AN EXECUTIVE PROGRAM AND HARDWARE DATA SOURCES FOR HIGH SPEED TRAFFIC SIMULATION USING A MINICOMPUTER

Colin Godfrey Yamey

A Dissertation submitted to the Faculty of Engineering,
University of the Witwatersrand, Johannesburg,
for the Degree of Master of Science in Engineering

Johannesburg 1976

ABSTRACT

Simulation has played a vital role in solving problems which are intractible or which are too complex to be solved mathematically. As road traffic is such a problem traffic simulators have become invaluable design tools for the traffic engineer.

Software and hardware traffic simulators have proliferated. Most software simulators have the disadvantages of low simulation speed and require large computers resulting in a high simulation cost. On the other hand hardware simulators have high speed potential but are inflexible and costly to develop and construct.

Advances in the field of mini computers have made a hybrid software-hardware simulator possible. Using this technique it is possible to design a simulator which has a relatively high speed, flexibility and moderate cost.

This dissertation describes a traffic simulator which comprises a NOVA 1200 mini computer, peripherals and two hardware random number generators which are used as data sources.

A large proportion of this dissertation is the design of a simulation executive. The executive provides facilities for defining the traffic network, entering run time data and provides the simulator user with interactive control over the simulation. The executive further provides a framework for simulation routines. The routines perform the interpretation of the network definition, data entered by the user and the actual modelling of traffic. Routines for simulating different traffic elements are easily added due to the modular design of the executive. In the present work routines have

been designed to simulate intersections with fixed time control, streets and entry streets. Additional routines are provided for printing at the queue lengths at intersections.

Two hardware data sources are described. The first, designed by Walker, is used for generating travelling times and gap acceptance times which may have arbitrary frequency distributions. The second is a generator which produces sixteen independent outputs with a Poissonian distribution. It is used for generating vehicles and assigning vehicles to lanes.

The simulator described can model a traffic network of up to forty intersections each having up to five lanes per approach. A computer with only 16K words of memory is used. The speed of simulation is approximately 200 times faster than real time per typical intersection.

DECLARATION

I hereby declare that this dissertation is my own work and that it has not previously been submitted for a degree at any University.

C.G. YAMEY

ACKNOWLEDGEMENTS

The investigations described in this dissertation were carried out in the Department of Electrical Engineering at the University of the Witwatersrand.

The author wishes to express his sincere thanks to Professor H.E. Hanrahan and Mr A.J. Walker for their constant support and helpful supervision.

The author also wishes to express his gratitude to the Council for Scientific and Industrial Research for their financial assistance.

TABLE OF CONTENTS

TABLE OF CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CHAPTER 1

AN INTRODUCTION TO TRAFFIC SIMULATION

1.1 General

Road traffic congestion is a problem in large cities throughout the world. Delays to vehicles are costly to commence and industry in terms of lost time and productivity. Studies of traffic behaviour must be made to improve road network design and traffic control policies, so as to minimise delays and improve traffic flow. Road traffic is a complex phenomena and a rigid mathematical approach cannot be applied except in a few simple and idealized cases. Simulation techniques are usually used.

Simulation of traffic involves a number of separate operations. The first operation is to define the geometrical layout of the traffic network. The method of defining the network is dependent on the type of simulator used. In the case of a software simulation (see 1.2), it may take the form of data punched on cards in some specific format or in an interactive question and answer manner. Hardware simulations (see 1.3) use a patch panel to define the network.

In the second phase, data must be entered into the simulator. This data provides information as to the flow rates, speed and other vehicle parameters, as well as timing data for signalized intersections.

The third phase involves the actual modelling of traffic. Digital processing is used to simulate traffic events and interactions which can be broken down to logical decisions. Although traffic flow is a continuous phenomenon, it may be digitally simulated by use of time

slicing techniques. Small time increments called epochs are used and the model is updated every epoch. If a small enough increment of time is used, the simulation can be regarded as continuous. Random number generation is used extensively in the modelling phase to enable decisions to be made on a statistical basis as traffic is essentially a stochastic phenomenon.

The final phase, which may take place concurrently with the third, is the representation of the results of the simulation. This could be in the form of printed results or a graphical representation.

