command then the controller will call the command function selector routine. The function selection process is done by simply matching up the contents in the input buffer to some test patterns to determine which function is to be implemented. This process is shown in Figure 4-5.

DLS has six command functions which can be performed. Two of these commands (COMP and EXEC) will pass control over to either the compiler or the execution modules. Three of the remaining four commands are editor orientated. This entails some sort of source program manipulation. The command NEW will reset the source program memory pointers, erasing any previous source program. The command LIST will print all of the source program which had been entered by the user. The command RESEQ will resequence all the line identifiers of the source program in memory. Starting from zero for the first line and working up in steps of ten.

The sixth command FAN can not be called upon until the compiler module has been implemented. FAN will calculate the fanout (the number of connections per logic line) of the simulated network.

## 4-4 The Controller/Editor Program Listing

```
;*****************************************************************
        ;FUNCTION       :MAIN PROGRAM
        ;CALLS          :OUTCH,GETCH,CRLF
        ;INPUTS         :BEGIN,DATA,BUFFR
        ;OUTPUTS        :START,LENTH
        ;DESCRIPTION    :THIS IS THE CONTROLLER AND
        ;               :EDITOR PROGRAM. IT PRINTS
        ;               :OUT ALL THE NECESARY TITLES
        ;               :AND INTERACTS WITH THE USER
        ;               :TO DETERMINE IF THE USER IS
        ;               :INPUTING A STRING OF DATA OR
        ;               :REQUESTING CERTAIN OPERATIONS
        ;               :TO TAKE PLACE
;*****************************************************************
;
;
;
;
;
BEGIN:  LXI     SP,BEGIN
        LXI     H,DATA  ;BEGINNING OF SOURCE PROGRAM
        SHLD    START
        LXI     H,AA2   ;PRINT OUT THE PROGRAM TITLES
AA1:    MOV     A,M
        INX     H
        CPI     0
        JZ      NEW
        CALL    OUTCH
        JMP     AA1
AA2:    DB      0DH,0AH
        DB      0AH,20H,20H,'DIGITAL LOGIC'
        DB      ' SIMULATOR',0AH,0AH,0
AA3:    CALL    CRLF
        MVI     B,64    ;SETUP AN INPUT BUFFER
        LXI     H,BUFFR
AA4:    MVI     M,20H
        INX     H
        DCR     B
        JNZ     AA4
        MVI     A,':'
        CALL    OUTCH   ;PRINT THE PROMPT MESSAGE
        LXI     H,BUFFR
        MVI     B,0
AA5:    CALL    GETCH   ;INPUT A STRING OF CHARACTERS
        CPI     0DH     ;TO BE INTERPERATED. CR ENDS A LINE
        JZ      AA7
        CPI     18H     ;CONTROL 'X' KILLS THE LINE
        JZ      AA3
        CPI     7FH     ;THIS BACKSPACES ONE CHARACTER IN
        JNZ     AA6     ;THE STRING
        MOV     A,B
```

```
        ORA     A
        JZ      AA3
        MVI     A,08H
        CALL    OUTCH
        DCR     B
        MVI     M,20H
        DCX     H
        JMP     AA5
AA6:    MOV     M,A     ;LOAD CHARACTER INTO BUFFER
        INX     H
        INR     B
        MOV     A,B
        CPI     65      ;BUFFER STRING CAN ONLY BE 64
        JNZ     AA5     ;CHARACTERS IN LENGTH
AA7:    LXI     H,BUFFR ;START TO INTERPERATE THE STRING
        MOV     A,B
        STA     LENTH
        MOV     A,M
        CPI     '0'     ;TEST TO SEE IF THE LINE STARTS
        JM      AA8
        CPI     '9'+1   ;WITH A DIGIT WHICH MEANS THE
        JM      LINE    ;STRING IS DATA TO BE STORED IN
        CPI     'L'     ;THE SOURCE PROGRAM
        JZ      LIST    ;JUMP TO LIST ROUTINE
        CPI     'N'
        JZ      NEW     ;JUMP TO 'NEW' ROUTINE
        CPI     'R'
        JZ      RESEQ   ;JUMP TO 'RESEQUENCE' ROUTINE
        CPI     'C'
        JZ      COMP    ;JUMP TO THE COMPILER ROUTINE
        CPI     'E'
        JZ      EXEC    ;JUMP TO THE EXECUTION ROUTINE
        CPI     'F'
        JZ      FAN     ;JUMP TO THE FANOUT ROUTINE
AA8:    LXI     H,AA10  ;IF NO MATCH EXISTS THEN
AA9:    MOV     A,M     ;PRINT OUT THE ERROR MESSAGE
        CPI     0
        JZ      AA3     ;THEN TRY AGAIN
        CALL    OUTCH
        INX     H
        JMP     AA9
AA10:   DB      0DH,0AH,'**ERROR**',0
;
;
;
;
;
;*****************************************************************
        ;FUNCTION        :LIST
        ;CALLS           :CRLF,OUTCH
        ;INPUTS          :NOTHING
```

```
          ;OUTPUTS          ;NOTHING
          ;DESCRIPTION      ;LIST PRINTS OUT THE USERS
          ;                 ;SOURCE PROGRAM FROM MEMORY
;*****************************************************
;
;
;
;
;
LIST:     CALL      CRLF    ;CALL CARRAGE RETURN AND
          CALL      CRLF    ;LINE FEED
          LHLD      NEXT    ;LAST BYTE OF SOURCE PROGRAM
          XCHG
          LHLD      START   ;FIRST BYTE OF SOURCE PROGRAM
AB1:      MOV       A,L
          CMP       E
          JNZ       AB2
          MOV       A,H     ;TEST TO SEE IF THIS IS THE LAST
          CMP       D       ;BYTE TO BE PRINTED
          JZ        AA3
AB2:      MOV       A,M
          CALL      OUTCH   ;OUTPUT THE CHARACTER TO THE PRINTER
          INX       H
          JMP       AB1     ;GET NEXT BYTE
;
;
;
;
;*****************************************************
          ;FUNCTION         ;NEW
          ;CALLS            ;NOTHING
          ;INPUTS           ;START
          ;OUTPUTS          ;NEXT
          ;DESCRIPTION      ;NEW CLEARS OUT THE SOURCE
          ;                 ;OLD PROGRAM MEMORY BUFFER
;*****************************************************
;
;
;
;
;
NEW:      CALL      CRLF    ;PRINT CARRAGE RETURN AND LINE FEED
          LXI       H,AC2   ;PRINT THE MEMORY PROTECT MESSAGE
AC1:      MOV       A,M
          CPI       0
          JZ        AC3
          CALL      OUTCH
          INX       H
          JMP       AC1
AC2:      DB        'CLEAR MEMORY ?',0
```

```
AC3:    CALL    GETCH   ;GET A CHARACTER FROM THE CONSOLE
        CPI     'N'     ;FOR THE RESPONCE TO THE QUESTION
        JZ      AA3     ;N-- DON'T CLEAR THE MEMORY
        CPI     'Y'     ;Y-- CLEAR THE MEMORY
        JNZ     AC1-3   ;ANYTHING ELSE TRY ACAIN
        LHLD    START
        SHLD    NEXT
        JMP     AA3
;
;
;
;
;*****************************************************************
        ;FUNCTION        :RESEQ
        ;CALLS           :NOTHING
        ;INPUTS          :NEXT,START,WORK
        ;OUTPUTS         :NOTHING
        ;DESCRIPTION     :EACH LINE OF SOURCE PROGRAM HAS
        ;                :A FOUR DIGIT NUMBER ASSIGNED TO
        ;                :IT, RESEQ WILL RESEQUENCE THE
        ;                :FOUR DIGITS IN STEPS OF TEN
;*****************************************************************
;
;
;
;
;
RESEQ:  LHLD    NEXT    ;GET THE FIRST AND LAST
        XCHG
        LHLD    START   ;BYTES OF THE SOURCE PROGRAM
        MVI     A,'0'   ;SET THE LINE COUNTER TO ZERO
        PUSH    H
        LXI     H,WORK
        MVI     B,4
AD1:    MOV     M,A     ;STORE THE LINE NUMBER AWAY
        INX     H
        DCR     B
        JNZ     AD1
        POP     H
AD2:    MOV     A,L     ;TEST TO SEE IF THIS IS THE LAST
        CMP     E       ;LINE HAS BEEN RESEQUENCED
        JNZ     AD3
        MOV     A,H
        CMP     D
        JZ      AA3     ;IF ALL DONE RETURN TO CONTROLLER
AD3:    MOV     A,M     ;SCAN FOR THE BEGINNING OF A LINE
        INX     H
        CPI     0AH
        JNZ     AD2
        PUSH    D
```

```
        MVI     B,4
        LXI     D,WORK  ;RESEQUENCE THIS LINE
AD4:    LDAX    D
        MOV     M,A     ;UPDATE RESEQUENCE COUNTER
        INX     D
        INX     H
        DCR     B
        JNZ     AD4
        MVI     B,3
AD5:    DCX     D
        DCX     D
        LDAX    D
        INR     A
        CPI     '9'+1   ;COUNTER IS A DECMIAL COUNT
        JP      AD7
        STAX    D
AD6:    POP     D
        JMP     AD3
AD7:    MVI     A,'0'
        STAX    D
        DCR     B
        JNZ     AD5
        JMP     AD6
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;LINE
        ;CALLS           ;EXIST,FIND,OPEN,TRANS
        ;INPUTS          ;BUFFR,LENTH
        ;OUTPUTS         ;NOTHING
        ;DESCRIPTION     ;LINE IS THE ROUTINE WHICH
        ;                ;TAKES THE INPUT DATA STRING
        ;                ;WHICH IS TEMPORALLY IN A DATA
        ;                ;BUFFER AND MOVES IT TO ITS'
        ;                ;PROPER LOCATION IN THE
        ;                ;SOURCE PROGRAM
;*****************************************************************
;
;
;
;
;
LINE:   LXI     H,BUFFR+1
        MVI     B,3     ;TEST TO MAKE SURE THAT THE
AE1:    MOV     A,M     ;LINE IN THE BUFFER HAS A
        CPI     '0'     ;FOUR DIGIT IDENTIFIER ON IT
        JM      AA8
        CPI     '9'+1
```

```
        JP      AA8
        INX     H
        DCR     B
        JNZ     AE1
        CALL    EXIST   ;SEE IF THE LINE EXISTS ALREADY
        LDA     LENTH   ;IF IT DOES DESTROY THAT LINE
        CPI     6
        JM      AA3
        CALL    FIND    ;FIND WHERE THE LINE SHOULD GO
        CALL    OPEN    ;OPEN A SPACE FOR THE LINE
        CALL    TRANS   ;MOVE BUFFER INTO SOURCE MEMORY
        JMP     AA3
;
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;EXIST
        ;CALLS           ;NOTHING
        ;INPUTS          ;NEXT,START,BUFFR
        ;OUTPUTS         ;NEXT
        ;DESCRIPTION     ;EXIST EXAMINES THE FOUR DIGIT
        ;                ;IDENTIFIERS IN THE TEMPORY
        ;                ;BUFFER AND SEARCHES THROUGH THE
        ;                ;SOURCE PROGRAM TO SEE IF A LINE
        ;                ;WITH THE SAME NUMBER EXISTS. IF
        ;                ;IT DOES THAT LINE WILL BE DESTROYED
;*****************************************************************
;
;
;
;
;
EXIST:  LHLD    NEXT    ;LOAD THE PARAMETERS OF THE
        XCHG            ;SOURCE PROGRAM
        LHLD    START
AG1:    MOV     A,L     ;TEST TO SEE IF A COMPLETE
        CMP     E       ;SEARCH HAS BEEN MADE
        JNZ     AG2
        MOV     A,H
        CMP     D
        JNZ     AG2
        RET             ;LINE NOT FOUND
AG2:    MOV     A,M
        INX     H
        CPI     0AH     ;FIND THE BEGINNING OF A LINE
        JNZ     AG1
        PUSH    D
        PUSH    H
        MVI     D,4
```

```
        LXI     B,BUFFR ;LOAD THE FOUR DIGITS FROM BUFFER
AG3:    LDAX    B
        CMP     M       ;COMPARE WITH THE IDENTIFIER
        JZ      AG4     ;IN THE SOURCE PROGRAM
        POP     H
        POP     D
        JMP     AG1
AG4:    INX     B
        INX     H
        DCR     D
        JNZ     AG3
        POP     H
        POP     D
        DCX     H
        DCX     H
        PUSH    D
        POP     B
        PUSH    H
        POP     D
        INX     D
AG5:    LDAX    D
        CPI     0DH
        JZ      AG6
        INX     D
        MOV     A,D
        CMP     B       ;IF THE LINE IS FOUND TO BE
        JNZ     AG5     ;THE LAST LINE IN MEMORY THEN
        MOV     A,E     ;RESET THE NEXT BYTE TO THE
        CMP     C       ;BEGINNING OF THIS LINE
        JNZ     AG5
        SHLD    NEXT
        RET
AG6:    LDAX    D
        MOV     M,A     ;LINE HAS BEEN FOUND DESTROY IT
        INX     D
        INX     H
        MOV     A,D     ;TRANSFER THE REST OF THE
        CMP     B       ;MEMORY BLOCK TO CLOSE THE
        JNZ     AG6     ;AREA WHERE THE OLD LINE WAS
        MOV     A,E
        CMP     C
        JNZ     AG6
        SHLD    NEXT    ;RECALCULATED NEXT BYTE ADDRESS
        RET
;
;
;
;
;
;*****************************************************************
        ;FUNCTION        :TRANS
```

```
          ;CALLS              :NOTHING
          ;INPUTS             :BUFFR,WORK
          ;OUTPUTS            :NOTHING
          ;DESCRIPTION        :TRANS WILL TRANSFER THE INPUT
          ;                   :DATA STRING WHICH RESIDES IN
          ;                   :THE TEMPORY BUFFER,TO THE
          ;                   :SOURCE PROGRAM MEMORY
;*************************************************************
;
;
;
;
;
TRANS:    LXI     D,BUFFR ;BEGINNING OF THE BUFFER
          LHLD    WORK    ;WHERE IN MEMORY IT WILL GO
          MVI     M,0DH
          INX     H
          MVI     M,0AH   ;ATTACH THE LEADER CHARACTERS
          INX     H
          MVI     B,64
AF1:      LDAX    D
          MOV     M,A     ;TRANSFER THE BUFFER OVER
          INX     H       ;TO SOURCE MEMORY
          INX     D
          DCR     B
          JNZ     AF1
          RET
;
;
;
;
;
;*************************************************************
          ;FUNCTION           :FIND
          ;CALLS              :NOTHING
          ;INPUTS             :NEXT,START,BUFFR
          ;OUTPUTS            :WORK
          ;DESCRIPTION        :FIND ROUTINE SEARCHES THROUGH
          ;                   :MEMORY TO FIND THE ADDRESS
          ;                   :WITHIN THE SOURCE PROGRAM
          ;                   :WHERE THE NEW LINE OF DATA GOES
;*************************************************************
;
;
;
;
;
FIND:     LHLD    NEXT    ;LOAD THE SOURCE PROGRAM PARAMETERS
          XCHG
          LHLD    START
AH1:      MOV     A,L     ;CONDUCT A MEMORY SEARCH
```

```
          CMP     E
          JNZ     AH2
          MOV     A,H
          CMP     D
          JNZ     AH2
          XCHG
          SHLD    WORK
          RET
AH2:      MOV     A,M
          INX     H
          CPI     0AH       ;TO FIND THE BEGINNING OF A LINE
          JNZ     AH1
          PUSH    D
          PUSH    H
          MVI     D,4
          LXI     B,BUFFR ;COMPARE THE FOUR DIGIT INDENTIFIERS
AH3:      LDAX    B         ;TO DETERMINE IF THE LINE IN THE
          CMP     M         ;BUFFER SHOULD GO BEFORE THIS
          JM      AH5       ;LINE IN MEMORY
          JZ      AH4
          POP     H
          POP     D
          JMP     AH1
AH4:      INX     B
          INX     H
          DCR     D
          JNZ     AH3       ;NOT THIS LINE MOVE ON TO
AH5:      POP     H         ;NEXT LINE
          DCX     H
          DCX     H
          POP     D
          SHLD    WORK      ;FOUND WHERE IT SHOULD GO
          RET
;
;
;
;
;*****************************************************************
          ;FUNCTION         :OPEN
          ;CALLS            :NOTHING
          ;INPUTS           :NEXT,WORK
          ;OUTPUTS          :NEXT
          ;DESCRIPTION      :OPEN IS THE ROUTINE WHICH
          ;                 :OPENS A 66 BYTE STRING IN
          ;                 :THE SOURCE PROGRAM TO MAKE
          ;                 :ROOM FOR THE INCERTION
          ;                 :OF THE NEW LINE OF DATA
;*****************************************************************
;
;
```

```
;
;
;
OPEN:   LHLD    NEXT    ;GET THE LAST BYTE OF DATA
        XCHG
        LHLD    WORK    ;THIS IS WHERE THE DATA INCERTION
        PUSH    H       ;WILL TAKE PLACE
        POP     B
        DCX     B
        LXI     H,66
        DAD     D
        SHLD    NEXT    ;MOVE THE LAST BYTE OF DATA
AI1:    MOV     A,D     ;66 BYTES LOWER
        CMP     B
        JNZ     AI2
        MOV     A,E
        CMP     C
        RZ
AI2:    LDAX    D       ;MOVE THE BLOCK OF DATA FROM
        MOV     M,A     ;THE POINT WHERE THE INCERTED
        DCX     D       ;LINE WILL GO TO THE LAST
        DCX     H       ;LINE,DOWN TO THE NEW NEXT LOCATION
        JMP     AI1
;
;
;
;
;
;*****************************************************************
        ;FUNCTION        :FAN
        ;CALLS           :CRLF,OUTCH,PRBYT
        ;INPUTS          :SYMBS,SIMTE,SIMTS,WORK
        ;OUTPUTS         :WORK
        ;DESCRIPTION     :FAN SEARCHES THROUGH THE
        ;                :SYMBOL TABLE TO FIND EACH
        ;                :SYMBOL AND COUNT HOWMANY
        ;                :TIMES THAT SYMBOL IS USED
        ;                :IN THE NETWORK FOR COMPUTING
        ;                :THE FANOUT OF EACH LOGIC LEVEL
;*****************************************************************
;
;
;
;
;
FAN:    CALL    CRLF
        CALL    CRLF
        LHLD    SYMBS   ;START OF SYMBOL TABLE
AJ0:    MVI     D,5
AJ1:    MOV     A,M     ;GET A SYMBOL
```

```
        CPI     '@'     ;END OF SYMBOL TABLE INDICATOR
        JZ      AA3
        CPI     0
        JNZ     $+5
        MVI     A,20H
        CALL    OUTCH   ;PRINT THE SYMBOL
        INX     H
        DCR     D
        JNZ     AJ1
        MVI     A,':'
        CALL    OUTCH
        MOV     C,M
        INX     H
        MOV     B,M
        INX     H
        PUSH    H
        MVI     A,0     ;STORE THE ADDRESS OF THE SYMBOL
        STA     WORK    ;FROM T1 TABLE IN THE WORK REGISTER
        LHLD    SIMTE
        XCHG
        LHLD    SIMTS   ;LOAD SIMULATION TABLE
AJ2:    MOV     A,H
        CMP     D       ;SYMBOL TABLE SEARCH
        JNZ     AJ3
        MOV     A,L
        CMP     E
        JNZ     AJ3
        POP     H
        LDA     WORK    ;SEARCH DONE PRINT THE RESULTS
        CALL    PRBYT   ;OF HOW MANY TIMES THAT
        CALL    CRLF    ;SYMBOL IS USED
        INX     H
        INX     H
        JMP     AJ0     ;MOVE ON TO NEXT SYMBOL
AJ3:    MOV     A,M
        INX     H
        CMP     C
        JZ      AJ4     ;MAKE THE ADDRESS COMPARISON
        INX     H
        JMP     AJ2
AJ4:    MOV     A,M
        INX     H
        CMP     B
        JNZ     AJ2
        LDA     WORK
        ADI     1       ;EACH TIME A MATCH EXISTS
        DAA             ;ADD ONE TO ITS FANOUT COUNT
        STA     WORK
        JMP     AJ2
```

## 4.5 The DLS Compiler

The routines which form the compiler portion of the simulator are the heart of DLS. The compiler module can be broken down into six sub-modules and it is the task of these sub-modules to create the various tables which drive the simulator.

Once the source program has been entered into memory via the controler/editor, the user issues the proper command word (COMP) which initates the execution of the compiler. The DLS compiler is unlike the standard meaning of a compiler, where the source program is broken down into another form of a program which is more easly understood by the computer. The DLS compiler does not work this way. It makes several passes over the source program extracting different pieces of information as it goes along.

Memory is partitioned off by the compiler for the formation of the tables where the extracted information will reside. For example the compiler has to know how many symbols the source program uses. This determines the size of the tables T1 and T2. The compiler must also know how many of each logic gate from the gate library are being called upon. This determines the size of the simulation update sequence tables and so on.

In the style of modular programming the compiler routine is simply a controller. Figure 4-6 is the flow-

Compiler Function
Routine

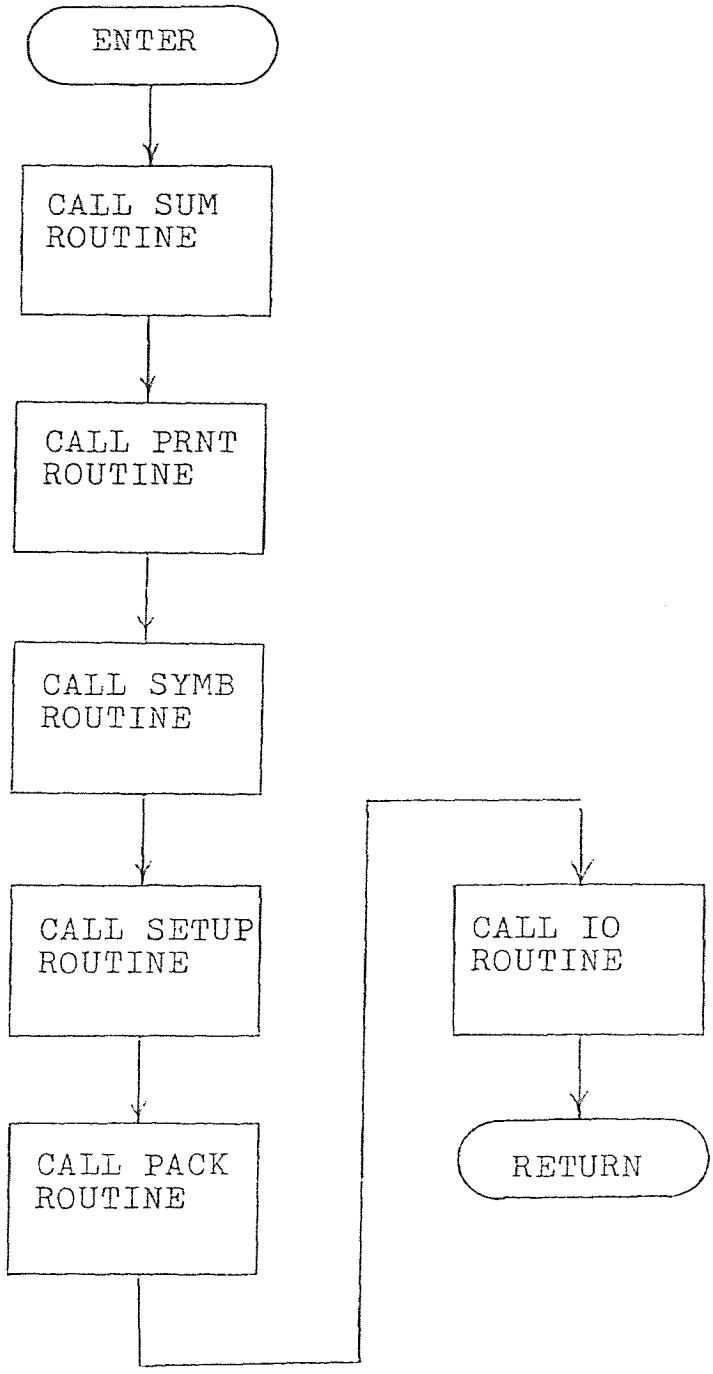



Figure 4-6

chart of the compiler routine. It performs the duty of directing the flow of the program through several routines. The six routines called upon are: SUM,PRNT,SYMB,PACK,IO,SETUP. Each of these sub-modules may have several sub-sub-modules which will be called upon.

The SUM routine is assigned the task of determining how many of each type of logic gate are going to be used in the simulation. Figure 4-7 is the flowchart for this routine. There are ten types of logic gates which can be implemented by DLS. The SUM routine sets up the table which will keep track of the gate count. The routine will terminate when the end control word is encountered.

The PRNT routine does not extract any information from the source program but rather aids in error detection. PRNT prints out the source program listing along with the gate count table. The user can readily determine if all the logic gates were accounted for in the compiling. Figure 4-8 shows the flowchart for this routine.

The SYMB routine performs a major task. It scans through the source program picking out all the different symbols being used. The routine must be able to distinguish between a symbol and some other type of information. Figure 4-9 has the flowchart of this routine. To determine what is what the routine first looks for a line containing a control word. Once this has been determined and the proper lines found, SYMB will proceed with its function.

Sum Function Routine

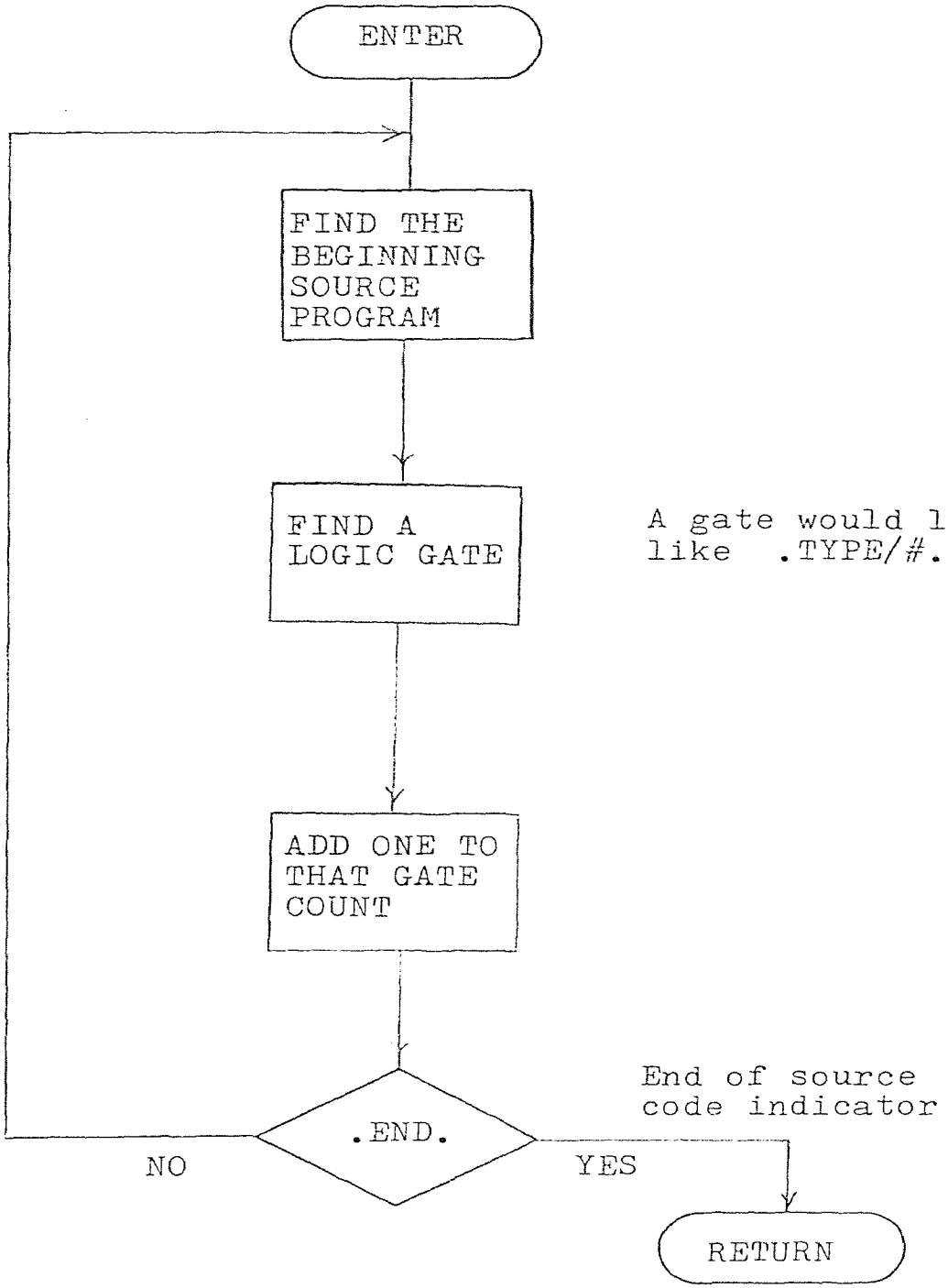
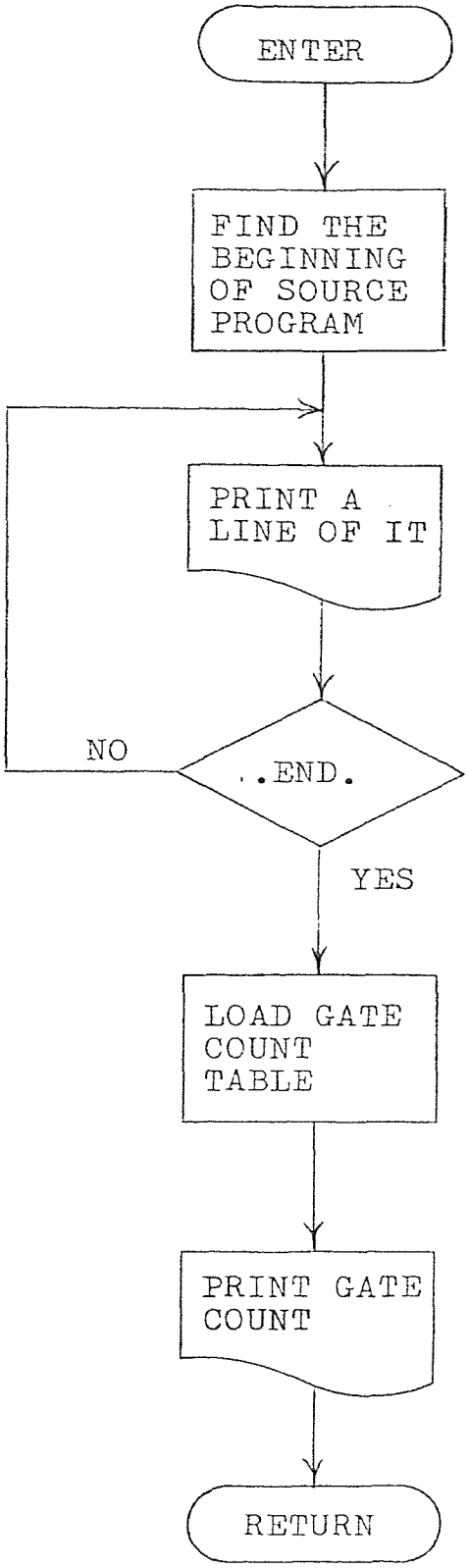



Figure 4-7




Prnt Function Routine

Figure 4-8

Symb Function
Routine
(Part 1)

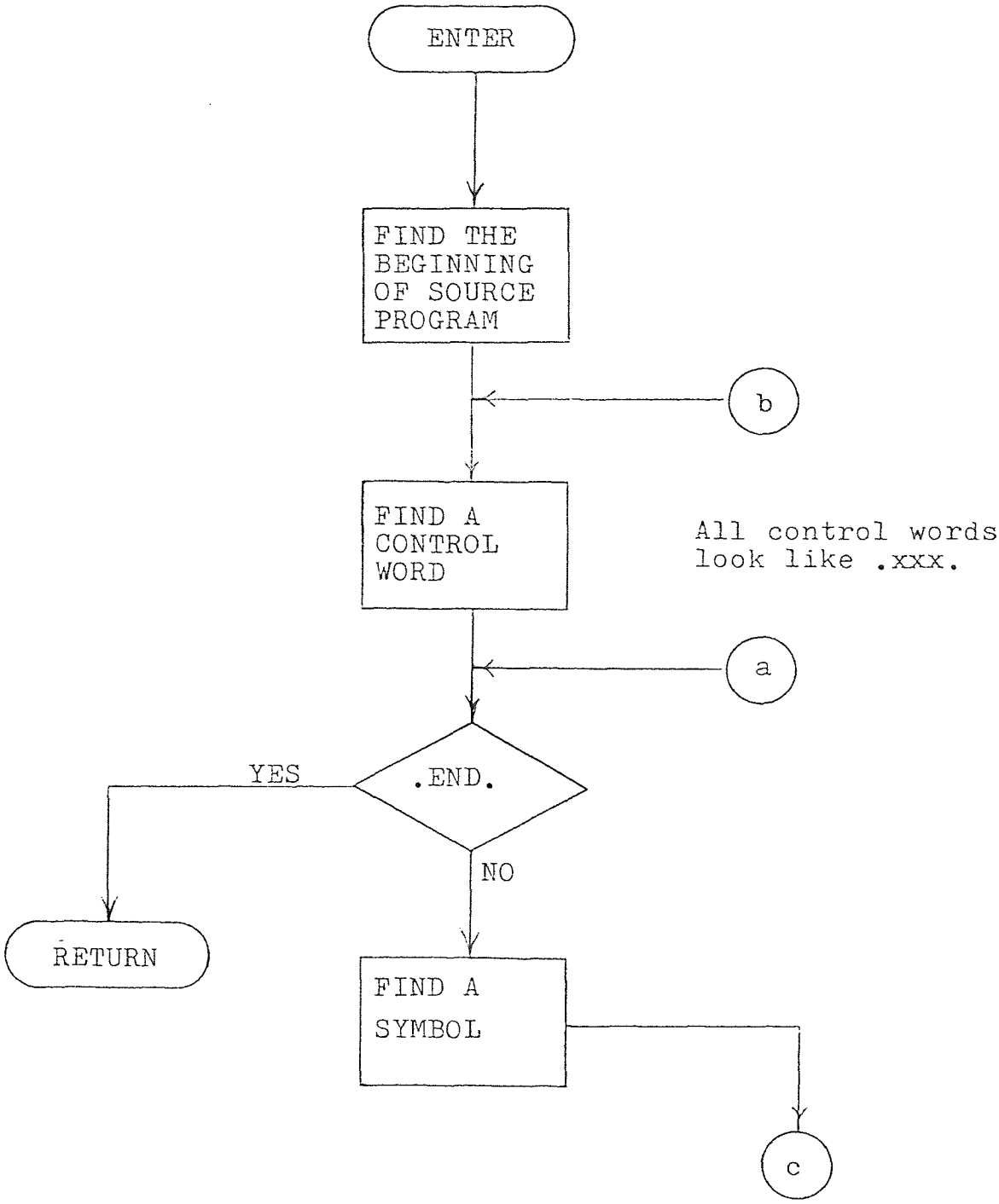



Figure 4-9

Symb Function
Routine
(Part 2)

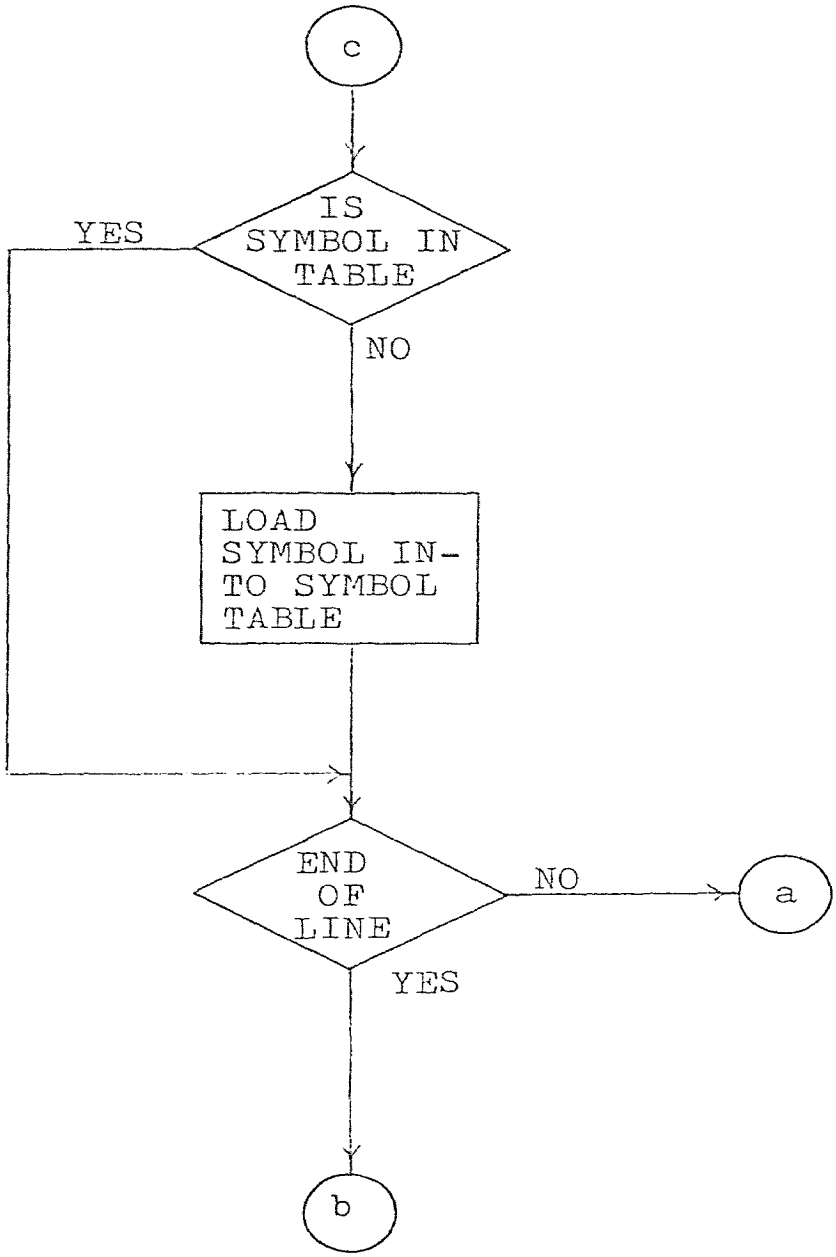


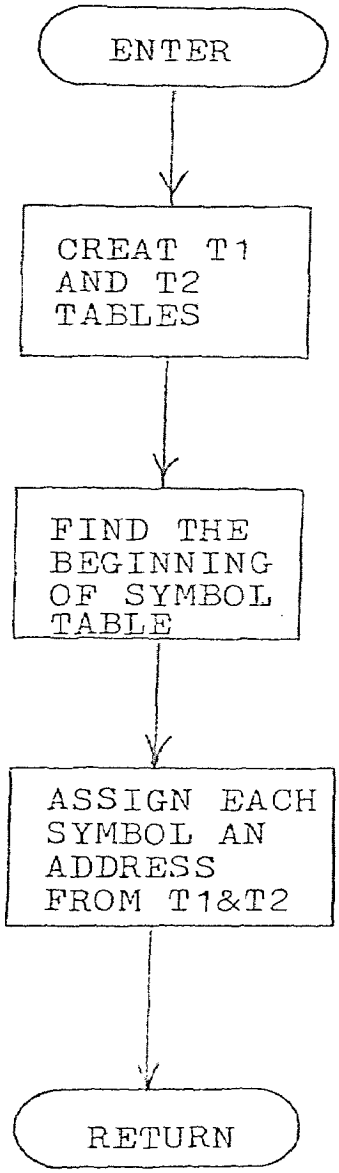

Figure 4-9

As each symbol is encountered it is run through a test to see if it already exists in the symbol table. If it is in the table the routine will move on to the next symbol. If not then this new symbol will be loaded into the table along with room for the two simulation table addresses to be assigned later. These addresses will be found once tables T1 and T2 are formed.

Once all the symbols have been found the next two tables can be formed. This is done by the SETUP routine, Figure 4-10. A count of the number of symbols used was kept by the last routine. The size of the two tables depends upon the number of symbols. After the beginning and end addresses of T1 and T2 are determined SETUP will go back and assign each symbol in the symbol table addresses to T1 and T2.

Now that each symbol has a place in both simulation tables and both tables have been formed, what is left is to make an update sequence. This is accomplished by the PACK routine. What this routine does is to search through the source program looking for logic gates. Each gate definition contains information related to the number of inputs. PACK then looks for the input symbols and the output symbol and gets their addresses from tables T1 and T2. It then assigns these addresses to the update sequence table. A two input gate has two locations in T1 and its output located in T2. For a four input type gate

Setup Function Routine




The size of T1&T2 =2+(#ofsymbols) locations

Figure 4-10

Pack Function
Routine
(Part 1)

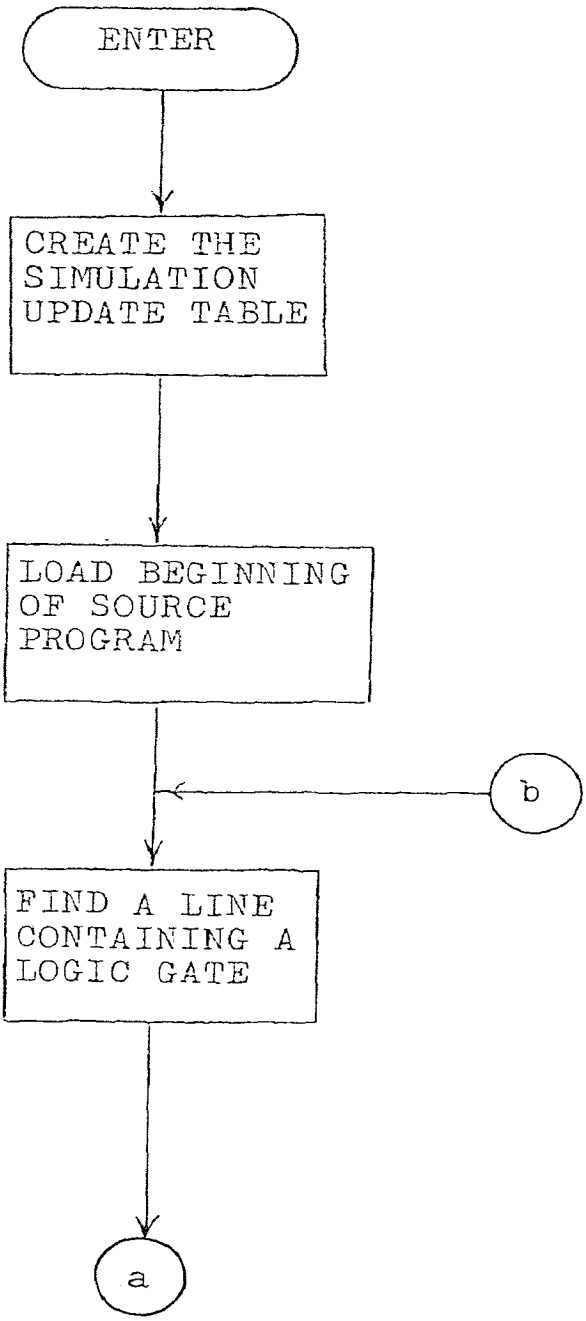



Figure 4-11

Pack Function
Routine
(Part 2)

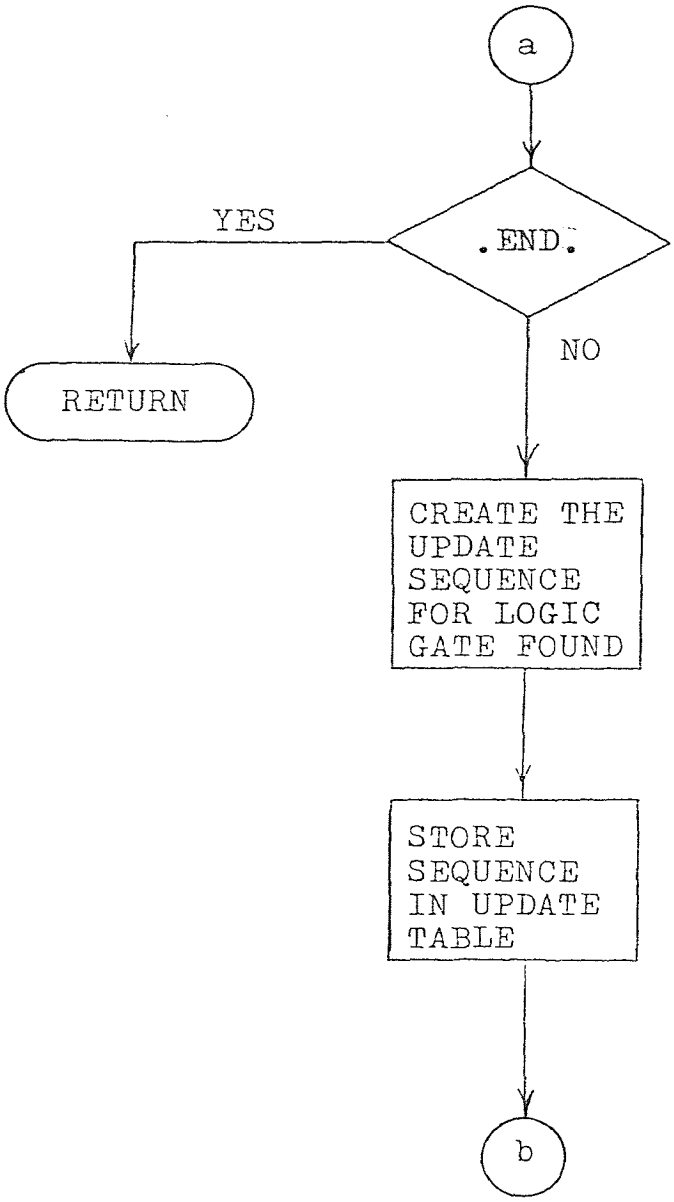



Figure 4-11

Io Function
Routine
(Part 1)