Two broad categories of traffic modelling have been used to date - macro and micro modelling. In a micro model every vehicle in the system is unique and its speed, position, destination and type is known at each instant. It is thus possible to follow an individual vehicle from the time it enters the system to the time it is discharged. Vehicle dynamics like acceleration and are rigidly modelled. Examples of this technique can be found in the literature [1,2,3,4,5], the most sophisticated being the UTCS 1 Network simulation model [1]. Micro modelling provides results which closely approximate real life traffic provided great care is taken in gathering data and in validating the model. Micro modelling requires a large computer due to the large amount of storage needed. The speed of simulation is low due to the number of calculations and decisions to be made in updating the model.

Macro modelling is based on the assumption that vehicles can be treated as if they are identical. Individual vehicles cannot be distinguished and vehicle dynamics are modelled using a statistical approach. Research [6,7] has shown that this assumption is valid under near saturation conditions and up to twenty percent heavy vehicles in the network have little effect on the traffic flow. Less com-

prehensive data needs to be gathered for macro modelling than for micro modelling but the results of the simulation can only be used for comparative testing rather than for obtaining accurate predictions of traffic below. Macro modelling requires relatively little storage capacity as extensive data is not required for each vehicle. Less computation is required for modelling the traffic so that high simulation speeds can be obtained.

1.1.2 Desirable Features of a Traffic Simulator

(a) Low Cost of Simulations

Simulation is often used as an interactive design tool using a trial and error approach. This implies that a large number of simulation runs may be required to arrive at the final design. Thus the cost of a design is dependent on the cost of a simulation run. This cost is related to the speed of simulation and the initial capital cost or rental of the simulator.

(b) High speed of Simulations

Apart from the fact that speed and cost are interrelated, high speed of simulation is desirable from the user's point of view. Since the simulator may be used for interactive design it is important that results are presented to the user as quickly as possible.

(c) Easy to Use

The simulation system must be easy to use and must not require specialised training or knowledge. Input and output to and from the system must be in terms and units meaningful to the end user.

(d) Comprehensive Basic Simulation System

A powerful system is required so that most common traffic networks can be modelled. This implies that multiple intersection networks and dynamic control policies should be catered for in the basic system.

(e) Ease of Development

The simulation system must be modular so that additional features may be easily added without major modifications to the system.

The objective of the present work was to design a cost effective simulator for interactive design with features as described above.

1.2 Software Simulators

The term software simulator is used to describe a computer program that models traffic conditions. It is the most widely used simulation technique and macro and micro modelling can be used. Large computer systems are usually required and networks of up to 200 intersections can be simulated. An exception is the model of Beilly [4,5] which is a micro model using a mini computer. However, only limited size networks can be simulated. This model had a low simulation speed (simulation time equals real time [5]). A recent trend [8,9] is to use large time-shared computers for interactive simulations. By using a bureau system many can have access to a simulator without incurring the capital cost of a large computer. Time sharing, however, is not well suited to applications which involve a great deal of calculation and data processing as in the case of simulation. In a time-shared system each user is only allocated a fraction of the available central processor (C.P.U.) time. Thus while a simulation may require only ten minutes of C.P.U. time, the user may have to wait considerably longer than that for the simulation to run. Interactive simulation using this type of system is tedious and time consuming.

One of the most important considerations in the design of a software simulator is the choice of programming language. A proliferation

of scientifically orientated high level languages exist, e.g. FORTRAN, ALGOL, PL1 and BASIC. These are general purpose languages and are available for many different makes of computers. They are attractive choices as they are largely machine independent, widely known and easy to use. In addition to the above there are simulation oriented languages such as GPSS, GASP, SIMULA and SIMSCRIPT. These languages are not as universally available as the general purpose languages but are more simulation orientated which reduces the programming effort. Most of the traffic simulators in the literature [1,4,8,9] use FORTRAN or one of the simulation languages [2]. A common failing of these simulators is a low real time to simulation time ratio. This is due to the generality of these languages which results in relatively inefficient code being generated for the particular problem. In addition most of these languages do not permit very basic machine operations such as bit and byte manipulations which can be very useful when simulating traffic.

Software simulators are flexible and simulation systems can be relatively easily modified to cater for different traffic situations. The computational power of a computer is useful in modelling dynamic control policies.

The major disadvantages of software simulations is the high cost of simulation. The cost of simulation can be reduced if it is possible to increase the speed of simulation without having to invest in more computer power. One of the aims of the present work was to reduce the cost of simulation by this technique.