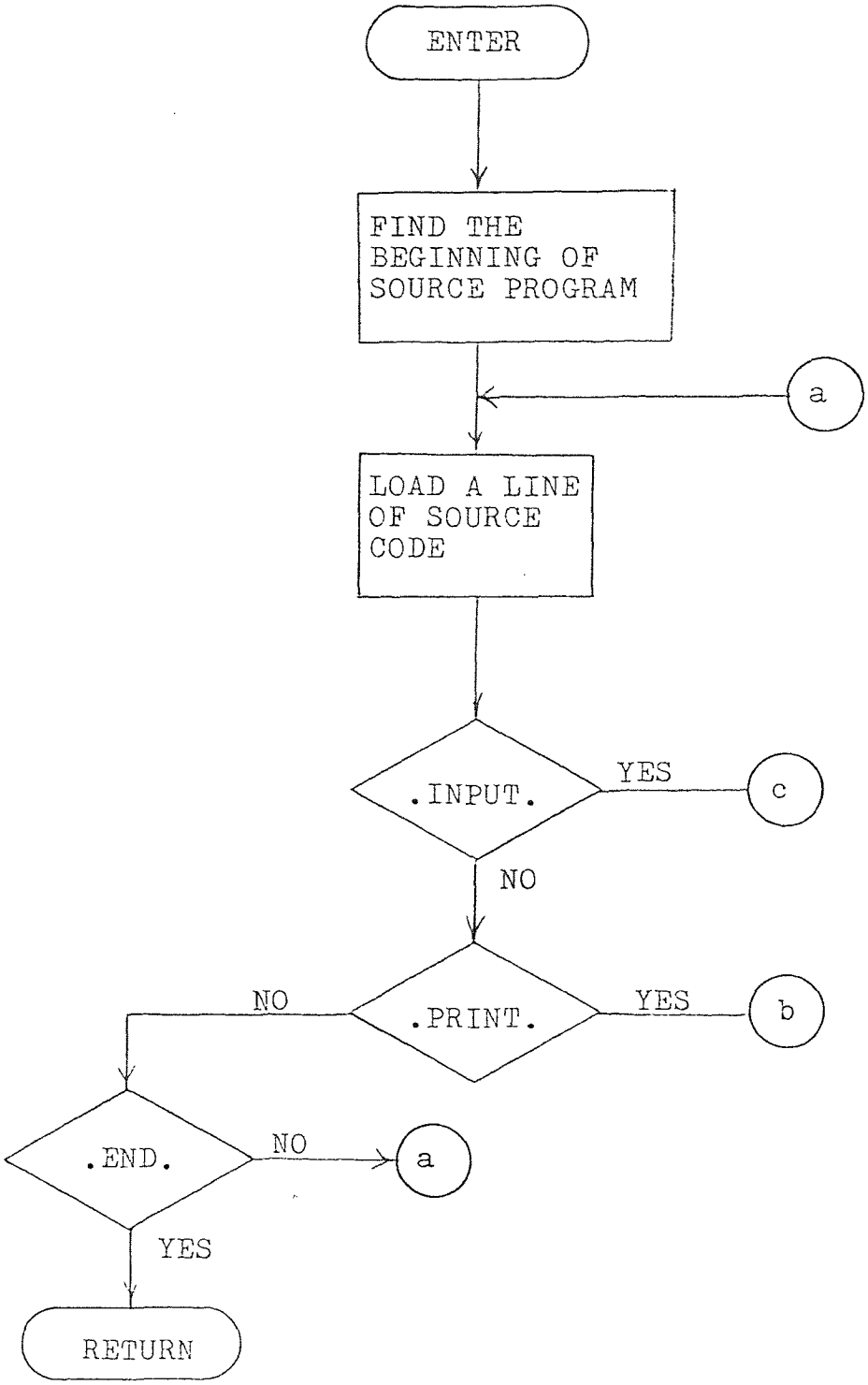



Figure 4-12

Io Function
Routine
(Part 2)

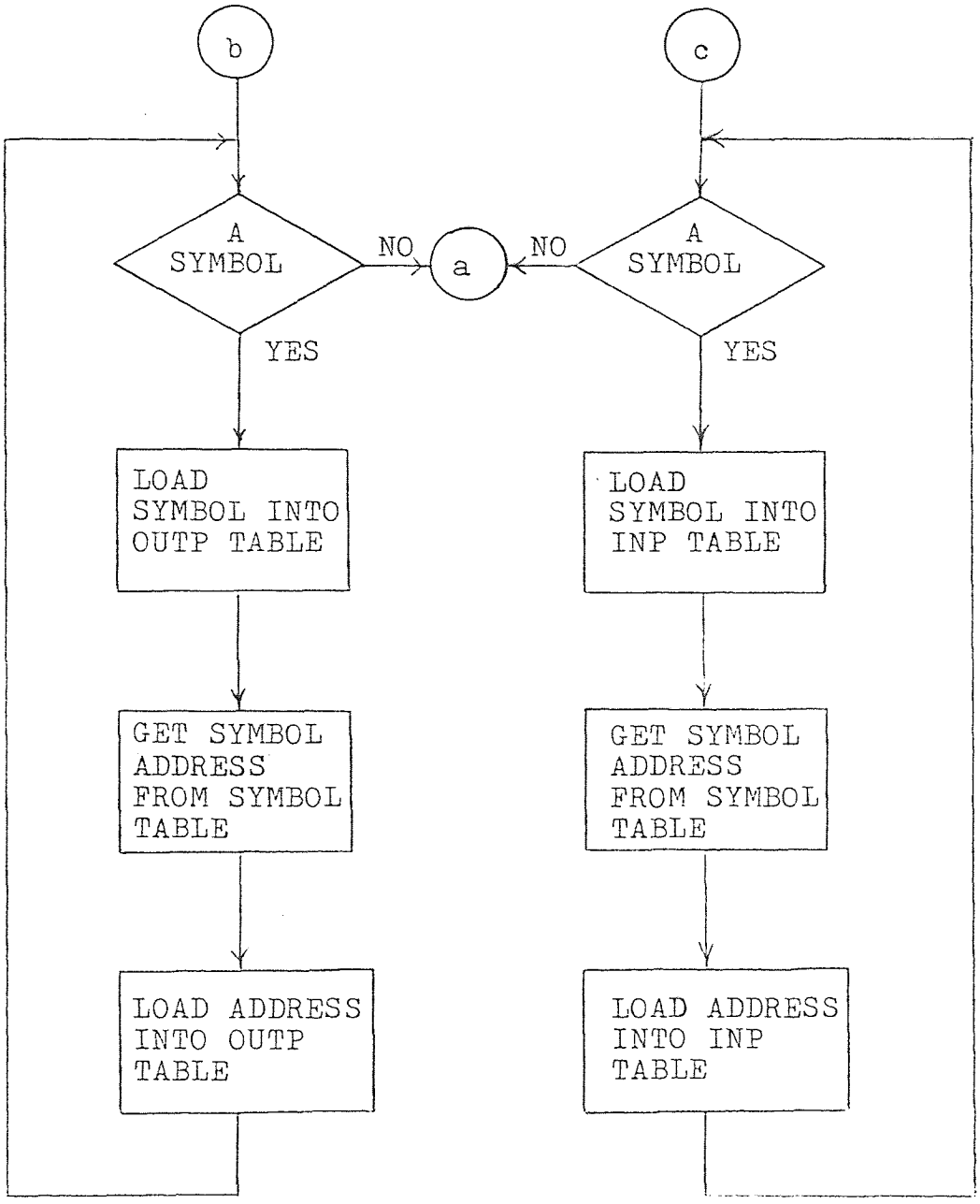



Figure 4-12

four of its locations are in T1 and its output is in T2. Figure 4-11 shows the flowchart for this routine.

The final sub-module of the compiler is the IO routine. It has the task of determining which variables are primary inputs and which are monitored outputs. Figure 4-12 shows this routine. This task is done by scanning through the source program looking for either INPUT or PRINT command words. When one of these is encountered each symbol which follows, along with its address of where in T1 it is located is stored in either table INP(input) or OUTP(output) depending on which command word was encountered. Once all the inputs and outputs have been stored away the compiling is complete. Control will now be passes back to the controller/editor where errors can be corrected or execution of the compiled program can take place.

## 4.6 The Compiler Program Listing

```
;*****************************************************************
          ;FUNCTION          :COMP -
          ;CALLS             :SUM,PRNT,SYMB,SETUP,PACK,IO
          ;INPUTS            :NOTHING
          ;OUTPUTS           :NOTHING
          ;DESCRIPTION       :COMP IS THE COMPILER ROUTINE
          ;                  :OF DLS.THE FUNCTION OF COMP
          ;                  :IS TO DIRECT THE IMPLIMENT-
          ;                  :ATION OF THE COMPILER. THERE
          ;                  :ARE SIX STAGES IN THIS COMPILER
          ;                  :AND COMP ACTS AS THE CONTROLLER
          ;                  :IT CALLS UPON THE NECESSARY
          ;                  :ROUTINES TO BREAKDOWN THE
          ;                  :SOURCE PROGRAM.
;*****************************************************************
;
;
;
;
COMP:     CALL     SUM       ;GET THE GATE TYPE COUNT
          CALL     PRNT      ;PRINT NETWORK PLUS GATE COUNT
          CALL     SYMB      ;ASSIGN LOCATIONS TO SYMBOLS
          CALL     SETUP     ;SETUP SIMULATION TABLES
          CALL     PACK      ;PUT THE INFORMATION IN TABLES
          CALL     IO        ;SETUP PRIMARY INPUT & OUTPUT
          JMP      AA3       ;GO BACK TO EDITOR
;
;
;
;
;*****************************************************************
          ;FUNCTION          :SUM
          ;CALLS             :FNDP,FNDCH,FNDS
          ;INPUTS            :START,WORK
          ;OUTPUTS           :WORK
          ;DESCRIPTION       :SUM HAS THE TASK OF DETERM-
          ;                  :INING HOW MANY OF THE POSSIBLE
          ;                  :ELEVEN TYPES OF GATES ARE IN
          ;                  :THE NETWORK.CERTAIN CHARACTERS
          ;                  :ARE USED TO KEYOFF THE ROUTINE.
          ;                  :FNDP- FINDS DECIMAL POINTS;
          ;                  :FNDCH- FINDS AN ALPHABETIC
          ;                  :CHARACTER.
;*****************************************************************
;
;
;
;
;
```

```
SUM:    MVI     A,11    ;COUNT OF GATE TYPES
        STA     WORK
        LXI     H,NA2   ;GATE COUNT TABLE
BA0:    MVI     M,0     ;INILIZATION OF TABLE
        INX     H
        DCR     A
        JNZ     BA0
        LXI     B,NA2   ;GATE TYPES ARE DETERMINED BY
        LXI     D,BA5   ;A STRING COMPARISON TO THE
BA1:    LHLD    START   ;SOURCE PROGRAM
BA2:    CALL    FNDP    ;LOOK FOR DECIMAL POINT
        JNC     BA4     ;THE CONTROL WORD
        CALL    FNDCH   ;GET FIRST CHARACTER
        PUSH    D
        LDAX    D
        CMP     M       ;COMPARE TO TEST STRING
        JNZ     BA3     ;NOT FOUND CONTINUE SCAN
        INX     H
        INX     D       ;NEXT CHARACTER
        LDAX    D
        CMP     M       ;COMPARE NEXT CHARACTER
        JNZ     BA3     ;IF NO MATCH TRY AGAIN
        CALL    FNDS    ;STILL GOOD FIND SLACH
        INX     D
        LDAX    D
        CMP     M       ;COMPARE # OF INPUTS
        JNZ     BA3     ;NOT THE SAME KEEP LOOKING
        CALL    FNDP    ;FIND END OF CONTROL WORD
        POP     D
        LDAX    B       ;GATE COUNT
        INR     A       ;INCREMENT COUNT
        STAX    B       ;SAVE THE NEW COUNT
        JMP     BA2     ;LOOK FOR ANOTHER ONE
BA3:    POP     D
        CALL    FNDP    ;NO GOOD LOOK FOR NEXT ONE
        JMP     BA2
BA4:    INX     D       ;NEXT TYPE OF GATE
        INX     D
        INX     D
        INX     B
        LDA     WORK    ;GATE COUNT
        DCR     A       ;ONE LESS GATE TO LOOK FOR
        STA     WORK
        CPI     0
        JNZ     BA1     ;ARE ALL GATES DONE
        RET
BA5:    DB      'NA2'   ;STRING COMPARSON DATA
        DB      'NA4AN2AN4OR2OR4'
        DB      'NO2NO4EX2EX4JKF'
;
;
```

```
;
;
;
;*****************************************************************
          ;FUNCTION          ;PRNT
          ;CALLS             ;CRLF,OUTCH,PRBYT
          ;INPUTS            ;START,NEXT
          ;OUTPUTS           ;NOTHING
          ;DESCRIPTION       ;PRNT DOES TWO THINGS
          ;                  ;FIRST FOR DOCUMENTATION
          ;                  ;IT WILL PRINT THE NETWORK
          ;                  ;PROGRAM,THEN IT WILL PRINT
          ;                  ;THE GATE COUNT ,THIS WILL
          ;                  ;HELP TO CONFERM THAT THE
          ;                  ;PROPER NETWORK HAS BEEN
          ;                  ;COMPILED
;*****************************************************************
;
;
;
;
;
PRNT:     MVI       B,5
BB1:      CALL      CRLF      ;CLEAR SCREEN
          DCR       B
          JNZ       BB1
          LXI       H,AA2     ;PRINT DLS TITLES
BB2:      MOV       A,M
          INX       H
          CPI       0
          JZ        BB3
          CALL      OUTCH
          JMP       BB2
BB3:      MVI       B,5
BB4:      CALL      CRLF
          DCR       B
          JNZ       BB4
          LHLD      NEXT      ;LOAD IN SOURCE PROGRAM
          XCHG
          LHLD      START     ;PARAMETERS
BB5:      MOV       A,L
          CMP       E         ;RUN A TEST TO DETERMINE WHEN
          JNZ       BB6       ;THE SOURCE DATA BLOCK HAS BEEN
          MOV       A,H       ;PRINTED OUT
          CMP       D
          JZ        BB7
BB6:      MOV       A,M
          INX       H
          CALL      OUTCH
          CPI       0AH
          JNZ       BB5
```

```
        INX     H       ;STRIP OFF THE FOUR
        INX     H       ;DIGIT LINE INDENTIFIERS
        INX     H
        INX     H
        JMP     BB5
BB7:    MVI     B,5
BB3:    CALL    CRLF
        DCR     B
        JNZ     BB8
        LXI     H,NA2   ;LOAD GATE TYPE COUNT
        LXI     B,BB12  ;GET GATE TITLE TO MATCH
BB9:    LDAX    B
        CPI     '!'     ;INDICATES END OF ROUTINE
        RZ              ;IF FOUND IN PRINT CYCLE
        CPI     '?'     ;INDICATES END OF THAT GATE TYPE
        JNZ     BB11
        INX     B
        MVI     A,99H
        MOV     E,M
        INR     E
        INX     H
BB10:   ADI     1       ;CONVERTS HEX TO DECIMAL
        DAA
        DCR     E
        JNZ     BB10
        CALL    PRBYT   ;PRINT THAT GATE COUNT
        DCR     D
        RZ
        CALL    CRLF    ;MOVE ON TO NEXT GATE TYPE
        JMP     BB9
BB11:   CALL    OUTCH   ;PRINT GATE TITLE
        INX     B
        JMP     BB9
BB12:   DB      'NAN'
        DB      'D/2=?NAND/4=?AND/2 =?AND/4 =?OR/2  =?'
        DB      'OR/4  =?NOR/2 =?NOR/4 =?EXOR/2=?EXOR/4=?!'
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;SYMB
        ;CALLS           ;FNDP,SYEX,SYST
        ;INPUTS          ;NEXT,START
        ;OUTPUTS         ;SYMBS,SYMBE,NUMB
        ;DESCRIPTION     ;SYMB SCANS THROUGH THE SOURCE
        ;                ;PROGRAM AND FINDS A SYMBOL.
        ;                ;IT THEN LOOKS TO SEE IF IT
        ;                ;ALREADY IS IN THE SYMBOL
        ;                ;TABLE,IF IT IS THEN NOTHING
```

```
;                       ;IS DONE, IF IT IS NOT THEN
;                       ;THE SYMBOL WILL BE PUT INTO
;                       ;THE TABLE.
;*****************************************************************
;
;
;
;
;
SYMB:   LHLD    NEXT    ;LOAD END OF SOURCE PROGRAM
        MVI     M,'@'
        INX     H
        SHLD    SYMBS   ;SETUP THE DEMENSIONS OF
        SHLD    SYMBE   ;THE SYMBOL TABLE
        LXI     H,0
        SHLD    NUMB    ;COUNT OF THE # OF SYMBOLS
        LHLD    START   ;START SYMBOL SEARCH
BC1:    CALL    FNDP
        RNC
        CALL    FNDP
BC2:    MOV     A,M     ;FIND OUT IF THE FIRST
        INX     H       ;CHARACTER IS A SYMBOL OR
        CPI     0DH     ;A CONTROL CHARACTER
        JZ      BC1
        CPI     '@'
        RZ
        CPI     'A'
        JM      BC2
        CPI     'Z'+1
        JP      BC2
        DCX     H
        CALL    SYEX    ;SEE IF SYMBOL ALREADY EXISTS
        JC      BC2     ;IF FOUND MOVE ONTO NEXT SYMBOL
        CALL    SYST    ;PUT NEW SYMBOLS INTO TABLE
        JMP     BC2
;
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;SYEX
        ;CALLS           ;NOTHING
        ;INPUTS          ;SYMBE,SYMBS,CARRY FLAG
        ;OUTPUTS         ;WORK
        ;DESCRIPTION     ;SYEX SEARCHES THROUGH THE
        ;                ;SYMBOL TABLE TO DETERMINE
        ;                ;IF A GIVEN SYMBOL ALREADY
        ;                ;EXISTS IN THE TABLE. IF IT
        ;                ;DOES THEN IT WILL STORE
        ;                ;THE ADDRESS OF THE SYMBOL
```

```
          ;                    ;IN WORK AND SET THE CARRY
          ;                    ;FLAG, IF NO MATCH THEN THE
          ;                    ;CARRY FLAG WILL BE RESET
;*****************************************************************
;
;
;
;
;
SYEX:     PUSH     H
          POP      B
          LHLD     SYMBE     ;END OF SYMBOL TABLE
          XCHG
          LHLD     SYMBS     ;BEGINNING OF SYMBOL TABLE
BD1:      MOV      A,L
          CMP      E
          JNZ      BD2       ;TEST TO SEE IF THE WHOLE
          MOV      A,H       ;TABLE HAS BEEN SCANED
          CMP      D
          JNZ      BD2
          PUSH     B
          POP      H
          STC
          CMC                ;NO FIND RESET CARRY FLAG
          RET
BD2:      PUSH     D
          PUSH     B
          PUSH     H
          MVI      D,5       ;SYMBOLS ARE 5 CHARACTERS LONG
BD3:      MOV      A,M       ;GET THE FIRST CHARACTER
          CPI      0         ;TEST TO SEE IF SYMBOL IS LESS
          JZ       BD7       ;THEN 5 CHARACTERS
          LDAX     B
          CMP      M         ;COMPARE TO SYMBOL IN TABLE
          INX      B
          INX      H
          JNZ      BD8       ;NO SYMBOL MATCH GET
          DCR      D         ;NEXT SYMBOL FROM TABLE
          JNZ      BD3
BD4:      POP      D         ;A POSSIBLE MATCH SO FAR
          POP      D
          POP      D
          PUSH     B
          MOV      A,M       ;ALL CHARACTERS MUST MATCH
          CPI      0
          JNZ      $+7
          INX      H
          JMP      BD4+4
          SHLD     WORK      ;STORE ADDRESS OF SYMBOL
          POP      H
BD5:      MOV      A,M
```

```
        CPI     'A'
        JM      BD6
        CPI     'Z'+1
        JP      BD6
        INX     H
        JMP     BD5
BD6:    STC             ;SYMBOL FOUND SET CARRY
        RET
BD7:    LDAX    B
        CPI     'A'
        JM      BD4
        CPI     'Z'+1
        JP      BD4
BD8:    POP     H       ;MOVE SYMBOL POINTER TO
        LXI     B,9     ;NEXT SYMBOL 9 CHARACTERS AWAY
        DAD     B
        POP     B
        POP     D
        JMP     BD1     ;CONTINUE SCAN
;
;
;
;
;*****************************************************************
        ;FUNCTION        :SYST
        ;CALLS           :NOTHING
        ;INPUTS          :SYMBE,NUMB
        ;OUTPUTS         :SYMBE,NUMB
        ;DESCRIPTION     :SYST IS USED TO TAKE A
        ;                :SYMBOL AND STORE IT INTO
        ;                :THE SYMBOL TABLE,ALSO
        ;                :LEAVING SPACE FOR THE TWO
        ;                :ADDRESSES WHICH WILL BE
        ;                :FILLED IN LATER WHEN T1
        ;                :AND T2 ARE FORMED
;*****************************************************************
;
;
;
;
;
SYST:   PUSH    H
        POP     B
        LHLD    SYMBE   ;LATE SYMBOL TABLE ADDRESS
        MVI     D,5     ;5 CHARACTERS TO BE PUT INTO TABLE
BE1:    LDAX    B
        CPI     'A'
        JM      BE3
        CPI     'Z'+1
        JP      BE3
```

```
        MOV     M,A     ;MOVE A CHARACTER INTO TABLE
        INX     H
        INX     B
        DCR     D       ;NEXT CHARACTER
        JNZ     BE1
BE2:    INX     H       ;LEAVE 4 BYTES OPEN
        INX     H       ;FOR THE ADDRESSES
        INX     H
        INX     H
        SHLD    SYMBE   ;NEW END OF SYMBOL TABLE
        LHLD    NUMB
        INX     H       ;INCREMENT SYMBOL COUNT
        SHLD    NUMB
        PUSH    B
        POP     H
        RET
BE3:    MVI     M,0     ;PACK A SYMBOL WITH NULL
        INX     H       ;CHARACTERS WHEN IT IS LESS
        DCR     D       ;5 CHARACTERS IN LENGTH
        JNZ     BE3
        JMP     BE2
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;SETUP
        ;CALLS           ;NOTHING
        ;INPUTS          ;SYMBE,NUMB,T1S,T2S,SYMBS
        ;                ;T1E,T2E
        ;OUTPUTS         ;T1S,T1E,T2S,T2E
        ;DECRIPTION      ;THE SETUP ROUTINE WILL CREAT
        ;                ;THE TWO SIMULATION TABLES. T1
        ;                ;AND T2. ONCE THESE TABLES ARE
        ;                ;MADE THEN SETUP WILL GO BACK
        ;                ;TO THE SYMBOL TABLE AND ASSIGN
        ;                ;EACH SYMBOL A LOCATION IN T1
        ;                ;AND T2
;*****************************************************************
;
;
;
;
;
SETUP:  LHLD    SYMBE
        MVI     M,'@'   ;PUT AN END MARKER ON
        INX     H       ;THE SYMBOL TABLE
        SHLD    T1S     ;START OF T1 TABLE
        XCHG
        LHLD    NUMB    ;THE NUMBER OF BYTES FOR T1
```

```
        INX     H
        INX     H
        PUSH    H
        DAD     D
        SHLD    T1E     ;THE END OF T1 TABLE
        INX     H
        SHLD    T2S     ;START OF T2 TABLE
        POP     D
        DAD     D
        SHLD    T2E     ;THE END OF T2 TABLE
        LHLD    T1S     ;START TO ASSIGN EACH
        PUSH    H       ;SYMBOL AN ADDRESS IN T1 & T2
        POP     B
        LHLD    T2S
        XCHG
        MVI     H,2
BF1:    INX     B       ;THE FIRST TWO BYTES OF T1
        INX     D       ;AND T2 ARE FOR CONSTANTS
        DCR     H
        JNZ     BF1
        LHLD    SYMBS   ;START OF SYMBOL TABLE
BF2:    PUSH    D
        XCHG
        LHLD    SYMBE
        MOV     A,E
        CMP     L
        JNZ     BF3
        MOV     A,D
        CMP     H
        JNZ     BF3
        POP     D
        JMP     BF4
BF3:    XCHG            ;FIND A SYMBOL AND
        LXI     D,5     ;SKIP OVER THE SYMBOL TO
        DAD     D       ;GET TO WHERE THE ADDRESS DATA
        POP     D       ;SHOULD GO
        MOV     M,C     ;B,C<-- CONTAINS T1 ADDRESS
        INX     H
        MOV     M,B
        INX     B
        INX     H
        MOV     M,E     ;D,E<-- CONTAINS T2 ADDRESS
        INX     H
        MOV     M,D
        INX     D
        INX     H
        JMP     BF2     ;GET NEXT SYMBOL
BF4:    LHLD    T1E     ;INILIZE ALL CONTENTS TO X LOGIC
        PUSH    H
        POP     B
        LHLD    T1S
```

```
BF5:    MOV     A,L
        CMP     C
        JNZ     BF6
        MOV     A,H
        CMP     B
        RZ
BF6:    MVI     A,'X'
        MOV     M,A     ;MOVE IT TO T1 TABLE
        INX     H
        JMP     BF5
;
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;PACK
        ;CALLS           ;FNDP,FNDCH,SYEX
        ;INPUTS          ;SIMTE,START,WORK,T1S,T2E
        ;OUTPUTS         ;SIMTS,SIMTE
        ;DESCRIPTION     ;PACK ROUTINE SCANS THROUGH
        ;                ;THE SOURCE PROGRAM LOOGING
        ;                ;FOR ALL THE GATES THEN LOADS
        ;                ;THE SIMULATION UPDATE SEQUENCE
        ;                ;TABLE WITH THE PROPER T1 AND
        ;                ;T2 ADDRESSES.THE UPDATE
        ;                ;SEQUENCE TABLE PERFORMS THE
        ;                ;ACTUAL NETWORK SIMULATION.
;*****************************************************************
;
;
;
;
;
PACK:   LHLD    T2E     ;END OF TABLE T2
        INX     H
        SHLD    SIMTS   ;PARAMETERS OF UPDATE SEQUENCE
        SHLD    SIMTE   ;SIMULATION TABLE
        LXI     B,BG16  ;COMPARISON STRING
        LHLD    START   ;SOURCE PROGRAM
        MVI     A,2     ;FIRST SCAN FOR ALL TWO
        STA     WORK+2  ;INPUT TYPE GATES
        PUSH    B
BG1:    CALL    FNDP    ;GET KEY CHARACTER
        JNC     BG5
        CALL    FNDCH   ;GET A CHARACTER AND COMPARE
BG2:    LDAX    B       ;IT TO THE TEST PATTERN STRING
        CPI     '?'     ;KEY TO SWITCH TO 4 INPUT TYPES
        JZ      BG4
        CPI     '*'     ;END OF TEST PATTERNS
        JZ      BG5
```

```
        CPI     ',';TEST END OF ALPHABETIC STRING
        JZ      BG8
        CPI     '!';COMPARE # OF INPUTS NOW
        JZ      BG3
        CMP     M       ;COMPARE STRING TO MEMORY
        JZ      BG7     ;SO LETS GO
        POP     B       ;TESTS FAILED TRY NEXT GATE TYPE
        PUSH    B
        CALL    FNDP
        JMP     BG1
BG3:    CALL    FNDS    ;GET SOURCE GATE # OF INPUTS
        INX     B
        JMP     BG2     ;GO BACK FOR COMPARISON
BG4:    POP     B
        INX     B
        PUSH    B
        MVI     A,4     ;SWITCH TO 4 INPUT GATE TYPES
        STA     WORK+2
        JMP     BG2     ;RESTART SCAN
BG5:    LHLD    START
        POP     B
BG6:    LDAX    B
        INX     B
        CPI     '*'     ;ALL DONE RETURN
        RZ
        CPI     ','     ;NEXT GATE TYPE INDICATOR
        JNZ     BG6
        PUSH    B
        JMP     BG1
BG7:    INX     B       ;ONCE THE GATE IS FOUND TO
        INX     H       ;MATCH THEN THE INPUT AND
        JMP     BG2     ;OUTPUT ADDRESSES FROM T1 AND T2
BG8:    LDA     WORK+2  ;ARE LOADED INTO THE UPDATE
        MOV     B,A     ;SEQUENCE TABLE
        CALL    FNDP
BG9:    MOV     A,M
        CPI     '1'     ;A LOGIC ONE CONSTANT
        JZ      BG14
        CPI     '0'     ;A LOGIC ZERO CONSTANT
        JZ      BG13
        CPI     'A'
        JM      BG10-4
        CPI     'Z'+1
        JM      BG10    ;SYMBOLS ARE FOUND AND THERE
        INX     H       ;ADDRESSES ARE PUT INTO THE TABLE
        JMP     BG9
BG10:   PUSH    B
        CALL    SYEX    ;GET THE SYMBOL ADDRESS
        XCHG
        LHLD    WORK
        XCHG
```

```
        POP     B
        MOV     A,B
        CPI     0           ;TEST TO SEE IF ALL SYMBOLS
        JZ      BG12        ;FOR THAT GATE WERE DONE
BG11:   PUSH    H
        DCR     B
        LHLD    SIMTE       ;LOAD END OF TABLE
        LDAX    D           ;MOVE THE NEW SEQUENCE
        MOV     M,A         ;INTO THE TABLE
        INX     D
        INX     H
        LDAX    D
        MOV     M,A
        INX     H
        SHLD    SIMTE       ;NEW END OF TABLE
        POP     H
        MOV     A,B
        CPI     OFFH
        JNZ     BG9
        POP     B
        PUSH    B
        MOV     A,M
        INX     H
        CPI     ODH         ;SCAN FOR END OF LINE
        JZ      BG1
        CPI     '0'         ;SCAN FOR INITIAL CONDITIONS
        JZ      $+8
        CPI     '1'
        JNZ     $-14
        PUSH    H
        PUSH    D
        LHLD    WORK        ;ADDRESSES OF WHERE INITIAL
        MOV     E,M         ;CONDITIONS GO
        INX     H
        MOV     D,M
        STAX    D
        POP     D
        POP     H
        JMP     BG1
BG12:   INX     D
        INX     D
        JMP     BG11
BG13:   LXI     D,T1S       ;ADDRESS OF LOGIC ZERO
        INX     H
        JMP     BG11
BG14:   PUSH    H
        LHLD    T1S
        INX     H           ;ADDRESS OF LOGIC ONE
        SHLD    WORK
        LXI     D,WORK
        POP     H
```

```
        INX     H
        JMP     BG11
BG16:   DB      'NA!2,AN!2,OR!2,NO!2,'
        DB      'EX!2,?NA!4,AN!4,OR!4,'
        DB      'NO!4,EX!4,JK!F,*'
;
;
;
;
;
;*****************************************************************
        ;FUNCTION       :IO
        ;CALLS          :FNDP,FNDCH,SYEX
        ;INPUTS         :SIMTE,START,WORK
        ;OUTPUTS        :INP,OUTP,INST
        ;DESCRIPTION    :IO ROUTINE SEARCHES THROUGH THE
        ;               :SOURCE PROGRAM LOOKING FOR THE
        ;               :PRIMARY INPUTS AND THE OUTPUT
        ;               :VARIABLES. CONTROL WORD .INPUT.
        ;               :AND .PRINT. ARE SEARCHED FOR
        ;               :AND THE SYMBOLS WHICH FOLLOW
        ;               :ARE COMPARED TO THE SYMBOL
        ;               :TABLE FOR THERE ADDRESSES.
        ;               :THESE ADDRESSES ARE PUT INTO
        ;               : THE TWO NEW TABLES INP AND OUTP.
;*****************************************************************
;
;
;
;
;
IO:     LHLD    SIMTE
        INX     H
        SHLD    INP     ;BEGINNING OF INP TABLE
        XCHG
        LHLD    START
BH1:    CALL    FNDP    ;FIND KEY SYMBOL
        JNC     BH6
        CALL    FNDCH   ;GET THE FIRST CHARACTER
        MOV     A,M
        CPI     'I'     ;IS IT INPUT CONTROL WORD
        JZ      BH2
        CALL    FNDP
        JMP     BH1
BH2:    CALL    FNDP
BH3:    MOV     A,M     ;FIND A CHARACTER WHICH
        INX     H       ;INDICATES THE BEGINNING OF A
        CPI     0DH     ;VARIABLE SYMBOL
        JZ      BH1
        CPI     'A'
        JM      BH3
```

```
        CPI     'Z'+1
        JP      BH3
        DCX     H
        PUSH    D
        CALL    SYEX    ;GET THE ADDRESS OF SYMBOL
        POP     D
        PUSH    H
        LHLD    WORK
        MVI     B,5
BH4:    DCX     H
        DCR     B
        JNZ     BH4
        MVI     B,7
BH5:    MOV     A,M     ;MOVE ADDRESS INTO INP TABLE
        CPI     0
        JNZ     $+5
        MVI     A,20H
        STAX    D
        INX     H
        INX     D
        DCR     B
        JNZ     BH5
        POP     H
        JMP     BH3     ;MOVE ON TO NEXT SYMBOL
BH6:    XCHG
        MVI     M,'*'   ;INDICATES END OF INP TABLE
        INX     H
        SHLD    OUTP    ;BEGINNING OF OUTPT TABLE
        XCHG
        LHLD    START
BH7:    CALL    FNDP    ;START OUTPUT SCAN
        JNC     BH12
        CALL    FNDCH
        MOV     A,M
        CPI     'P'     ;LOOK FOR ,PRINT,
        JZ      BH8
        CALL    FNDP
        JMP     BH7
BH8:    CALL    FNDP
BH9:    MOV     A,M     ;FOUND,NOW TO THE SYMBOLS
        INX     H
        CPI     0DH
        JZ      BH7
        CPI     'A'
        JM      BH9
        CPI     'Z'+1
        JP      BH9
        DCX     H
        PUSH    D
        CALL    SYEX    ;GET THE ADDRESS FOR THE SYMBOL
        POP     D
```

```
        PUSH    H
        LHLD    WORK
        MVI     B,5
BH10:   DCX     H
        DCR     B
        JNZ     BH10
        MVI     B,7
BH11:   MOV     A,M     ;MOVE ADDRESS INTO OUTP TABLE
        CPI     0
        JNZ     $+5
        MVI     A,20H
        STAX    D
        INX     H
        INX     D
        DCR     B
        JNZ     BH11
        POP     H
        JMP     BH9
BH12:   XCHG
        MVI     M,'*'   ;END OF OUTP TABLE INDICATOR
        INX     H
        SHLD    INST
        RET
```

## 4.7 The DLS Executer

The executer module has the function of performing the actual simulation of the digital logic circuit. It takes the data which the compiler creates and interprets it to form a simulation of the users network. Aside from the source program other information is required by DLS to carry out the simulation.

When the user issues the execute command (EXEC) the first piece of information which is required is the number of update cycles per clock cycle. This is for race condition testing. For example if the user informs DLS that there will be seven update time units per clock cycle and during the simulation it takes the network eight time units for it to reach a stable state, a race condition would exist. The second piece of information is the number of test input patterns. The simulated network has a certain number of primary inputs. The user must tell DLS how many test patterns should be put through the simulated circuit. The third piece of information concerns the mode settings during simulation. The first choice is between the normal and trace modes. The normal mode will print the logic values of the monitored outputs after each clock cycle. In the trace mode a printing will be made after every update cycle. This aids in viewing certain hazard conditions. The second mode choice is between two value

simulation and three value simulation. In the two value simulation only the logic '0' and logic '1' are used. In the three value simulation the logic 'X' is used in the update cycle where each gate when changing uses it as the transition logic value. This helps in detecting certain possible hazard conditions.

The last thing which the executer requests is the test input patterns. Each primary symbol is printed and then the user types in the test pattern for that symbol. This is done for each primary input until the whole test pattern string has been loaded.

The actual executer module is comprised of twenty one separate routines. For simplicity these routines are described by four flowcharts. Figure 4-13 is the EXEC routine flowchart. This encompasses the controlling part of the executer. It has the job of calling the proper routines to first get the needed information from the user and then controlling the simulation process.

The EXEC routine calls upon the UPDAT (Figure 4-14) routine to perform the operation of logic simulation. This is done by manipulation of the data in the two simulation tables, T1 and T2, UPDAT passes the proper data from T1 to each gate simulation routine, which performs its operation then puts the returning data into T2. This makes up the update cycle, which is done until a stable state is reached. The other thing UPDAT does is when a hazard

Exec Function
Routine

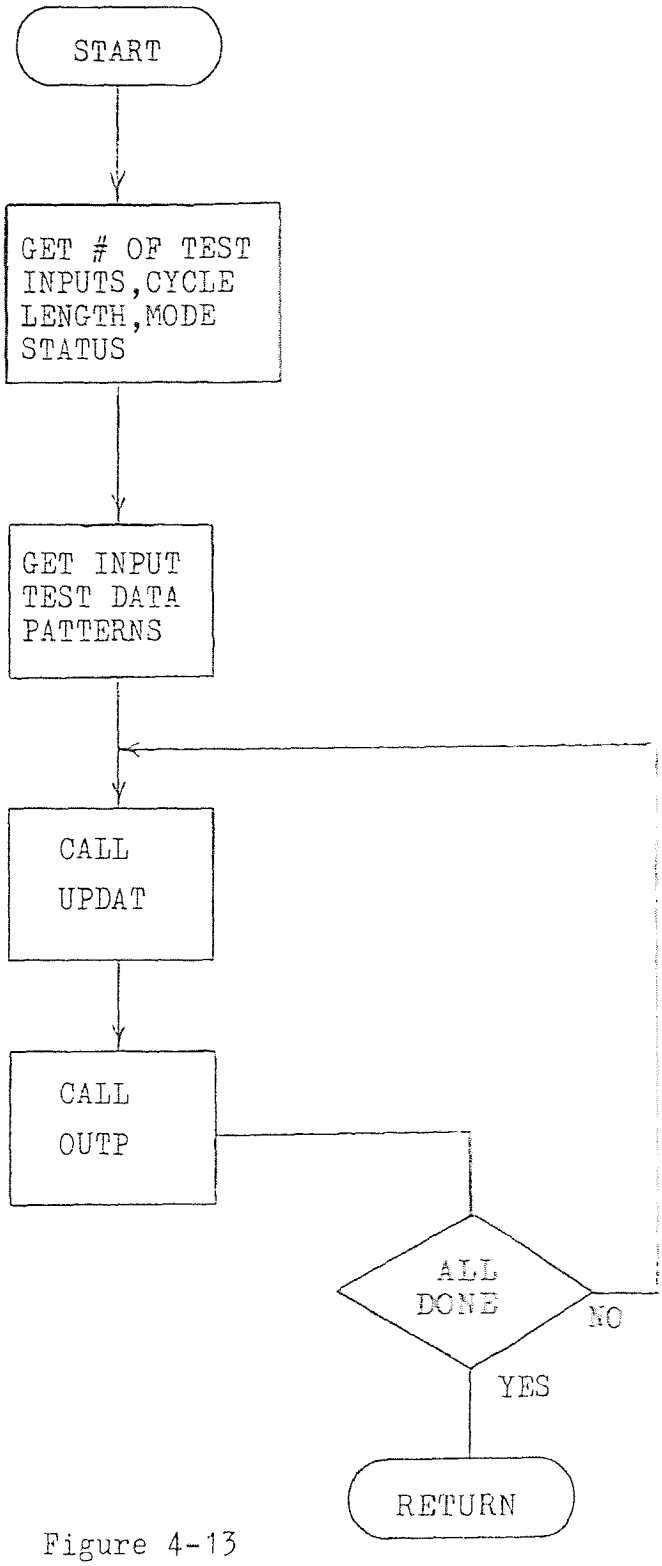
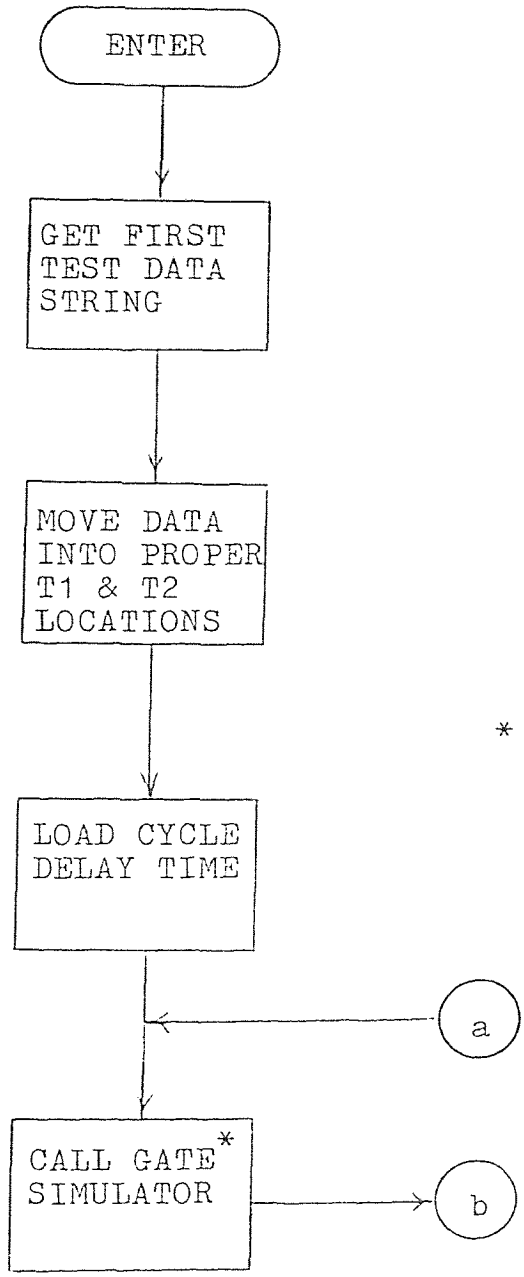



Figure 4-13

Updat Function
Routine
(Part 1)




* this subroutine really is 10 different routines, one per gate

Figure 4-14

Updat Function
Routine
(Part 2)

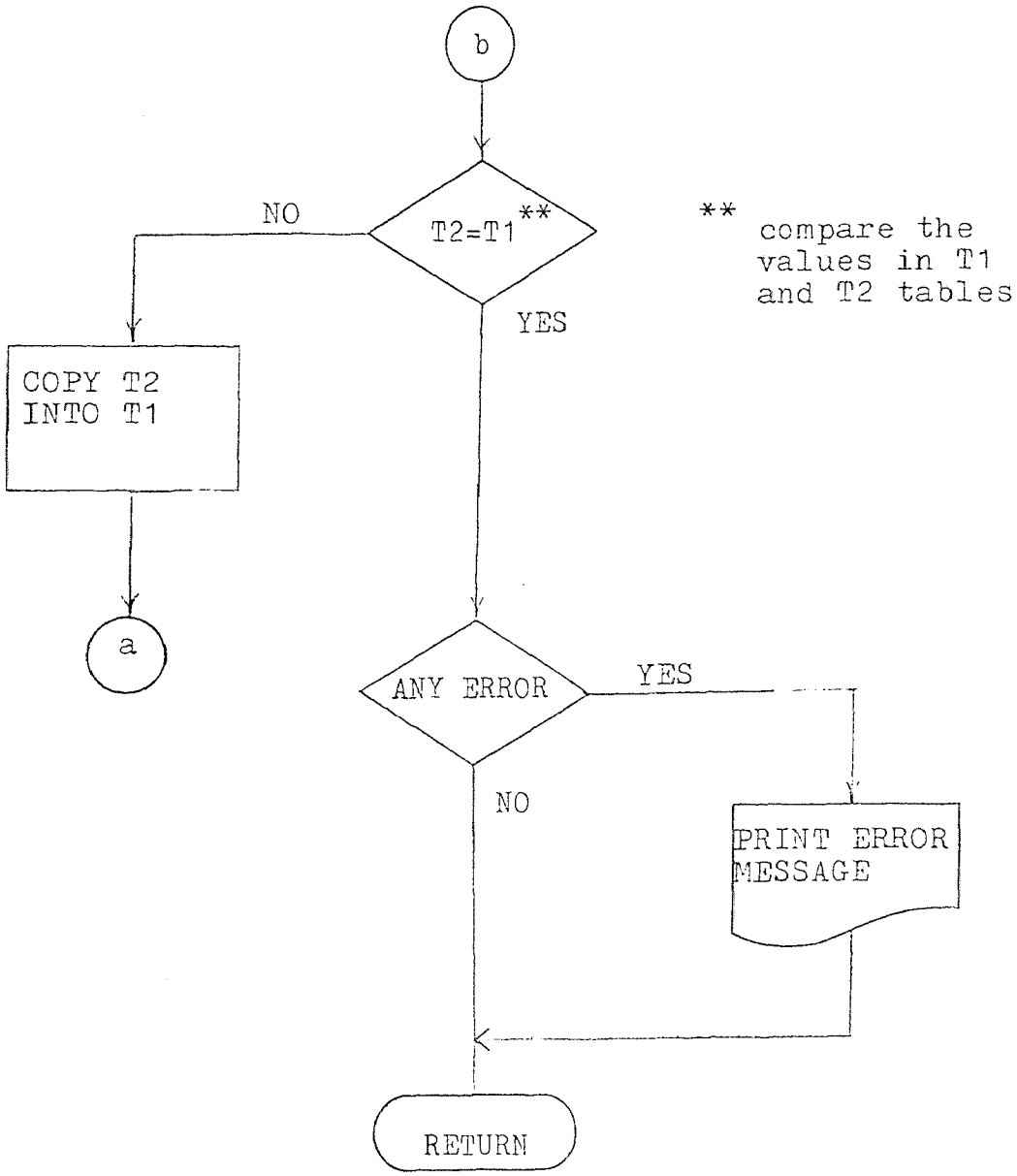



Figure 4-14

Gate Simulation
Routine
(Part 1)