1.3 Hardware Simulations

Hardware simulators are special purpose computers which model traffic situations. Advances in computer technology have made the heardware simulator an unattractive alternative to software, however,

the former is still of historical interest.

In 1961, the University of Manchester Institute of Science and Technology (UMIST), started to develop a hardware traffic simulator [10,11]. At that time the hardware approach was attractive due to the high cost and slow execution speeds of available computers. The initial system [10,11] could simulate a single intersection with fixed time traffic signals at a speed of fifty times real time, which was far in excess of available software simulator speeds. The system was expanded by adding core memories and multiplexing the basic simulator so that up to sixteen linked intersections could be simulated [12]. In order to simulate dynamic control policies a computer was interfaced to the model [13]. Although this arrangement provided most of the facilities required, the speed of simulation was reduced due to the time required for transferring data between the memory, the computer and the simulator. In addition, during this time, computer execution speeds increased and prices dropped dramatically resulting in the hardware simulator becoming an unattractive alternative to software simulators.

The hardware simulator is of interest, however, as it demonstrated that relatively simple hardware logic can effectively model certain traffic phenomena. This is especially true in the case of a macro model. Functions which are especially suitable for h dware implementation include the modelling of vehicles moving along a street, as well as vehicle and data generation. A shift e,.ster is effective in modelling a street as the positions of all the vehicles in the street can be updated with a single shift operation. Hardware random number generators can be used for vehicle and data generation and are inherently faster than similar software implemented generators.

1.4 System Design Philosophy in Present Work

The design of a simulator is governed by the three basic considerations given below. In addition, this work was contrained by the available computer and peripherals (see Fig. 1.1).

(a) Type of simulator

Hardware and software simulations were reviewed. It is apparent that both types have inherent limitations. The hardware approach was rejected due to high initial cost, development effort and inflexibility. In addition it is difficult to use for simulating multiple intersections and dynamic control policies. The software approach usually results in a poor simulation to real time ratio and simulations are costly.

If the speed of a software simulator could be improved, a powerful simulator would result. This can be achieved by improving the software (see 1.4(c)), and implementing some of the simulation in hardware. An examination of the operations which are required for traffic simulation, showed that the slowest single operation in software is the generation of random numbers. The work on hardware simulators showed that this operation is fast and easily implemented in hardware. While other functions may be implemented in hardware it was found that the increase in overall speed would be negligible.

Thus the basic choice of simulator consists of a computer with a peripheral hardware random number generator. This results in a simulator which has the flexibility of software whilst having a high simulation speed.

(b) Type of Computer

The choice of the type of computer is, to a large extent, constrained by the requirement that a special purpose peripheral is used. In addition, since time sharing and batch operation are unsuitable for

interactive operation, a dedicated computer is required. These two factors rule out the use of a large mainframe computer, as interfacing peripherals, is complicated and the capital cost is too high.

A Data General Corporation NOVA 1200 mini computer was available for this work and was found to be suitable.

The low capital cost and simple interfacing requirement makes mini computers ideal for use as dedicated interactive traffic simulators.

(c) Programming Language

Various programming languages were reviewed in 1.3. Of the high level languages FORTRAN and BASIC are available for the computer used. None of the special simulation languages are available. Of the above two, only FORTRAN can be considered as BASIC is an interpretive language and is slow in execution. Although the efficiency of modern FORTRAN compilers is relatively high, this language was not used as it is inefficient in core utilisation for this specific application. In addition, the hardware traffic simulator has shown that basic machine operations, which are not available in FORTRAN, are useful in simulating traffic. These include bit manipulation, logical operations such as AND and OR, and bit shifting. Thus the program for the present simulator was written in NOVA Assembly Language which resulted in a high speed simulator, requiring less than 16K words of memory (see Chapter 4).

The type of modelling used, i.e. macro or micro modelling, is also influenced by the choice of computer. Micro modelling was rejected as it requires a large amount of computer memory, is slow in execution and requires extensive data gathering and calibration. The additional accuracy of micro modelling is unnecessary for comparative testing of control policies, and is not required if generalized control strategies are to be investigated.

There are two disadvantages in using an assembly language for programming a simulator. An unavoidable problem is that the program is not machine independent. This is not so serious, however, since a dedicated computer is used and the peripheral hardware generators are also machine dependent. The second drawback of the language, is that a specialist programmer is required. In this work an effort has been made to minimise the effects of this disadvantage. The program has been written in such a way that the end user of the simulator does not require a knowledge of computer programming. The definition of the network and data for the model are entered in a free format in terms which are familiar to the user. The task of the specialist programmer has also been eased by the use of a simulation executive which is part of this work. The executive allows individual traffic element subroutines to be added or deleted without affecting the operation of the system. The programming of these routines is facilitated by the executive performing routine tasks such as data conversion and input output.

The overall configuration of the system is shown in Fig. 1.1. It is basically a program development system and the only essential peripherals required for simulation, are the hardware random number generators, disk and keyboard-printer. A graphic display terminal was not available but would be a useful addition to the system.

1.5 Review of Present Work

The potential advantages of a hybrid software and hardware traffic simulator have been described. The work that follows shows that interactive simulation is provided for and the simulation is easy to use. The cost of simulation is low as an inexpensive minicomputer is used. The average real time to simulation time ratio is 200 per

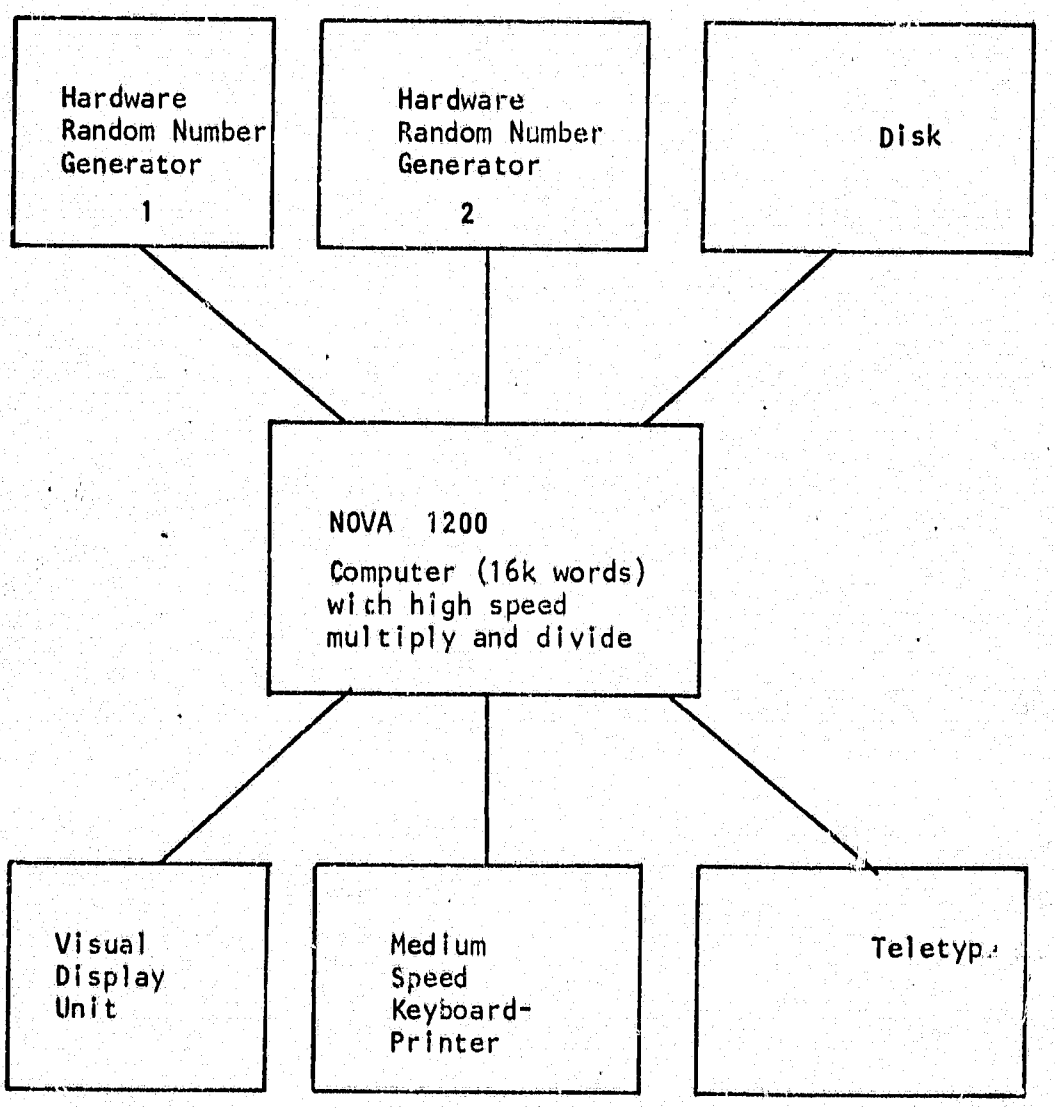

Figure 1.1. Traffic Simulator Configuration

intersection, compared with 1 - 18 for comparable simulations found in the ature [9,14].