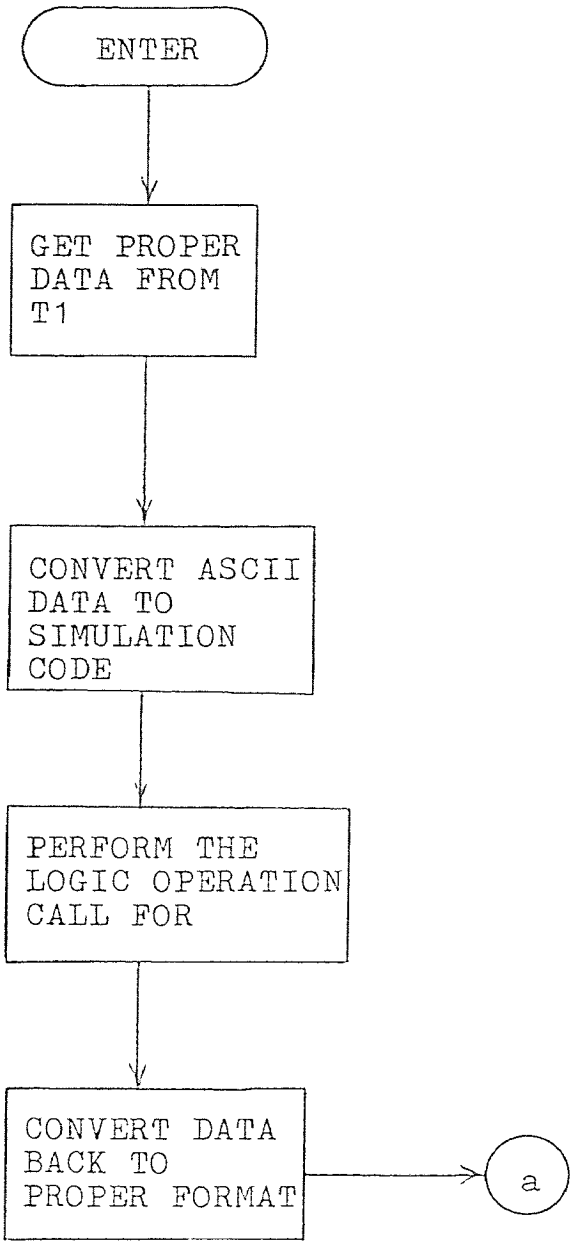



Figure 4-15

Gate Simulation
Routine
(Part 2)

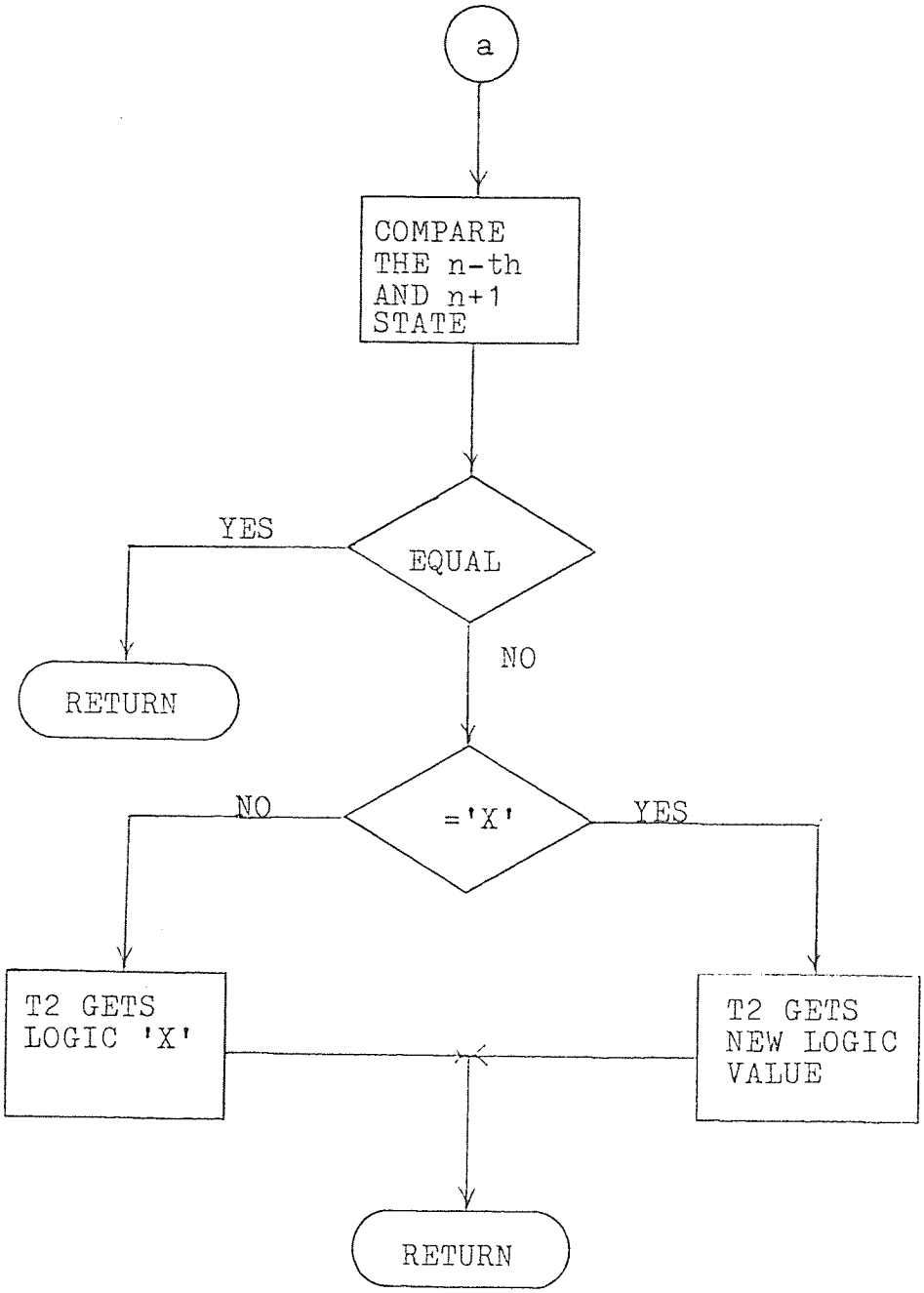



Figure 4-15

Outp Function
Routine

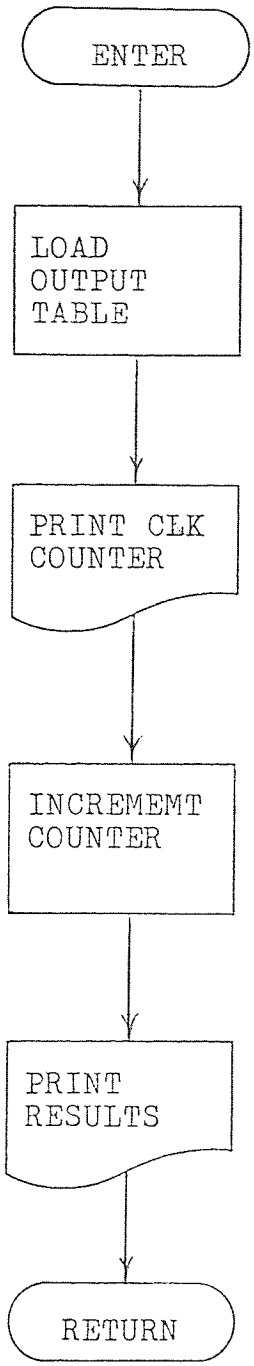



Figure 4-16

condition has been detected it informs the user what type of hazard had arisen during the simulation.

There are ten types of logic gates which DLS has in its gate library, each of these gates has its own routine. Figure 4-15 is the general flowchart for a logic gate module. The data which comes from T1 into the gate routine is first converted into a different format for operation in the routine. After the logic operation is performed x-pass analysis is done, only if x-pass mode of operation was chosen. X-pass only operates when three value simulation is in operation.

The last flowchart (Figure 4-16) is the OUTPT routine. After all updating is done for each time cycle the monitored output variables will be printed. If the trace mode was used then OUTPT would be called upon after every update cycle.