A ma r portion of the present work was the design of a simulation executive. The executive is modular and general, and can be used for simulating systems other than traffic. The executive provides facilities for the user to easily define the network, enter data and run the simulation. The executive is used primarily for building up a unique program from the definition statements entered by the user. The program is constructed by linking a number of simulation subroutines together in the correct sequence. The simulation subroutines are not part of the executive and any type of simulator can be constructed by adding suitable routines to the executive.

The description of the executive has been divided into two chapters. This unorthodox approach was used so as to cater for two distinct types of users. The first group is the traffic engineer who wishes to use the executive and the simulation routines described in Chapter 5 for simulating traffic problems. This user does not need to be a computer specialist and the description provided in Chapter 2 is an outline of the basic concept, design, and operation of the executive. This description is sufficient to enable a simulation system which has been configured with the executive to be used.

Chapter 3 is intended for the computer specialist who wishes to expand the present system or write simulation routines for different problems. Details of how simulation routines must be written are provided. It is assumed that the reader of this chapter has a knowledge of the Nova Assembly Language. Details of how to load the present simulator, generate a new simulator and console procedures are given in the appendices.

The high speed of the simulator was partly due to the use of hardware data sources. A special data source was available in the Department of Electrical Engineering and was used for generating random numbers with arbitrary frequency distributions. A second generator was designed and constructed for generating the special case of the Poisson distribution. This generator produces sixteen independent outputs in less time than is required to execute a single computer instruction. A full description of this generator, as well as an outline of the first generator is given in Chapter 4.

Chapter 5 describes simulation routines which form, in conjunction with the executive, a basic traffic simulator. Using these routines, a network commonly encountered in cities may be simulated. The routines provide facilities for simulating streets, entry streets and intersections and are intended as a basis for future work.

Tests on the simulation routines are given in Chapter 6 and validation of the simulator is discussed. The use of the simulator is illustrated by a simulation of a network containing four intersections. This network was also used for determining the speed of simulation. Comments on the present work and suggestions for future work are given in Chapter 7.

CHAPTER 2

A FLEXIBLE SIMULATION EXECUTIVE

2.1 Introduction

A simulation program can be divided into two distinct but inter-related sections - the executive and the simulation routines. The executive is the controlling segment of the program. It provides facilities for defining the traffic network, entering data and supervises the running of the simulation. The overall operation of a simulator is defined largely by the design of the executive. The ease of operation from the users point of view, speed of simulation and format of the simulation routines are related to the design of the executive. Thus in the present work a large amount of effort was devoted to the executive program.

In the case of a simulation program which is written in Assembler language, the executive should be designed to facilitate programming of the simulation routines. This can be achieved by programming the executive to perform certain routine functions such as input, output, number conversions and interrupt servicing. Thus these repetitive tasks do not have to be coded in the simulation routines, relieving the programmer of considerable effort. These facilities minimise the disadvantage of using an Assembler language.

2.2 Requirements of a Simulation Executive

The requirements for a traffic simulation executive are given below.

(i) Minimisation of programmer's effort

A function of any executive is to relieve the simulation routine programmer from writing code for routine repetitive tasks. This is particularly important when application programming is done in Assembler language. The executive should raise the level of the normal Assembly lanugage to a type of macro assembly language. Macros are defined in the executive to perform routine tasks. The application program simply calls the appropriate macro with a suitable instruction and the task is performed by the executive.

(ii) Introduction of minimal core overhead

The executive should require as little memory as possible so as not to significantly reduce the memory space available to the user program.

(iii) Simple definition of the network

The executive should provide facilities to enable a traffic engineer to specify the layout of the network he is studying. It should not be necessary for the simulator user to be a computer specialist. The method of specifying the network should be in terms relevant and meaningful to the end user.

(iv) Provide fast interpretation or compilation of the network definition

The executive should be able to translate the symbolic representation of the network into a program for simulating the network. The translation should be fast so as not to introduce a noticeable delay to the end user. If a compiler is used, it should be able to perform this operation with

only one scan of the input data so that the network definition may be entered directly from the keyboard.

(v) Flexible Compiler

The interpreter or compiler must be flexible so that the system programmer can specify the format of the network definition statements and control the interpretation of these statements. In other words the operation of the compiler must be under control of the simulation routines so as not to limit the scope of the executive or place constraints on the simulation routines.

(vi) Modular organisation

The whole simulation system must be modular so that additional features can be easily added. The addition or deletion of network elements should not affect the operation of other elements.

(vii) Simple data entry

Run time data for the model should be easily entered. This requires that a free format be used. The interpretation and manipulation of this data should be under control of the simulation routine to which this data is relevant. Error checking facilities and editing should be provided to minimise the effects of inaccurate data entry.

(ix) Interactive operation

As simulation is often used for trial and error design, interactive operation is important. Facilities should be available to the designer to run a simulation, evaluate the results, change a parameter and rerun. This also implies that the speed of simulation must be high.

(x) Communication between simulation routines

In traffic simulation it is important for various routines to communicate with one another. For example, an intersection element must be able to examine a street element to see if conditions in the street allow vehicles queued at the intersection to turn right. This facility can be provided by using a common data block (i.e. data that is shared between routines) or by providing facilities in the executive for one routine to gain access to another routine's storage.

2.3 Design of a Simulation Executive

2.3.1 General

A simulation oriented executive called FASP (Flexible Assembler Simulation Package) was designed to meet the criteria listed in 2.2. Although primarily intended for use in traffic simulation applications, the flexible, modular structure of FASP makes it suitable for simulating any discrete stochastic system provided appropriate simulation subroutines are added.

The design of FASP, as far as it affects the user of the traffic simulation program, will be described below. The detailed description of how simulation routines are written and interfaced with FASP and a description of the macro facilities of FASP are described in Chapter 3.

2.3.2 Definition of the Network

A fundamental segment of a simulation executive is the control program. This is a program that links the various simulation routines together in such a way as to simulate the desired network. The design of the control program determines, interalia, the way a net-

work must be specified; how simulation routines may be added or deleted from the system; and how data is stored. It also affects the speed of simulation. Most simulation programs use a modular subroutine structure in which each traffic element e.g. vehicle generation, vehicle movement and vehicle discharge, is a separate subroutine. Different methods of linking these subroutines together have been used.

A common type of control program calls each of the various subroutines in turn. Each subroutine is entered once per epoch and all the vehicles in the particular situation are processed. For example, all vehicles are generated at the same time for all input streets in the network. The UTCS 1 simulation program [1] uses this technique and a simplified flow diagram of the control program is shown in Fig. 2.1. This method is efficient as each subroutine is entered only once, resulting in minimal time lost and subroutine calls and in subroutine initialisation. The disadvantages of this system are that a strict numbering system for various traffic elements is required or an elaborate method of interpreting the network definition data must be provided. A common data area is used for this method and this data must be scanned by each subroutine. This involves a considerable wastage of time as subroutines must scan all data rather than just the data which is relevant to them. The addition or deletion of a traffic element requires that the control program be modified so that a strictly modular approach cannot be used.

An alternative method was used by Beilby [4,5] for a micro model which was run on a small computer. In this method each word of computer memory corresponds to a section of roadway. The position of a vehicle is indicated by the vehicle's parameters (e.g. speed, acceleration, destination) being stored in the word memory corresponding