## 4.8 The Executer Program Listing

```
;*************************************************************
        ;FUNCTION       :EXEC
        ;CALLS          :CRLF,OUTCH,GETDM,TRACE,GETCH
        ;               :TITL,UPDAT,OUTPT
        ;INPUTS         :T1S,T2S,INP,INST,TEST,PLACE
        ;OUTPUTS        :DELAY,TEST,PLACE,WORK
        ;DESCRIPTION    :THE EXEC ROUTINE IS THE
        ;               :CONTROLLING SUB-MODULE OF THE
        ;               :EXECUTER. EXEC GETS THE
        ;               :INFORMATION ON THE MODES OF
        ;               :OPERATION AND THE TEST INPUT DATA
        ;               :AND PRODUCES SIMULATED NETWORKS.
        ;               :THE TWO IMPORTANT SUB-SUB-MODULES
        ;               :WHICH EXEC CALLS UPON 'UPDAT' AND
        ;               :'OUTP'. THE FIRST MAKES ONE UPDATE
        ;               :PASS THROUGH THE NETWORK AND THE
        ;               :LATTER PRINTS OUT THE RESULTS.
;*************************************************************
;
;
;
;
;
EXEC:   LXI     H,CA2   ;PRINT THE MESSAGE TO FIND THE
        CALL    CRLF    ;NUMBER OF CLOCK UPDATE CYLES
        CALL    CRLF    ;PER UNIT OF TIME
        PUSH    H
        LHLD    T1S     ;LOAD THE ADDRESSES OF T1S AND
        XCHG            ;TS2
        LHLD    T2S
        MVI     A,'0'   ;STORE TWO CONSTANTS ;LOGIC '0'
        STAX    D
        MOV     M,A
        INR     A       ;AND LOGIC '1' IN THE FIRST
        INX     D       ;TWO LOCATIONS IN T1 AND T2
        INX     H
        STAX    D
        MOV     M,A
        POP     H
CA1:    MOV     A,M     ;INIATE PRINTING
        INX     H
        CPI     '?'
        JZ      CA3
        CALL    OUTCH
        JMP     CA1
CA2:    DB      '# OF TIME UNITS PER PULSE=?'
CA3:    MVI     B,0
CA4:    CALL    GETDM   ;GET A DECIMAL NUMBER FROM
        MOV     A,B     ;USER INDICATING THE # OF UNITS PER
        STA     DELAY   ;PULSE FOR DELAY ANALYSIS
        MVI     B,0
```

```
        LXI     H,CA6   ;LOAD NEXT MESSAGE
        CALL    CRLF
CA5:    MOV     A,M
        INX     H
        CPI     '?'
        JZ      CA7
        CALL    OUTCH   ;GET THE NUMBER OF TEST INPUTS
        JMP     CA5     ;FROM THE USER
CA6:    DB      '# OF TEST INPUTS=?'
CA7:    CALL    GETDM   ;GET THE DECIMAL NUMBER
        MOV     A,B
        STA     TEST    ;SAVE FOR LATER USE
        CALL    CRLF
        CALL    TRACE   ;FIND OUT IF TAACE MODE IS WANTED
        CALL    CRLF
        CALL    CRLF
        CALL    CRLF
        LHLD    INP     ;ADDRESS OF INPUT TEST STRING
        XCHG
        LHLD    INST    ;A TEMPORY TABLE CONSISTS OF ALL
CA8:    MVI     C,5     ;THE TEST INPUT PATTERNS
        LDA     TEST    ;SIZE OF THE TABLE
        MOV     B,A
CA9:    LDAX    D
        CPI     '*'     ;FIND IF ALL THE TEST INPUT
        JZ      CA11    ;DATA HAS BEEN INPUTED
        CPI     0
        JNZ     $+5
        MVI     A,20H
        CALL    OUTCH   ;PRINT THE TEST INPUT SYMBOL
        INX     D
        DCR     C
        JNZ     CA9     ;SYMBOLS ARE ALL 5 CHARACTERS LONG
        INX     D
        INX     D
        MVI     A,';'   ;FOLLOWED BY A PROMPT
        CALL    OUTCH
CA10:   CALL    GETCH   ;USER ENTERS TEST INPUT VALUES
        MOV     M,A     ;SAVE IN INPUT STRING TABLE
        INX     H
        DCR     B       ;KEEP TRACK ON COUNT
        JNZ     CA10
        CALL    CRLF    ;ALL DONE FOR THAT INPUT
        JMP     CA8     ;MOVE ON TO NEXT INPUT
CA11:   CALL    TITL    ;PRINT THE TITLE OF MONOTORED SYMBOL
        MVI     A,0
        STA     PLACE   ;KEEP TRACK OF # OF UPDATES
        STA     WORK+2
CA12:   CALL    UPDAT   ;MAKE ONE UPDATE SEQUENCE PASS
        CALL    OUTPT   ;PRINT RESULTS
        LDA     TEST    ;COMPUTE # OF TEST INPUT
```

```
          MOV     B,A
          LDA     PLACE     ;POINT TO PLACE IN TABLE
          CMP     B
          JC      CA12      ;NOT DONE DO ANOTHER UPDATE
          CALL    CRLF
          CALL    CRLF
          CALL    CRLF
          JMP     AA3       ;RETURN ALL DONE FOR NOW
;
;
;
;
;*****************************************************************
          ;FUNCTION          ;TITL
          ;CALLS             ;CRLF,OUTCH
          ;INPUTS            ;INST,OUTP
          ;OUTPUTS           ;NOTHING
          ;DESCRIPTION       ;THE TITL ROUTINE PRINTS ALL
          ;                  ;THE VARIABLE SYMBOLS WHICH
          ;                  ;THE USER REQUESTED IN A EASLY
          ;                  ;READABLE FORMAT
;*****************************************************************
;
;
;
;
TITL:     CALL    CRLF      ;CARRAGE RETURN AND LINE FEED
          CALL    CRLF
          MVI     B,5
          MVI     A,20H     ;SPACE OVER AWAY FROM
CB1:      CALL    OUTCH     ;THE EDGE OF THE PAPER
          DCR     B
          JNZ     CB1
          MVI     B,5
          LHLD    INST      ;THIS ADDRESS MARKS THE END OF THE
          XCHG              ;OUTP TABLE
          LHLD    OUTP      ;THIS TABLE HAS THE LIST OF ALL THE
          DCX     D         ;SYMBOLS WHICH ARE TO BE PRINTED
          PUSH    H
CB2:      MOV     A,L       ;DETERMINE IF ALL THE
          CMP     E         ;SYMBOLS HAVE BEEN PRINTED
          JC      CB4
          MOV     A,H       ;A TRICK IS DONE HERE WHERE
          CMP     D         ;ALL SYMBOLS ARE PRINTED VERTICALY
          JC      CB4       ;THIS IS DONE BY PRINTING ALL THE
          POP     H         ;FIRST CHARACTERS OF EACH SYMBOL
          INX     H         ;THEN A CRLF AND PRINTING THE
          PUSH    H         ;NEXT CHARACTER OF EACH SYMBOL
          DCR     B         ;AND SO ON FOR THE REST
```

```
        JZ      CB6
        CALL    CRLF
        MVI     C,5
        MVI     A,20H
CB3:    CALL    OUTCH
        DCR     C
        JNZ     CB3
CB4:    MOV     A,M
CB5:    CALL    OUTCH   ;GET A CHARACTER FROM ONE OF THE
        MVI     A,20H   ;SYMBOLS,PRINT IT THEN INCERT
        CALL    OUTCH   ;TWO SPACES BEFORE NEXT CHARACTER
        CALL    OUTCH   ;IS PRINTED
        PUSH    D
        LXI     D,7     ;EACH SYMBOL IS 7 LOCATIONS
        DAD     D       ;AWAY FROM EACH OTHER
        POP     D
        JMP     CB2     ;KEEP THIS PROCESS GOING
CB6:    POP     H
        CALL    CRLF    ;WHEN ALL SYMBOLS HAVE BEEN PRINTED
        MVI     B,60    ;PRINT OUT A SOLID LINE WHICH
        MVI     A,'-'   ;SEPERATES SYMBOLS FROM UPCOMING DATA
CB7:    CALL    OUTCH
        DCR     B
        JNZ     CB7
        CALL    CRLF
        RET
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;OUTPT
        ;CALLS           ;PRBYT,OUTCH,OSSL
        ;INPUTS          ;OUTP,ERROR,WORK
        ;OUTPUTS         ;WORK
        ;DESCRIPTION     ;OUTPT ROUTINE PRINTS THE
        ;                ;SIMULATION TABLE T1(MONOTORED
        ;                ;POINTS ONLY) AFTER EACH UPDATE
        ;                ;ALONG WITH THE CLOCK CYCLE COUNT
;*****************************************************************
;
;
;
;
;
OUTPT:  LHLD    OUTP    ;LOAD TABLE
        LDA     ERROR   ;TEST FOR POSIBLE ERRORS
        CPI     'T'
        JZ      TIME
        LXI     B,5
```

```
        LDA     WORK+2  ;UPDATE COUNTER
        PUSH    PSW
        CALL    PRBYT   ;PRINT CLOCK UPDATE COUNTER
        POP     PSW
        ADI     1       ;INCREMENT COUNTER
        DAA
        STA     WORK+2
        MVI     A,':'
        CALL    OUTCH
        MVI     A,20H
        CALL    OUTCH
        CALL    OUTCH
CC1:    MOV     A,M     ;SEARCH THROUGH OUTP TABLE FOR
        CPI     '*'     ;END MARKER
        JNZ     CC2
        LDA     ERROR   ;ERROR TEST
        CPI     '0'
        JNZ     $+6
        CALL    OSSL
        CALL    CRLF
        RET
CC2:    DAD     B       ;PASS OVER SYMBOL
        MOV     E,M     ;TO THE ADDRESS PORTION
        INX     H
        MOV     D,M     ;POINTER TO LOACTION IN T1
        INX     H
        LDAX    D
        CALL    OUTCH   ;PRINT LOGIC VALUE
        MVI     A,20H
        CALL    OUTCH
        CALL    OUTCH
        JMP     CC1
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;UPDAT
        ;CALLS           ;N2,N4,A2,A4,O2,O4,R2,R4
        ;                ;E2,E4,TRAC
        ;INPUTS          ;INP,INST,PLACE,TEST,T2S,T1E
        ;                ;T1S,SYMTS,TRON,COUNT,DELAY
        ;OUTPUTS         ;ERROR,PLACE,COUNT
        ;DESCRIPTION     ;UPDAT ROUTINE HAS THE TASK OF
        ;                ;TAKING ALL THE DATA IN T1,RUNNING
        ;                ;THROUGH THE UPDATE SEQUENCE AND
        ;                ;STORING THE RESULTS IN T2.THERE
        ;                ;ARE TEN LOGIC GATE ROUTINES WHICH
        ;                ;ARE CALLED WHICH ARE USED TO DO
        ;                ;THE UPDATING. IF THE TRACE MODE
```

```
          ;                    ;WAS SELECTED THEN THE RESULTS ARE
          ;                    ;PRINTED AFTER EACH UPDATE.
;*************************************************************
;
;
;
;
;
UPDAT:    MVI       A,0
          STA       ERROR      ;CLEAR ERROR FLAG
          LHLD      INP        ;LOAD INPUT SYMBOL STRING
          PUSH      H
          LHLD      INST       ;LOAD INPUT DATA STRING
          LDA       PLACE      ;FIND WHAT PLACE WE ARE UP TO
          MOV       C,A
          MVI       B,0
          DAD       B          ;GET THE PROPER INPUTS
          INR       A
          STA       PLACE      ;ADD ONE TO PLACE
          XCHG                 ;FOR NEXT UPDATE PASS
          POP       B
CD1:      LDAX      B          ;TEST TO SEE IF THE
          CPI       '*'        ;END OF THE INPUT STRING
          JZ        CD2        ;WAS ENCOUNTERED
          LXI       H,5
          DAD       B
          PUSH      H
          POP       B
          LDAX      B
          MOV       L,A        ;MOVE THE DATA FROM THE INPUT
          INX       B          ;STRING TABLE INTO THE T1
          LDAX      B          ;SIMULATION TABLE.ONCE THIS
          MOV       H,A        ;IS DONE THEN AN UPDATE IS READY
          INX       B          ;TO BE PERFORMED ON THIS TEST
          LDAX      D          ;PATTERN
          MOV       M,A
          XCHG
          MVI       D,0
          LDA       TEST
          MOV       E,A
          DAD       D
          XCHG
          JMP       CD1
CD2:      LHLD      T2S        ;UPDATE IS COMPLETE
          PUSH      H          ;A TEST IS MADE BETWEEN T1 AND
          POP       B          ;T2 TO SEE IF THEY CONTAIN THE
          LHLD      T1E        ;SAME DATA. IF IT DOES NO MORE
          XCHG                 ;UPDATES ARE NEEDED FOR THIS TIME
          LHLD      T1S        ;FRAME.IF THERE IS A DIFFERENCE
CD3:      MOV       A,H        ;THEN A STABLE STATE HAS NOT BEEN
          CMP       D          ;REACHED,ANOTHER UPDATE IS NEEDED.
```

```
        JNZ     CD4
        MOV     A,L
        CMP     E
        JZ      CD5
CD4:    MOV     A,M     ;T1 T2 COMPARISON TEST
        STAX    B
        INX     H
        INX     B
        JMP     CD3
CD5:    MVI     A,0
        STA     COUNT   ;UPDATE COUNTER
CD6:    LHLD    SIMTS   ;START UPDATE SEQUENCE
        CALL    N2
        CALL    N4
        CALL    A2
        CALL    A4
        CALL    O2
        CALL    O4
        CALL    R2
        CALL    R4
        CALL    E2
        CALL    E4
        LDA     TRON    ;TEST TO SEE IF TRACE MODE IS ON
        CPI     'Y'
        CZ      TRAC
        LDA     COUNT   ;UPDATE COUNTER
        INR     A
        JZ      CD13
        STA     COUNT
        MOV     B,A
        LDA     DELAY   ;TEST AGAINST USERS SET DELAY
        INR     A
        CMP     B
        JNZ     CD7
        MVI     A,'T'   ;STORE TIMING ERROR
        STA     ERROR
CD7:    LHLD    T1S     ;LOAD SIMULATION TABLES
        PUSH    H
        POP     B
        LHLD    T2E
        XCHG
        LHLD    T2S
CD8:    MOV     A,H     ;TEST TO SEE IF A COMPLETE SEARCH
        CMP     D       ;THROUGH THE TWO TABLES HAS
        JNZ     CD9     ;BEEN MADE
        MOV     A,L
        CMP     E
        RZ
CD9:    LDAX    B       ;MAKE T1 T2 COMPARISON TEST
        CMP     M
        JNZ     CD10
```

```
        INX     B
        INX     H
        JMP     CD8
CD10:   LHLD    T1S
        PUSH    H
        POP     B
        LHLD    T2E
        XCHG
        LHLD    T2S
CD11:   MOV     A,H
        CMP     D
        JNZ     CD12
        MOV     A,L
        CMP     E
        JZ      CD6
CD12:   MOV     A,M     ;MOVE T1 INTO T2
        STAX    B       ;TO START SIMULATION
        INX     B
        INX     H
        JMP     CD11
CD13:   MVI     A,'0'   ;OSCILLATION ERROR
        STA     ERROR
        LHLD    T2S     ;SEARCH THROUGH THE TWO TABLES
        PUSH    H       ;TO FIND WHERE THEY DIFFER
        POP     B
        LHLD    T1E
        XCHG
        LHLD    T1S
CD14:   MOV     A,L
        CMP     E
        JNZ     CD15
        MOV     A,H
        CMP     D
        RZ
CD15:   LDAX    B
        CMP     M
        JZ      CD16
        MVI     M,'X'   ;PUT LOGIC 'X' IN LOCATIONS WHICH
CD16:   INX     B       ;DIFFER
        INX     H
        JMP     CD14
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;N2
        ;CALLS           ;CONV,RCONV,CHANG
        ;INPUTS          ;NA2
        ;OUTPUTS         ;NOTHING
```

```
          ;DESCRIPTION     ;N2 IS THE TWO INPUT NAND
          ;                ;GATE SIMULATION ROUTINE.
;*************************************************************
;
;
;
;
;
N2:       LDA      NA2     ;# OF 2 INPUT NAND GATES
          ORA      A       ;IF ZERO MOVE TO NEXT GATE TYPE
          RZ
          MOV      C,A
CE1:      CALL     CONV    ;GET THE FIRST INPUT VALUE
          MOV      B,A
          CALL     CONV    ;GET THE NEXT INPUT VALUE
          ANA      B       ;LOGICAL AND
          CMA              ;COMPLEMENT RESULTS
          ANI      03H     ;STRIP OFF UNIMPORTANT INFORMATION
          CPI      1
          JNZ      $+5
          MVI      A,2
          CALL     RCONV
          CALL     CHANG   ;STORE RESULTS AWAY
          DCR      C
          JNZ      CE1     ;SEE IF ALL THESE GATES ARE DONE
          RET
;
;
;
;
;
;*************************************************************
          ;FUNCTION        ;N4
          ;CALLS           ;CONV,RCONV,CHANG
          ;INPUTS          ;NA4
          ;OUTPUTS         ;NOTHING
          ;DESCRIPTION     ;N4 IS THE ROUTINE WHICH
          ;                ;SIMULATES A FOUR INPUT
          ;                ;NAND GATE
;*************************************************************
;
;
;
;
;
N4:       LDA      NA4     ;# OF 4 INPUT NAND GATES
          ORA      A
          RZ
          MOV      C,A     ;SAVE COUNT
CF1:      CALL     CONV    ;FIRST INPUT
          MOV      B,A
```

```
        CALL    CONV    ;SECOND INPUT
        ANA     B       ;LOGICAL AND
        MOV     B,A
        CALL    CONV    ;THIRD INPUT
        ANA     B       ;LOGICAL AND
        MOV     B,A
        CALL    CONV    ;FOURTH INPUT
        ANA     B       ;LOGICAL AND
        CMA             ;COMPLEMENT THE ANSWER
        ANI     03H
        CPI     1
        JNZ     $+5
        MVI     A,2
        CALL    RCONV   ;CONVERT TO PROPER FORMAT
        CALL    CHANG   ;STORE ANSWER AWAY
        DCR     C
        JNZ     CF1     ;DECREMENT GATE COUNT
        RET
;
;
;
;
;*****************************************************************
        ;FUNCTION        :A2
        ;CALLS           :CONV,RCONV,CHANG
        ;INPUTS          :AN2
        ;OUTPUTS         :NOTHING
        ;DESCRIPTION     :A2 IS THE ROUTINE WHICH
        ;                :SIMULATES A TWO INPUT
        ;                :AND GATE
;*****************************************************************
;
;
;
;
;
A2:     LDA     AN2     ;# OF 2 INPUT AND GATES
        ORA     A
        RZ
        MOV     C,A     ;SAVE COUNT
CG1:    CALL    CONV    ;FIRST INPUT
        MOV     B,A
        CALL    CONV    ;SECOND INPUT
        ANA     B       ;LOGICAL AND
        ANI     03H
        CPI     1
        JNZ     $+5
        MVI     A,2
        CALL    RCONV   ;CONVERT TO PROPER FORMAT
        CALL    CHANG   ;STORE AWAY THE ANSWER
```

```
          DCR     C
          JNZ     CG1       ;DECREMENT GATE COUNT
          RET
;
;
;
;
;
;*****************************************************************
          ;FUNCTION         :A4
          ;CALLS            :CONV,RCONV,CHANG
          ;INPUTS           :AN4
          ;OUTPUTS          :NOTHING
          ;DESCRIPTION      :A4 IS THE ROUTINE WHICH
          ;                 :SIMULATES A FOUR INPUT
          ;                 :AND GATE
;*****************************************************************
;
;
;
;
;
A4:       LDA     AN4       ;# OF 4 INPUT AND GATES
          ORA     A
          RZ
          MOV     C,A       ;SAVE COUNT
CH1:      CALL    CONV      ;FIRST INPUT
          MOV     B,A
          CALL    CONV      ;SECOND INPUT
          ANA     B         ;LOGICAL AND
          MOV     B,A
          CALL    CONV      ;THIRD INPUT
          ANA     B         ;LOGICAL AND
          MOV     B,A
          CALL    CONV      ;FOURTH INPUT
          ANA     B         ;LOGICAL AND
          ANI     03H
          CPI     1
          JNZ     $+5
          MVI     A,2
          CALL    RCONV     ;CONVERT TO PROPER FORMAT
          CALL    CHANG     ;STORE AWAY THE ANSWER
          DCR     C
          JNZ     CH1       ;DECREMENT GATE COUNT
          RET
;
;
;
;
;
;*****************************************************************
```

```
          ;FUNCTION          ;O2
          ;CALLS             ;CONV,RCONV,CHANG
          ;INPUTS            ;OR2
          ;OUTPUTS           ;NOTHING
          ;DESCRIPTION       ;O2 IS THE ROUTINE WHICH
          ;                  ;SIMULATES A TWO INPUT
          ;                  ;OR GATE
;*****************************************************************
;
;
;
;
;
O2:       LDA       OR2      ;# OF 2 INPUT OR GATES
          ORA       A
          RZ
          MOV       C,A      ;SAVE GATE COUNT
CI1:      CALL      CONV     ;FIRST INPUT
          MOV       B,A
          CALL      CONV     ;SECOND INPUT
          ORA       B        ;LOGICAL OR
          ANI       03H
          CPI       1
          JNZ       $+5
          MVI       A,2
          CALL      RCONV    ;CONVERT TO PROPER FORMAT
          CALL      CHANG    ;SAVE THE ANSWER
          DCR       C
          JNZ       CI1      ;DECREMENT GATE COUNT
          RET
;
;
;
;
;
;*****************************************************************
          ;FUNCTION          ;O4
          ;CALLS             ;CONV,RCONV,CHANG
          ;INPUTS            ;OR4
          ;OUTPUTS           ;NOTHING
          ;DESCRIPTION       ;O4 IS THE ROUTINE WHICH
          ;                  ;SIMULATES A FOUR INPUT
          ;                  ;OR GATE
;*****************************************************************
;
;
;
;
;
O4:       LDA       OR4      ;# OF 4 INPUT OR GATES
          ORA       A
```

```
        RZ
        MOV     C,A     ;SAVE GATE COUNT
CJ1:    CALL    CONV    ;FIRST INPUT
        MOV     B,A
        CALL    CONV    ;SECOND INPUT
        ORA     B       ;LOGICAL OR
        MOV     B,A
        CALL    CONV    ;THIRD INPUT
        ORA     B       ;LOGICAL OR
        MOV     B,A
        CALL    CONV    ;FOURTH INPUT
        ORA     B       ;LOGICAL OR
        ANI     03H
        CPI     1
        JNZ     $+5
        MVI     A,2
        CALL    RCONV   ;CONVERT TO PROPER FORMAT
        CALL    CHANG   ;STORE AWAY THE ANSWER
        DCR     C
        JNZ     CJ1     ;DECREMENT GATE COUNT
        RET
;
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;R2
        ;CALLS           ;CONV,RCONV,CHANG
        ;INPUTS          ;NO2
        ;OUTPUTS         ;NOTHING
        ;DESCRIPTION     ;R2 IS THE ROUTINE WHICH
        ;                ;SIMULATES A TWO INPUT
        ;                ;NOR GATE
;*****************************************************************
;
;
;
;
;
R2:     LDA     NO2     ;# OF 2 INPUT NOR GATES
        ORA     A
        RZ
        MOV     C,A     ;SAVE GATE COUNT
CK1:    CALL    CONV    ;FIRST INPUT
        MOV     B,A
        CALL    CONV    ;SECOND INPUT
        ORA     B       ;LOGICAL OR
        CMA             ;COMPLEMENT ANSWER
        ANI     03H
        CPI     1
```

```
        JNZ     $+5
        MVI     A,2
        CALL    RCONV   ;CONVERT TO PROPER FORMAT
        CALL    CHANG   ;STORE AWAY THE ANSWER
        DCR     C
        JNZ     CK1     ;DECREMENT GATE COUNT
        RET
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;R4
        ;CALLS           ;CONV,RCONV,CHANG
        ;INPUTS          ;NO4
        ;OUTPUTS         ;NOTHING
        ;DESCRIPTION     ;R4 IS THE ROUTINE WHICH
        ;                ;SIMULATE A FOUR INPUT
        ;                ;NOR GATE
;*****************************************************************
;
;
;
;
;
R4:     LDA     NO4     ;# OF 4 INPUT NOR GATES
        ORA     A
        RZ
        MOV     C,A     ;SAVE GATE COUNT
CL1:    CALL    CONV    ;FIRST INPUT
        MOV     B,A
        CALL    CONV    ;SECOND INPUT
        ORA     B       ;LOGICAL OR
        MOV     B,A
        CALL    CONV    ;THIRD INPUT
        ORA     B       ;LOGICAL OR
        MOV     B,A
        CALL    CONV    ;FOURTH INPUT
        ORA     B       ;LOGICAL OR
        CMA             ;COMPLEMENT ANSWER
        ANI     03H
        CPI     1
        JNZ     $+5
        MVI     A,2
        CALL    RCONV   ;CONVERT TO PROPER FORMAT
        CALL    CHANG   ;STORE AWAY THE ANSWER
        DCR     C
        JNZ     CL1     ;DECREMENT GATE COUNT
        RET
```

```
;
;
;
;
;*****************************************************************
          ;FUNCTION          ;E2
          ;CALLS             ;CONV,RCONV,CHANG
          ;INPUTS            ;EX2
          ;OUTPUTS           ;NOTHING
          ;DESCRIPTION       ;E2 IS THE ROUTINE WHICH
          ;                  ;SIMULATES A TWO INPUT
          ;                  ;EXOR GATE
;*****************************************************************
;
;
;
;
;
E2:       LDA       EX2       ;# OF 2 INPUT EXOR GATES
          ORA       A
          RZ
          MOV       C,A       ;SAVE GATE COUNT
CM1:      CALL      CONV      ;FIRST INPUT
          MOV       B,A
          CALL      CONV      ;SECOND INPUT
          XRA       B         ;LOGICAL EXOR
          ANI       03H
          CPI       1
          JNZ       $+5
          MVI       A,2
          CALL      RCONV     ;CONVERT TO PROPER FORMAT
          CALL      CHANG     ;STORE AWAY THE ANSWER
          DCR       C
          JNZ       CM1       ;DECREMENT THE GATE COUNT
          RET
;
;
;
;
;
;*****************************************************************
          ;FUNCTION          ;E4
          ;CALLS             ;CONV,RCONV,CHANG
          ;INPUTS            ;EX4
          ;OUTPUTS           ;NOTHING
          ;DESCRIPTION       ;EX4 IS THE ROUTINE WHICH
          ;                  ;SIMULATES A FOUR INPUT
          ;                  ;EXOR GATE
;*****************************************************************
;
;
```

```
;
;
;
E4:     LDA     EX4     ;# OF 4 INPUT EXOR GATES
        ORA     A
        RZ
        MOV     C,A     ;SAVE GATE COUNT
CN1:    CALL    CONV    ;FIRST INPUT
        MOV     B,A
        CALL    CONV    ;SECOND INPUT
        XRA     B       ;LOGICAL EXOR
        MOV     B,A
        CALL    CONV    ;THIRD INPUT
        XRA     B       ;LOGICAL EXOR
        MOV     B,A
        CALL    CONV    ;FOURTH INPUT
        XRA     B       ;LOGICAL EXOR
        ANI     03H
        CPI     1
        JNZ     $+5
        MVI     A,2
        CALL    RCONV   ;CONVERT TO PROPER FORMAT
        CALL    CHANG   ;STORE AWAY THE ANSWRE
        DCR     C
        JNZ     CN1     ;DECREMENT GATE COUNT
        RET
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;CONV
        ;CALLS           ;NOTHING
        ;INPUTS          ;NOTHING
        ;OUTPUTS         ;NOTHING
        ;DESCRIPTION     ;CONV TAKES A DIGITAL LOGIC
        ;                ;CONSTANT AND CONVERTS IT TO ONE
        ;                ;WHICH DLS CAN OPERATE ON.FIRST
        ;                ;THE ADDRESS FROM THE UPDATE
        ;                ;SEQUENCE TABLE IS GOTTEN. THIS
        ;                ;POINTS TO TABLE T1 WHICH HAS
        ;                ;THE LOGIC VALUE FOR THE GIVEN
        ;                ;ADDRESS.
        ;                ;LOGIC '1' --> 01H
        ;                ;LOGIC '0' --> 00H
        ;                ;LOGIC 'X' --> 10H
;*****************************************************************
;
;
;
```

```
;
;
CONV:   MOV     E,M     ;THE UPDATE SEQUENCE TABLE
        INX     H       ;ADDRESS, POINTING TO T1
        MOV     D,M
        INX     H
        LDAX    D       ;DATA FROM T1
        ANI     0FH     ;STRIP OFF THE 4 MSBS'
        RZ
        CPI     1
        JNZ     DA1     ;CONVERT TO NEW FORMAT
        ORI     2
        RET             ;RETURN WITH NEW FORMATED
DA1:    MVI     A,2     ;DATA IN ACC
        RET
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;RCONV
        ;CALLS           ;NOTHING
        ;INPUTS          ;NOTHING
        ;OUTPUTS         ;NOTHING
        ;DESCRIPTION     ;RCONV ROUTINE TAKES DATA WHICH
        ;                ;THE PROGRAM HAS OPERATED UPON
        ;                ;AND CONVERTS IT BACK TO THE
        ;                ;PROPER FORMAT TO BE STORE AWAY
;*****************************************************************
;
;
;
;
;
RCONV:  CPI     0       ;IF IT MATCHES THEN CONVERT
        JNZ     $+6
        MVI     A,'0'   ;IT TO LOGIC '0'
        RET
        CPI     3
        JNZ     $+6     ;DEFALT CONVERT IT TO LOGIC '1'
        MVI     A,'1'
        RET
        MVI     A,'X'   ;CONVERT IT TO LOGIC 'X'
        RET
;
;
;
;
;
;*****************************************************************
```

```
          ;FUNCTION          :CHANG
          ;CALLS             :CONV,RCONV
          ;INPUTS            :XPAS
          ;OUTPUTS           :NOTHING
          ;DESCRIPTION       :THE CHANG ROUTINE INTRODUCES
          ;                  :THE PROPAGATION DELAY INTO
          ;                  :THE SIMULATED NETWORK.THIS IS
          ;                  :DONE BY TESTING AN OUTPUT TO
          ;                  :MAKE SURE IT HAS BEEN IN THE
          ;                  :TRANSITION STATE FOR ONE
          ;                  :UPDATE CYCLE
;*****************************************************************
;
;
;
;
;
CHANG:    MOV       B,A
          CALL      CONV       ;THE DELAY PROCESS IS DONE BY
          CALL      RCONV      ;TESTING TO SEE IF ANY CHANGE
          CMP       B          ;OCCURED BETWEEN THE N & N+1 STATE
          RZ
          CPI       'X'
          DCX       H
          DCX       H
          JNZ       DB2        ;GET OUTPUT READY TO CHANGE
          MOV       A,B
DB1:      MOV       E,M        ;GET LOCATION IN T2 TABLE
          INX       H
          MOV       D,M
          INX       H
          STAX      D          ;STORE NEW OUTPUT IN T2
          RET
DB2:      LDA       XPAS       ;TEST TO SEE IF USER
          CPI       'Y'        ;HAD ISSUED THE X-PASS COMMAND
          JNZ       DB1-1
          MVI       A,'X'      ;X-PASS
          JMP       DB1
;
;
;
;
;
;*****************************************************************
          ;FUNCTION          :TRACE
          ;CALLS             :OUTCH,GETCH,CRLF
          ;INPUTS            :NOTHING
          ;OUTPUTS           :TRON,XPAS
          ;DESCRIPTION       :TRACE FINDS OUT FROM THE USER IF
          ;                  :THEY WANT TO OPERATE IN THE TRACE
          ;                  :MODE AND IF THEY WANT TO OPERATE
```

```
        ;                       ;WITH THE X-PASS MODE.
;***************************************************************
;
;
;
;
;
TRACE:  LXI     H,DC2
DC1:    MOV     A,M             ;PRINT OUT THE FIRST MESSAGE
        CPI     '?'
        JZ      DC3
        INX     H
        CALL    OUTCH           ;DOES USER WANT TRACE MODE
        JMP     DC1
DC2:    DB      0AH,0DH,'TRACE=?'
DC3:    CALL    GETCH           ;GET RESPONCE TO THE QUESTION
        STA     TRON            ;STORE THE RESPONCE
        CALL    CRLF
        LXI     H,DC5
DC4:    MOV     A,M
        CPI     '?'
        JZ      DC6
        INX     H
        CALL    OUTCH           ;PRINT SECOND MESSAGE
        JMP     DC4             ;DOES USER WANT X-PASS
DC5:    DB      'X-PASS=?'
DC6:    CALL    GETCH           ;GET ANSWER TO QUESTION
        STA     XPAS            ;SAVE ANSWER
        CALL    CRLF
        CALL    CRLF
        CALL    CRLF
        CALL    CRLF
        CALL    CRLF
        RET
;
;
;
;
;***************************************************************
        ;FUNCTION        :TRAC
        ;CALLS           :OUTPT
        ;INPUTS          :WORK
        ;OUTPUTS         :WORK
        ;DESCRIPTION     :TRAC IS THE ROUTINE WHICH
        ;                :WHEN THE USER REQUESTS A
        ;                :TRACE, A PRINTOUT OF EACH
        ;                :UPDATE CYCLE IS MADE.
;***************************************************************
;
;
```

```
;
;
;
TRAC:   CALL    OUTPT   ;PRINT MONITORED LOGIC POINTS
        LDA     WORK+2
        ADI     99H     ;DECREMENT BCD COUNTER
        DAA
        STA     WORK+2  ;WHICH IS IN THIS LOCATION
        RET
;
;
;
;
;*****************************************************************
        ;FUNCTION       :TIME
        ;CALLS          :OUTCH
        ;INPUTS         :BEGIN
        ;OUTPUTS        :NOTHING
        ;DESCRIPTION    :TIME IS THE ERROR ROUTINE
        ;               :WHICH IS CALLED WHEN THE
        ;               :SIMULATED NETWORK HAS NOT
        ;               :REACHED A STABLE STATE IN
        ;               :THE ALLOTED TIME.
;*****************************************************************
;
;
;
;
;
TIME:   LXI     H,EB1   ;PRINT ERROR MESSAGE
        MOV     A,M     ;STOP THE SIMULATION DEAD
        CPI     '?'     ;AND INFORM USER OF TIME PROBLEM
        JZ      EB2
        CALL    OUTCH
        INX     H
        JMP     TIME+3
EB1:    DB      'THE CIRCUIT HAS NOT REACHED A STABLE STATE?'
EB2:    LXI     SP,BEGIN        ;RESTART SIMULATOR
        JMP     AA3
;
;
;
;
;*****************************************************************
        ;FUNCTION       :OSSL
        ;CALLS          :OUTCH
        ;INPUTS         :NOTHING
        ;OUTPUTS        :NOTHING
        ;DESCRIPTION    :OSSL IS THE ROUTINE WHICH
```

```
          ;                    ;INFORMS THE USER THAT THE
          ;                    ;SIMULATED NETWORK IS IN A
          ;                    ;OSCILATION STATE.AFTER THE
          ;                    ;MESSAGE IS PRINTED THE ROUTINE
          ;                    ;WILL INCERT LOGIC 'X' IN ALL
          ;                    ;T1 AND T2 POSITIONS WHICH
          ;                    ;ARE CAUSING THE PROBLEM.
;*************************************************************
;
;
;
;
;
OSSL:     LXI       H,EC1    ;PRINT ERROR MESSAGE
          MOV       A,M
          CPI       '?'
          RZ                 ;RETURN TO NORMAL OPERATION
          CALL      OUTCH
          INX       H
          JMP       OSSL+3
EC1:      DB        'THE CIRCUIT IS OSSILATING?'
```

## 4.9 General Purpose Routines and Memory Allocation

```
;*****************************************************************
        ;FUNCTION       ;GETCH & OUTCH
        ;CALLS          ;CI,CO
        ;INPUTS         ;NOTHING
        ;OUTPUTS        ;NOTHING
        ;DESCRIPTION    ;GETCH IS THE GET CHARACTER
        ;               ;FROM THE TERMINAL ROUTINE.
        ;               ;OUTCH IS THE ROUTINE WHICH
        ;               ;PRINTS THE CHARACTER THAT IS
        ;               ;IN THE ACC.THIS ROUTINE IS THE
        ;               ;ONLY ONE WHICH MUST BE CHANGED
        ;               ;FOR CUSTOMIZING THE I/O.
;*****************************************************************
;
;
;
;
;
GETCH:  CALL    CI      ;CPM GET CHARACTER ROUTINE
;
OUTCH:  PUSH    B
        MOV     C,A
        CALL    CO      ;CPM PRINT CHARACTER ROUTINE
        POP     B
        RET             ;ALL DONE
;
;
;
;
;
;*****************************************************************
        ;FUNCTION       ;CRLF
        ;CALLS          ;OUTCH
        ;INPUTS         ;NOTHING
        ;OUTPUTS        ;NOTHING
        ;DESCRIPTION    ;CRLF ROUTINE PRINTS A
        ;               ;CARRAGE RETURN FOLLOWED
        ;               ;BY A LINE FEED
;*****************************************************************
;
;
;
;
;
CRLF:   MVI     A,0DH   ;CARRAGE RETURN
        CALL    OUTCH   ;PRINT IT
        MVI     A,0AH   ;LINE FEED
        CALL    OUTCH   ;PRINT IT
        RET
;
;
```

```
;
;
;
;*****************************************************************
          ;FUNCTION          ;FNDP
          ;CALLS             ;NOTHING
          ;INPUTS            ;NEXT
          ;OUTPUTS           ;CARRY FLAG
          ;DESCRIPTION       ;FNDP ROUTINE SCANS THROUGH
          ;                  ;THE SOURCE PROGRAM TO FIND
          ;                  ;THE ',' CHARACTER
;*****************************************************************
;
;
;
;
;
FNDP:     PUSH     D
          XCHG
          LHLD     NEXT      ;GET ADDRESS OF LAST
          XCHG               ;SOURCE PROGRAM BYTE
ZA1:      MOV      A,L       ;TEST TO SEE IF WE ARE
          CMP      E         ;AT THE LAST BYTE
          JNZ      ZA2
          MOV      A,H
          CMP      D         ;IF NO ',' FOUND THEN
          JNZ      ZA2       ;CLEAR THE CARRY FLAG
          STC
          CMC
          POP      D
          RET
ZA2:      MOV      A,M       ;GET A SOURCE BYTE
          INX      H
          CPI      ','       ;COMPARE IT TO THIS
          JNZ      ZA1       ;NOT FOUND KEEP LOOKING
          STC                ;FOUND SET CARRY
          POP      D
          RET
;
;
;
;
;*****************************************************************
          ;FUNCTION          ;FNDS
          ;CALLS             ;NOTHING
          ;OUTPUTS           ;NOUTING
          ;OUTPUTS           ;NOTHING
          ;DESCRIPTION       ;FNDS IS THE ROUTINE WHICH
          ;                  ;FINDS A '/' IN THE SOURCE
          ;                  ;PROGRAM
```

```
;*****************************************************************
;
;
;
;
;
FNDS:   MOV     A,M     ;GET A CHARACTER
        INX     H
        CPI     ','     ;SEE IF WE HAVE OVER SHOT THE MARK
        RZ
        CPI     '/'     ;IS IT A MATCH
        RZ
        JMP     FNDS    ;NO TRY AGAIN
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;FNDCH
        ;CALLS           ;NOTHING
        ;INPUTS          ;NOTHING
        ;OUTPUTS         ;NOTHING
        ;DESCRIPTION     ;FNDCH IS THE ROUTINE WHICH
        ;                ;FINDS A CHARACTER WHICH LIES
        ;                ;BETWEEN 'A' AND 'Z'.
;*****************************************************************
;
;
;
;
;
FNDCH:  MOV     A,M     ;GET A CHARACTER
        CPI     'A'     ;SEE IF IT IS BETWEEN 'A'
        JP      ZB1
        INX     H
        JMP     FNDCH
ZB1:    CPI     'Z'+1   ;AND 'Z'
        RM
        INX     H
        JMP     FNDCH
;
;
;
;
;*****************************************************************
        ;FUNCTION        ;PUBYT
        ;CALLS           ;OUTCH
        ;INPUTS          ;NOTHING
```

```
        ;DESCRIPTION    ;THE PRBYT ROUTINE TAKES
        ;               ;THE CONTENTS OF ACC AND
        ;               ;PRINTS IT AS TWO HEX
        ;               ;DIGITS
;*******************************************************
;
;
;
;
;
PRBYT:  PUSH    PSW
        RAR             ;MOVE THE FOUR MSB'S TO
        RAR             ;THE LSB POSITIONS
        RAR
        RAR
        ANI     0FH     ;STRIP OFF THE TOP
        ORI     30H     ;CONVERT TO PROPER FORMAT
        CPI     3AH
        JC      $+5
        ADI     7
        CALL    OUTCH   ;PRINT FIRST DIGIT
        POP     PSW     ;BRING BYTE BACK
        ANI     0FH     ;TAKE THE TOP OFF
        ORI     30H     ;CONVERT IT
        CPI     3AH
        JC      $+5
        ADI     7
        CALL    OUTCH   ;PRINT IT
        RET
;
;
;
;
;
;*******************************************************
        ;FUNCTION       ;GETDM
        ;CALLS          ;GETCH
        ;INPUTS         ;NOTHING
        ;OUTPUTS        ;NOTHING
        ;DESCRIPTION    ;GETDM ROUTINE GETS A
        ;               ;DECMIAL NUMBER (0-255)
        ;               ;FROM THE USER.ALL NUMBERS
        ;               ;COME IN ASCII MUST BE
        ;               ;CONVERTED TO HEX.
;*******************************************************
;
;
;
;
;
GETDM:  CALL    GETCH   ;GET A CHARACTER
```

```
          CPI       '0'         ;IT MUST BE BETWEEN
          RM                    ;0 AND 9
          CPI       '9'+1
          RP
          PUSH      PSW         ;SAVE IT
          MOV       A,B         ;B<-- CURRENT COUNT
          ORA       A           ;CLEAR CARRY
          RAL                   ;DOUBLE IT. 10=(2*2*2*N+2*N)+(N+1)
          MOV       B,A
          ORA       A
          RAL
          ORA       A
          RAL
          ADD       B           ;FINAL RESULTS
          MOV       B,A
          POP       PSW         ;GET NUMBER
          ANI       0FH
          ADD       B           ;ADD THE NEXT DIGIT
          MOV       B,A
          JMP       GETDM
;
;
;
;
;*****************************************************************
          ;MEMORY ALLOCATION
          ;
          ;
;*****************************************************************
;
;
;
;
;
START:    DS        2   ;BEGINNING OF SOURCE PROGRAM
NEXT:     DS        2   ;END OF SOURCE PROGRAM
WORK:     DS        4   ;TEMPORY WORK REGISTERS
LENTH:    DS        1   ;LENGTH OF A SOURCE LINE
DELAY:    DS        1   ;CYCLE DELAY COUNT
TEST:     DS        1   ;NUMBER OF UPDATE COUNTS
COUNT:    DS        1   ;TOTAL NUMBER OF SYMBOLS
PLACE:    DS        1   ;TEMPORY UPDATE COUNT
ERROR:    DS        1   ;TYPE OF ERROR
NUMB:     DS        2   ;NUMBER OF LOGIC GATES
TRON:     DS        1   ;TRACE CONTROL CHARACTER
XPAS      DS        1   ;X-PASS CONTROL CHARACTER
SYMBS:    DS        2   ;START OF SYMBOL TABLE
SYMBE:    DS        2   ;END OF SYMBOL TABLE
T1S:      DS        2   ;START OF T1 TABLE
T1E:      DS        2   ;END OF T1 TABLE
```

```
T2S:     DS      2   ;START OF T2 TABLE
T2E:     DS      2   ;END OF T2 TABLE
SIMTS:   DS      2   ;START OF UPDATE SEQUENCE TABLE
SIMTE:   DS      2   ;END OF UPDATE SEQUENCE TABLE
INP:     DS      2   ;START OF INP TABLE
OUTP:    DS      2   ;START OF OUTP TABLE
INST:    DS      2   ;START OF TEST DATA STRING
NA2:     DS      1   ;# OF 2 INPUT NAND GATES
NA4:     DS      1   ;# OF 4 INPUT NAND GATES
AN2:     DS      1   ;# OF 2 INPUT AND GATES
AN4      DS      1   ;# OF 4 INPUT AND GATES
OR2:     DS      1   ;# OF 2 INPUT OR GATES
OR4:     DS      1   ;# OF 4 INPUT OR GATES
NO2:     DS      1   ;# OF 2 INPUT NOR GATES
NO4:     DS      1   ;# OF 4 INPUT NOR GATES
EX2:     DS      1   ;# OF 2 INPUT EXOR GATES
EX4:     DS      1   ;# OF 4 INPUT EXOR GATES
BUFFR:   DS      64  ;INPUT DATA BUFFER
DATA:    DS      1   ;START OF SOURCE PROGRAM
         END
```

# CHAPTER 5

# USING THE DIGITAL LOGIC SIMULATOR

## 5.1 Design Examples

In order for the user to get a better grasp of how DLS operates a few design examples are given. On the computer printouts that which is underlined is what the user has typed. The comments along the right side were added later to emphasize certain points.

The first design example has its printout previously shown in Figure 2-2. Now what will be done is to show how all that came about. Figure 5-1 is the circuit to be simulated.

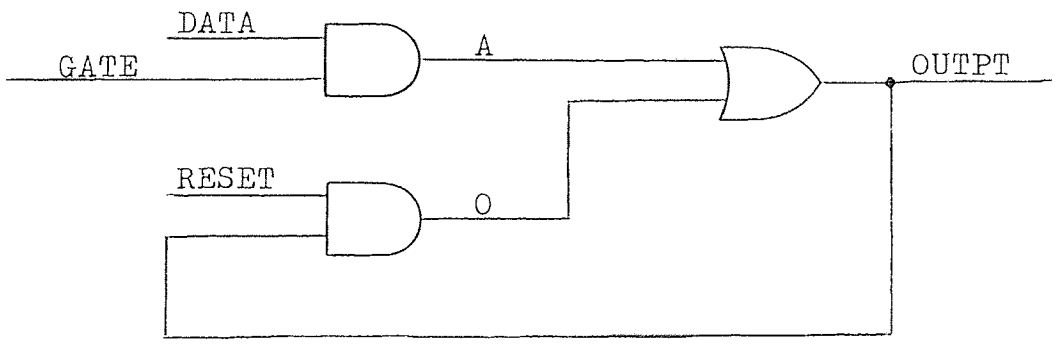



Figure 5-1

Once the simulator starts to run the title will be printed followed by a question. The simulator wants to know if it should clear the tables in the memory. This is for protecting against destroying old files in memory.

```
DIGITAL LOGIC SIMULATOR

CLEAR MEMORY ? YES
:1ØØØ .INPUT.  DATA,GATE,RESET
:2ØØØ .AND/2.  DATA,GATE,A
:3ØØØ .AND/2.  RESET,OUTPT,O
:4ØØØ .OR/2.   A,O,OUTPT
:5ØØØ .PRINT.  DATA,GATE,RESET,A,OUTPT
:6ØØØ .END.
```

The user types in topigraphical discription of the network.

```
:COMP

.INPUT.  DATA,GATE,RESET
.AND/2.  DATA,GATE,A
.AND/2.  RESET,OUTPT,O
.OR/2.   A,O,OUTPT
.PRINT.  DATA,GATE,RESET,A,OUTPT
.END.

AND/2 =Ø2
OR/2  =Ø1
```

Once the discription is done the compile command is issued.

The compiler will print the discription along with the logic gate count.

```
:FANOUT

DATA  :Ø1
GATE  :Ø1
RESET :Ø1
A     :Ø1
```

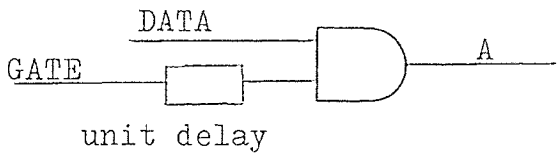
The user requests fanout analysis.

```
OUTPT:Ø1
O     :Ø1

:EXEC

# OF TIME UNITS PER PULSE = 1Ø
# OF TEST INPUTS = 7
TRACE = NO
X-PASS = YES

DATA  :XX1111Ø
GATE  :XØ1ØØØ1
RESET :XO11Ø11

      DATA  GATE  RESET  A  OUTPT
-------------------------------
ØØ:   X     X     X     X  X
Ø1:   X     Ø     Ø     Ø  Ø
Ø2:   1     1     1     1  1
Ø3:   1     Ø     1     Ø  1
Ø4:   1     Ø     Ø     Ø  Ø
Ø5:   1     Ø     1     Ø  Ø
Ø6:   Ø     1     1     Ø  X   THE CIRCUIT IS OSCILLATING
```

The execution command is given. The executer will request some simulation parameters.

The input test patterns.

DLS simulation printout. There exists a possible hazard when DATA and GATE change at the same time.




By putting a delay in the GATE line the two logic levels no longer change simultaniously.

Figure 5-2

```
:2ØØØ .AND/2.  DELAY,DATA,A
:15ØØ .AND/2.  GATE,GATE,DELAY

:COMP

.INPUT. DATA,GATE,RESET
.AND/2. GATE,GATE,DELAY
.AND/2. DELAY,DATA,A
.AND/2. RESET,OUTPT,O
.OR/2.  A,O,OUTPT
.PRINT. DATA,GATE,RESET,A,OUTPT
.END.

AND/2  =Ø3
OR/2   =Ø1

:FANOUT

DATA   :Ø1
GATE   :Ø2
RESET  :Ø1
DELAY  :Ø1
A      :Ø1
OUTPT  :Ø1
O      :Ø1

:EXEC

# OF TIME UNITS PER PULSE = 1Ø
# OF TEST INPUTS = 7
TRACE = NO
X-PASS = YES
```

The delay is simply an AND gate with both inputs tied together.

The user simply modifies one line and adds another then recompiles the network.

```
DATA :XX1111Ø
GATE :XØ1ØØØ1
RESET:XØ11Ø11
```

Run it through the same test pattern.

```
      D  G  R  E        O  U  T  P  A  T
      A  A  E  S  E     U  T  P  T  A  E
      T  T  S  E  T  A  T  P  T  A  E  T
-------------------------------------------
ØØ:   X  X  X  X  X     X  X  X  X  X  X
Ø1:   X  Ø  Ø  Ø  Ø     X  Ø  Ø  Ø  Ø  Ø
Ø2:   1  1  1  1  1     1  1  1  1  1  1
Ø3:   1  Ø  1  Ø  1     1  Ø  1  Ø  1  Ø
Ø4:   1  Ø  Ø  Ø  Ø     1  Ø  Ø  Ø  Ø  Ø
Ø5:   1  Ø  1  Ø  Ø     1  Ø  1  Ø  Ø  Ø
Ø6:   Ø  1  1  Ø  Ø     Ø  1  1  Ø  Ø  1
```

No hazard exists.

The second example will show how DLS detects race conditions. Figure 5-3 is simply a string of OR gates.

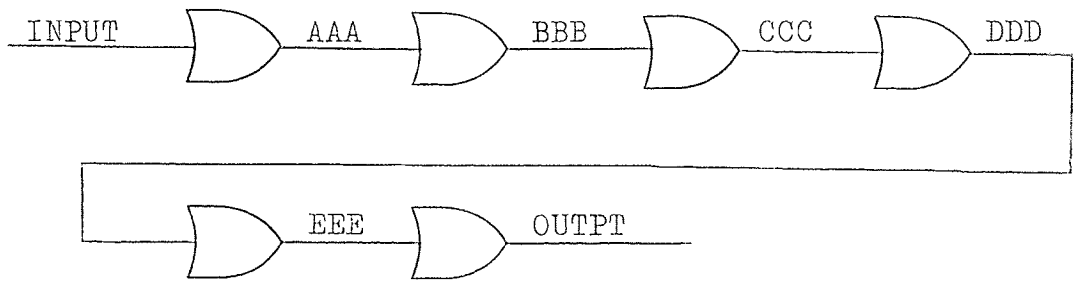



Figure 5-3

By using the DLS trace mode signal propagation can be viewed.

```
:NEW                                          Clear out the
CLEAR MEMORY ?YES                             memory and describe
:1ØØØ .INPUT.  INPUT                          the new network.
:2ØØØ .OR/2.  INPUT,INPUT,AAA
:2ØØ1 .OR/2.  AAA,AAA,BBB
:2ØØ2 .OR/2.  BBB,BBB,CCC
:2ØØ3 .OR/2.  CCC,CCC,DDD
:2ØØ4 .OR/2.  DDD,DDD,EEE
:2ØØ5 .OR/2.  EEE,EEE,OUTPT
:3ØØØ .PRINT. INPUT,AAA,BBB,CCC,DDD,EEE,OUTPT
:4ØØØ .END.
:COMP

.INPUT. INPUT
.OR/2.  INPUT,INPUT,AAA
.OR/2.  AAA,AAA,BBB
.OR/2.  BBB,BBB,CCC
.OR/2.  CCC,CCC,DDD
.OR/2.  DDD,DDD,EEE
.OR/2.  EEE,EEE,OUTPT
.PRINT. INPUT,AAA,BBB,CCC,DDD,EEE,OUTPT
.END.

OR/2  = Ø6

:FANOUT

INPUT:Ø2
AAA  :Ø2
BBB  :Ø2
```

```
CCC   :Ø2
DDD   :Ø2
EEE   :Ø2
OUTPT:ØØ

:EXEC
# OF TIME UNITS PER PULSE = 1Ø
# OF TEST INPUTS = 2
TRACE = YES
X-PASS = YES

INPUT:Ø1
```

Execute the simulator with the trace mode on.

| | INPUT | AA | BB | CC | DD | EE | OUTPT | |
|---|---|---|---|---|---|---|---|---|
| ØØ: | Ø | X | X | X | X | X | X | |
| ØØ: | Ø | Ø | X | X | X | X | X | |
| ØØ: | Ø | Ø | Ø | X | X | X | X | |
| ØØ: | Ø | Ø | Ø | Ø | X | X | X | |
| ØØ: | Ø | Ø | Ø | Ø | Ø | X | X | |
| ØØ: | Ø | Ø | Ø | Ø | Ø | Ø | X | |
| ØØ: | Ø | Ø | Ø | Ø | Ø | Ø | Ø | |
| ØØ: | Ø | Ø | Ø | Ø | Ø | Ø | Ø | ←-stable state |
| Ø1: | 1 | Ø | Ø | Ø | Ø | Ø | Ø | |
| Ø1: | 1 | X | Ø | Ø | Ø | Ø | Ø | |
| Ø1: | 1 | 1 | X | Ø | Ø | Ø | Ø | |
| Ø1: | 1 | 1 | 1 | X | Ø | Ø | Ø | |
| Ø1: | 1 | 1 | 1 | 1 | X | Ø | Ø | |
| Ø1: | 1 | 1 | 1 | 1 | 1 | X | Ø | |
| Ø1: | 1 | 1 | 1 | 1 | 1 | 1 | X | |
| Ø1: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| Ø1: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | ←-stable state |

Re-execute the network but this time set the clock up so that there will only be five update cycles per time unit.

```
:EXEC

# OF TIME UNITS PER PULSE = 5
# OF TEST INPUTS = 2
TRACE = YES
X-PASS = YES

INPUT:Ø1

      I                       O
      N                       U
      P   A   B   C   D   E   T
      U   A   B   C   D   E   P
      T   A   B   C   D   E   T
-----------------------------
ØØ:   Ø   1   1   1   1   1   1
ØØ:   Ø   X   1   1   1   1   1
ØØ:   Ø   Ø   X   1   1   1   1
ØØ:   Ø   Ø   Ø   X   1   1   1
ØØ:   Ø   Ø   Ø   Ø   X   1   1
ØØ:   Ø   Ø   Ø   Ø   Ø   X   1
THE CIRCUIT HAS NOT REACHED A STABLE STATE
```

Re-executer the network this time with only five update cycles per time unit.

Since the network was not recompiled all outputs start with their last value.

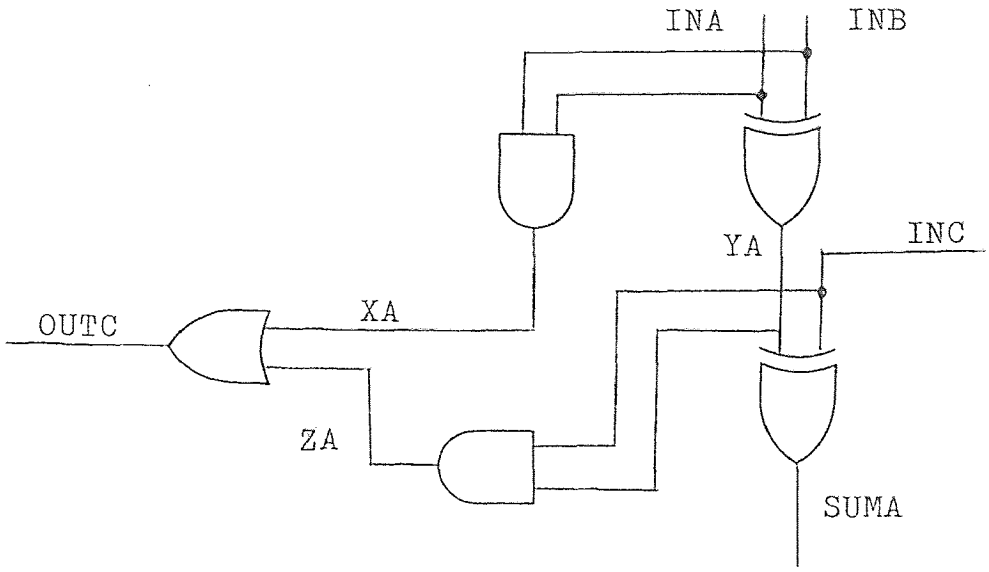
After five update cycles no stable state was reached.

The next example is the design of a two bit full-adder. First a one bit full-adder will be simulated then the modification to a two bit adder. Figure 5-4a is the basic full-adder and Figure 5-4b is how two such full-adder blocks are put together to form the circuit.

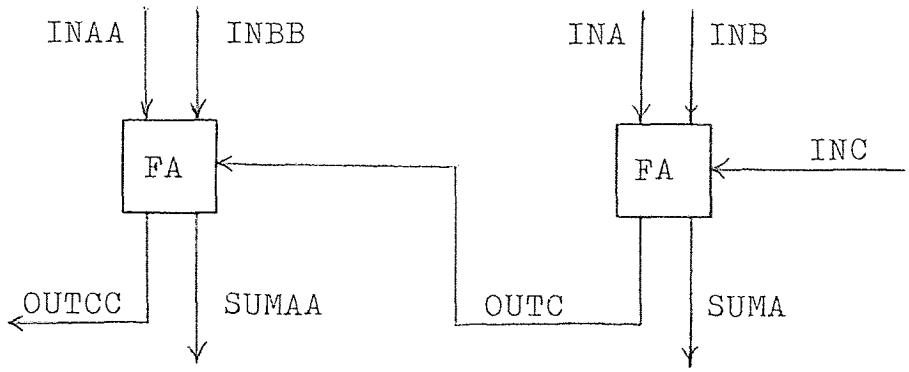
```
:NEW
CLEAR MEMORY ? YES
```

New network to be feed to DLS.

Full-Adder Circuit




(a)




(b)

Figure 5-4

```
:1ØØØ .INPUT.  INA,INB,INC
:2ØØØ .EXOR/2. INA,INB,YA
:2ØØ1 .EXOR/2. INC,YA,SUMA
:2ØØ2 .AND/2.  INC,YA,ZA
:2ØØ3 .AND/2.  INA,INB,XA
:2ØØ4 .OR/2.   XA,ZA,OUTC
:3ØØØ .PRINT.  INC,INB,INA,SUMA,OUTC
:4ØØØ .END.
```

Describe the basic full adder.

```
:COMP

.INPUT. INA,INB,INC
.EXOR/2. INA,INB,YA
.EXOR/2. INC,YA,SUMA
.AND/2.  INC,YA,ZA
.AND/2.  INA,INB,XA
.OR/2.   XA,ZA,OUTC
.PRINT.  INC,INB,INA,SUMA,OUTC
.END.

AND/2 =Ø2
OR/2  =Ø1
EXOR/2=Ø2

:FANOUT
INA  :Ø2
INB  :Ø2
INC  :Ø2
YA   :Ø2
SUMA :ØØ
ZA   :Ø1
XA   :Ø1
OUTC :ØØ
```

Test the first design stage.

```
:EXEC

# OF TIME UNITS PER PULSE =1Ø
# OF TEST INPUTS =8
TRACE =NO
X-PASS =YES

INA   :Ø1Ø1Ø1Ø1
INB   :ØØ11ØØ11
INC   :ØØØØ1111
```

There are $2^n$ posible test patterns where n equals the number of inputs.

| | INC | INB | INA | SUMA | OUTC |
|---|---|---|---|---|---|
| ØØ: | Ø | Ø | Ø | Ø | Ø |
| Ø1: | Ø | Ø | 1 | 1 | Ø |
| Ø2: | Ø | 1 | Ø | 1 | Ø |
| Ø3: | Ø | 1 | 1 | Ø | 1 |
| Ø4: | 1 | Ø | Ø | 1 | Ø |
| Ø5: | 1 | Ø | 1 | Ø | 1 |
| Ø6: | 1 | 1 | Ø | Ø | 1 |
| Ø7: | 1 | 1 | 1 | 1 | 1 |

Everything checks out.

```
:1ØØØ .INPUT.  INA,INAA,INB,INBB,INC
:3ØØØ .EXOR/2. INAA,INBB,YAA
:3ØØ1 .EXOR/2. OUTC,YAA,SUMAA
:3ØØ2 .AND/2.  OUTC,YAA,ZAA
:3ØØ3 .AND/2.  INAA,INBB,XAA
:3ØØ4 .OR/2.   XAA,ZAA,OUTCC
:4ØØØ .PRINT.  INC,INBB,INB,INAA,INA,OUTCC,SUMAA,SUMA
:5ØØØ .END.
```

Modify the description for the second stage.

```
:COMP

.INPUT.  INA,INAA,INB,INBB,INC
.EXOR/2. INA,INB,YA
.EXOR/2. INC,YA,SUMA
.AND/2.  INC,YA,ZA
.AND/2.  INA,INB,XA
.OR/2.   XA,ZA,OUTC
.EXOR/2. INAA,INBB,YAA
.EXOR/2. OUTC,YAA,SUMAA
.AND/2.  OUTC,YAA,ZAA
.AND/2.  INAA,INBB,XAA
.OR/2.   XAA,ZAA,OUTCC
.PRINT.  INC,INBB,INB,INAA,INA,OUTCC,SUMAA,SUMA
.END.

AND/2 =Ø4
OR/2  =Ø2
EXOR/2=Ø4

:EXEC

# OF TIME UNITS PER PULSE =2Ø
# OF TEST INPUTS =32
TRACE =NO
X-PASS =YES

INA   :Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1
INAA  :ØØ11ØØ11ØØ11ØØ11ØØ11ØØ11ØØ11ØØ11
INB   :ØØØØ1111ØØØØ1111ØØØØ1111ØØØØ1111
INBB  :ØØØØØØØØ11111111ØØØØØØØØ11111111
INC   :ØØØØØØØØØØØØØØØØ1111111111111111
```

32 possible test patterns.

| | INC | INBB | INB | INAA | INA | OUTCC | SUMAA | SUMA |
|---|---|---|---|---|---|---|---|---|
| ØØ: | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø |
| Ø1: | Ø | Ø | Ø | Ø | 1 | Ø | Ø | 1 |
| Ø2: | Ø | Ø | Ø | 1 | Ø | Ø | 1 | Ø |
| Ø3: | Ø | Ø | Ø | 1 | 1 | Ø | 1 | 1 |
| Ø4: | Ø | Ø | 1 | Ø | Ø | Ø | Ø | 1 |
| Ø5: | Ø | Ø | 1 | Ø | 1 | Ø | 1 | Ø |
| Ø6: | Ø | Ø | 1 | 1 | Ø | Ø | 1 | 1 |
| Ø7: | Ø | Ø | 1 | 1 | 1 | 1 | Ø | Ø |
| Ø8: | Ø | 1 | Ø | Ø | Ø | Ø | 1 | Ø |
| Ø9: | Ø | 1 | Ø | Ø | 1 | Ø | 1 | 1 |
| 1Ø: | Ø | 1 | Ø | 1 | Ø | 1 | Ø | Ø |
| 11: | Ø | 1 | Ø | 1 | 1 | 1 | Ø | 1 |
| 12: | Ø | 1 | 1 | Ø | Ø | Ø | 1 | 1 |
| 13: | Ø | 1 | 1 | Ø | 1 | 1 | Ø | Ø |
| 14: | Ø | 1 | 1 | 1 | Ø | 1 | Ø | 1 |
| 15: | Ø | 1 | 1 | 1 | 1 | 1 | 1 | Ø |
| 16: | 1 | Ø | Ø | Ø | Ø | Ø | Ø | 1 |
| 17: | 1 | Ø | Ø | Ø | 1 | Ø | 1 | Ø |
| 18: | 1 | Ø | Ø | 1 | Ø | Ø | 1 | 1 |
| 19: | 1 | Ø | Ø | 1 | 1 | 1 | Ø | Ø |
| 2Ø: | 1 | Ø | 1 | Ø | Ø | Ø | 1 | Ø |
| 21: | 1 | Ø | 1 | Ø | 1 | Ø | 1 | 1 |
| 22: | 1 | Ø | 1 | 1 | Ø | 1 | Ø | Ø |
| 23: | 1 | Ø | 1 | 1 | 1 | 1 | Ø | 1 |
| 24: | 1 | 1 | Ø | Ø | Ø | Ø | 1 | 1 |
| 25: | 1 | 1 | Ø | Ø | 1 | 1 | Ø | Ø |
| 26: | 1 | 1 | Ø | 1 | Ø | 1 | Ø | 1 |
| 27: | 1 | 1 | Ø | 1 | 1 | 1 | 1 | Ø |
| 28: | 1 | 1 | 1 | Ø | Ø | 1 | Ø | Ø |
| 29: | 1 | 1 | 1 | Ø | 1 | 1 | Ø | 1 |
| 3Ø: | 1 | 1 | 1 | 1 | Ø | 1 | 1 | Ø |
| 31: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

The modified circuit works fine.

The next example shows how the use of the initial condition aids in the circuit analysis. Figure 5-5 is anasynchronous finite state machine to be simulated.

Asynchronous Finite State Machine

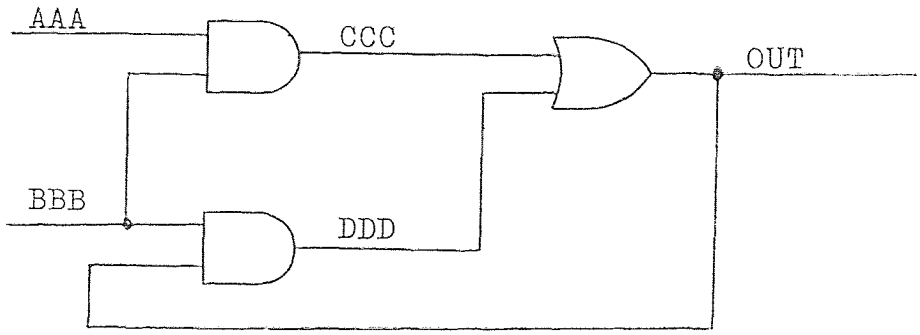



Figure 5-5

New circuit for DLS to simulate.

```
CLEAR MEMORY ?YES
:1ØØØ .INPUT.  AAA,BBB
:2ØØØ .AND/2.  AAA,BBB,CCC
:3ØØØ .AND/2.  BBB,OUT,DDD
:4ØØØ .OR/2.   CCC,DDD,OUT
:5ØØØ .PRINT.  AAA,BBB,CCC,DDD,OUT
:6ØØØ .END.
:COMP

.INPUT.  AAA,BBB
.AND/2.  AAA,BBB,CCC
.AND/2.  BBB,OUT,DDD
.OR/2.   CCC,DDD,OUT
.PRINT.  AAA,BBB,CCC,DDD,OUT
.END.
```

```
AND/2 =Ø2
OR/2  =Ø1

:EXEC

# OF TIME UNITS PER PULSE =1Ø
# OF TEST INPUTS =4
TRACE =NO
X-PASS =YES

AAA  :Ø1ØØ
BBB  :111Ø


     A  B  C  D  O
     A  B  C  D  U
     A  B  C  D  T
--------------------
ØØ:  Ø  1  Ø  X  X
Ø1:  1  1  1  1  1
Ø2:  Ø  1  Ø  1  1
Ø3:  Ø  Ø  Ø  Ø  Ø

:4ØØØ .OR/2.   CCC,DDD,OUT   IC=Ø
:COMP

.INPUT.  AAA,BBB
.AND/2.  AAA,BBB,CCC
.AND/2.  BBB,OUT,DDD
.OR/2.   CCC,DDD,OUT    IC=Ø
.PRINT.  AAA,BBB,CCC,DDD,OUT
.END.

AND/2  =Ø2
OR/2   =Ø1
```

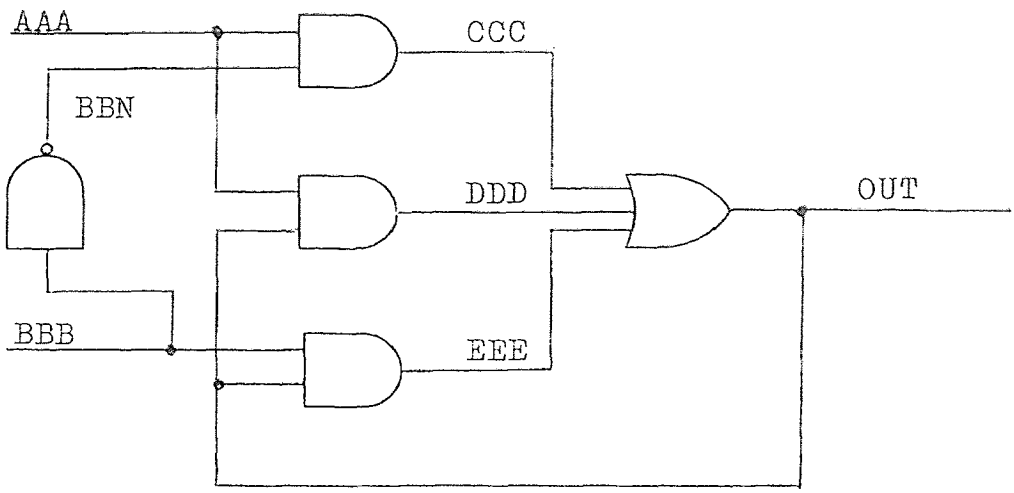
OUT starts in the unknown state.

See what happens with OUT having a initial value.

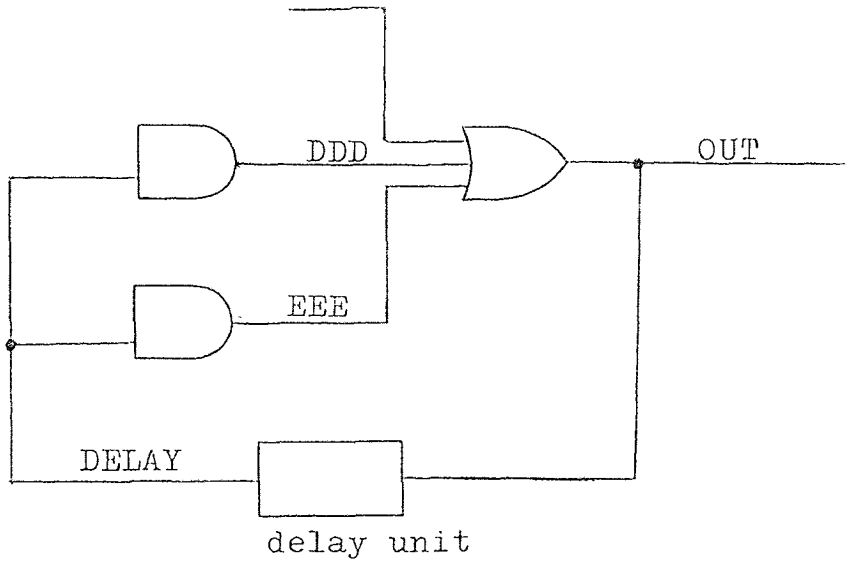
```
:EXEC

# OF TIME UNITS PER PULSE =1Ø
# OF TEST INPUTS =4
TRACE =NO
X-PASS =YES

AAA  :Ø1ØØ                          Run through the
BBB  :111Ø                          same test pattern.

     A  B  C  D  O
     A  B  C  D  U
     A  B  C  D  T
--------------------
ØØ:  Ø  1  Ø  Ø  Ø                  This time all
Ø1:  1  1  1  1  1                  is well.
Ø2:  Ø  1  Ø  1  1
Ø3:  Ø  Ø  Ø  Ø  Ø
```

The final example is another asynchronous finite state machine, this time with two possible hazards. The first problem is the need for a initial condition on the output and the second problem is that there exists a race condition in the feedback path of the circuit. Figure 5-6a is the basic circuit which has the two possible hazard conditions in it. Figure 5-6b is the modified circuit which has introduced into the feedback path a delay which should eliminate one of the hazards.

Circuit with Race Condition




(a)




(b)

Figure 5-6

```
:NEW
CLEAR MEMORY ?YES
:1ØØØ .INPUT.  AAA,BBB
:2ØØØ .NAND/2. BBB,BBB,BBN
:3ØØØ .AND/2.  AAA,BBN,CCC
:4ØØØ .AND/2.  AAA,OUT,DDD
:5ØØØ .AND/2.  BBB,OUT,DDD
:6ØØØ .OR/4.   Ø,CCC,DDD,EEE,OUT
:7ØØØ .PRINT.  AAA,BBB,OUT
:8ØØØ .END.
```

New circuit to be simulated.

```
:RESEQ
```

Issue the resequence command. Then print the program.

```
:LIST

ØØØØ .INPUT.  AAA,BBB
ØØ1Ø .NAND/2. BBB,BBB,BBN
ØØ2Ø .AND/2.  AAA,BBN,CCC
ØØ3Ø .AND/2.  AAA,OUT,DDD
ØØ4Ø .AND/2.  BBB,OUT,EEE
ØØ5Ø .OR/2.   Ø,CCC,DDD,EEE,OUT
ØØ6Ø .PRINT.  AAA,BBB,OUT
ØØ7Ø .END.
```

```
:COMP
```

Compile the network.

```
.INPUT.  AAA,BBB
.NAND/2. BBB,BBB,BBN
.AND/2.  AAA,BBN,CCC
.AND/2.  AAA,OUT,DDD
.AND/2.  BBB,OUT,EEE
.OR/2.   Ø,CCC,DDD,EEE,OUT
.PRINT.  AAA,BBB,OUT
.END.
```

```
NAND/2 =Ø1
AND/2  =Ø3
OR/4   =Ø1

:EXEC

# OF TIME UNITS PER PULSE =1Ø
# OF TEST INPUTS =4
TRACE =NO
X-PASS =YES


     A  B  O
     A  B  U
     A  B  T
--------------------
ØØ:  1  1  X
Ø1:  1  Ø  1
Ø2:  Ø  1  1  THE CIRCUIT IS OSCILLATING
Ø3:  Ø  Ø  Ø

:ØØ3Ø .AND/2.  AAA,DELAY,DDD
:ØØ4Ø .AND/2.  BBB,DELAY,EEE
:ØØ45 .AND/2.  OUT,OUT,DELAY
:ØØ5Ø .OR/2.   Ø,CCC,DDD,EEE    IC=Ø

:COMP

.INPUT.  AAA,BBB
.NAND/2. BBB,BBB,BBN
.AND/2.  AAA,BBN,CCC
.AND/2.  AAA,DELAY,DDD
.AND/2.  BBB,BELAY,EEE
.AND/2.  OUT,OUT,DELAY
.OR/2.   Ø,CCC,DDD,EEE,OUT    IC=Ø
.PRINT.  AAA,BBB,OUT
.END.
```

Execute the program and find the hazards.

There are two problems to be corrected.

Recompile the corrected network.

```
NAND/2 =Ø1
AND/2  =Ø4
OR/2   =Ø1

:EXEC

# OF TIME UNITS PER PULSE =1Ø
# OF TEST INPUTS =8
TRACE =NO
X-PASS =YES

AAA  :ØØ1111ØØ
BBB  :Ø1Ø11Ø1Ø
```

| | A | B | O |
|---|---|---|---|
| | A | B | U |
| | A | B | T |
| ØØ: | Ø | Ø | Ø |
| Ø1: | Ø | 1 | Ø |
| Ø2: | 1 | Ø | 1 |
| Ø3: | 1 | 1 | 1 |
| Ø4: | 1 | 1 | 1 |
| Ø5: | 1 | Ø | 1 |
| Ø6: | Ø | 1 | 1 |
| Ø7: | Ø | Ø | Ø |

The two possible hazards have been eliminated.

# CHAPTER 6

# CONCLUSION

## 6.1 A Few Last Words.

With the use of DLS it is now possible for a digital circuit designer to debug most, if not all of his digital designs in a matter of minutes. The designer also has the satisfaction that the logic is correct and that he now can concentrate on hardware connection and failure errors.

The DLS program has proven beneficial to the logic designer in several cases, including the following.

1) The simulator saves money by correcting design errors before the hardware is fabricated.

2) The simulator saves time by permiting redesign prior to fabrication.

3) The computer listing serves as documentation of the actual design.

4) The simulator aids in debugging of the hardware by supplying accurate timing diagrams to which the waveforms monitored in the system can be compared.

5) By requiring the designer to describe his work in detail, the designer is made more aware of the design techniques and any redundancies he may be prone to use.

6) By providing accounting statistics of each type of element and loading of each element, the program aids the designer in making selections of assignments and card types for the building of the hardware.

7) The computer listings expedite the checking of the circuit after the hardware is built by limiting the number of causes of errors to be checked.

8) The computer outputs allow the designer to see many signals at one time, as opposed to a few at a time, as would be the case when limited by available traces on oscilloscopes.

9) Often the design will lend itself to the case where the number of inputs is small and all combinations and permutations of the inputs can be created by the computer and the design totally checked. Usually in a hardware setup only a limited number of inputs can be checked.

10) The timing diagrams when sampled at "gate" times will often show logic spikes in hard copy as opposed to the small time duration of a spike on a scope.

# BIBLIOGRAPHY


Breuer, Melvin A., Design Automation of Digital Systems. New Jersey: Prentice-Hall, Inc., 1972

Breuer, Melvin A., Digital System Design Automation. California: Computer Science Press, Inc., 1975

Breuer, Melvin A., "Recent Developments in Design Automation," Computer, May/June 1972, pp. 23-35

Chu, Yaohan, Computer Organization and Microprogramming. New Jersey: Prentice-Hall, Inc., 1972

Chu, Yaohan, Introduction to Computer Organization. New Jersey: Prentice-Hall, Inc., 1970

Flomenhoft, Mark J., and Csencsits, Brenda m., "A Minicomputer-Based Logic Circuit Fault Simulator," ACM Sigda Newsletter, Vol. 4, No. 3, 1974, pp. 15-19

Hartenstein, Reiner W,, Fundamentals of Structured Hardware Design. New York: North Holland Publishing Company, 1977

Jephson, J. S., McQuarrie, R. P., and Vogelsberg, R. E., "A Three-Value Computer Design Verification System," IBM System Journal, Vol. 8, No. 3, 1969, pp. 178-189

Kahn, Hilary J., and May, J. W. R., "The Using of Logic Simulation in the Design of a Large Computer System," The Radio and Electronic Engineer, Vol. 43, No. 8, 1978

Osborne, Adam, 8080 Programming For Logic Design. California: Adam Osborne and Associates, Inc., 1976

Ulrich, E.G., and Baker, T., "Concurrent Simulation of Nearly Identical Digital Networks," Computer, April 1974, pp.39-44

Yoeli, Michaël, and Rinon, Shlomo, "Application of Ternary Algebra to the Study of Static Hazards," Journal of the Association for Computing Machinery, Vol. 11, No. 1, 1964, pp. 84-97