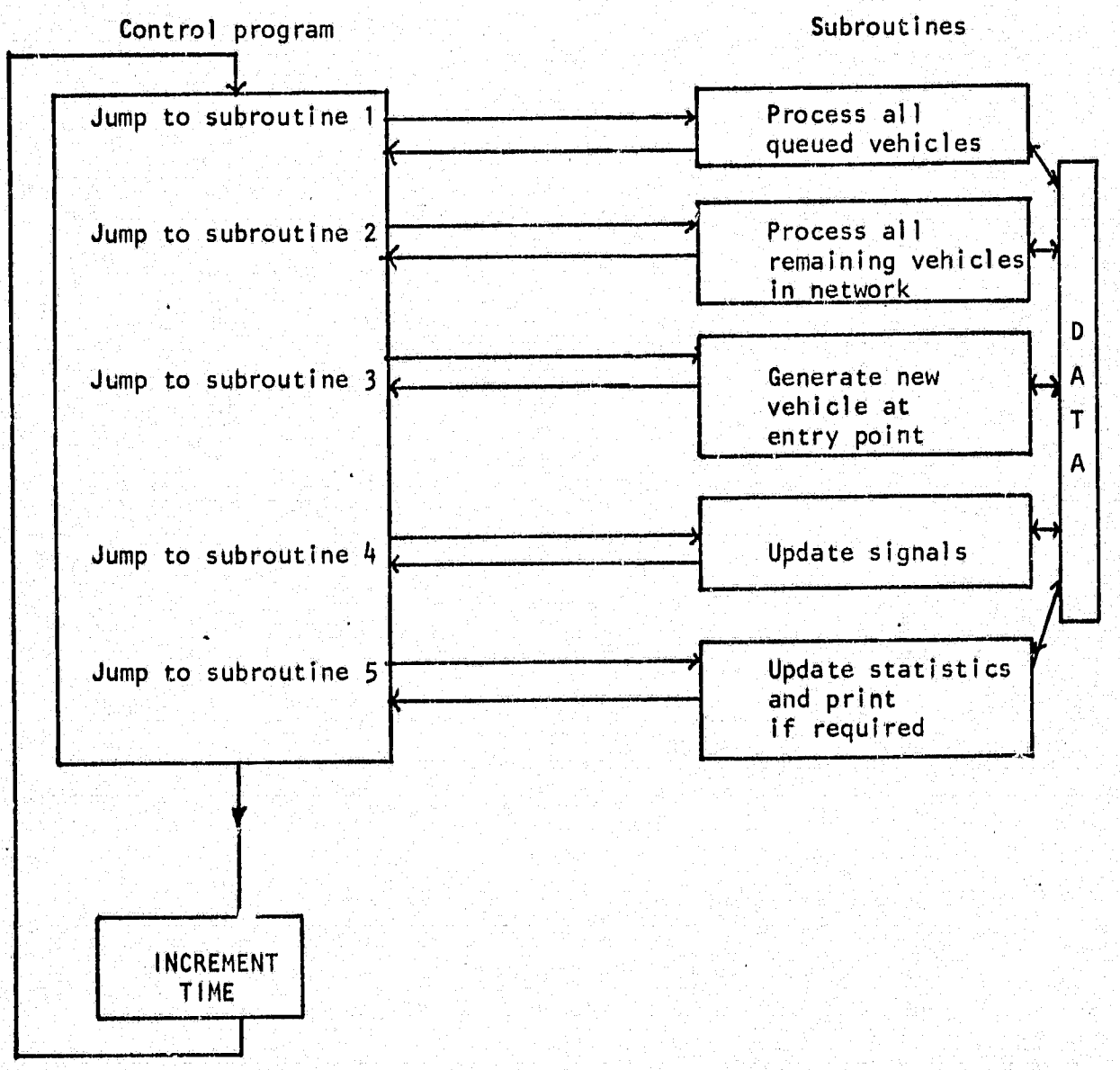

Figure 2.1. Flow Diagram of Control Program used by UTCS 1 Program

to its position in the road. In order to reduce the scan time of the road, the network definition was made part of the control program. The control program is shown in Fig. 2.2. The basic operation of the program is as follows: The computer "steps" along the road by executing a series of dummy instructions (no-ops) until a jump instruction (indicating the presence of a vehicle) is encountered. The jump instruction points to a subroutine which updates the vehicles parameters and position. This subroutine also moves the jump instruction to a new location to indicate the change in position of the vehicle. This procedure is repeated for all the roads in the network and for every epoch in time. Thus the control program is in effect a direct representation of the geometry of the network. This method is fast as no scanning of data needs to be performed.

An adaptation of the above approach is used in the present work. Since a macro model is used it is not necessary for the control program to model the detailed geometry of each street in the road network. The network is modelled by a collection of simulation routines which model lumped traffic elements such as streets and intersections. The control program is used only to call the simulation subroutines and pass required parameters to the subroutines in an order defined by the layout of the network. A generalised format of a control program is shown in Fig. 2.3. As an example consider a network shown in Fig. 2.4, consisting of an entry street feeding one leg of an intersection. Two simulation elements are needed. These are called subroutine ELINK (for entry street) and subroutine TLIGHT (for intersection) respectively. The basic control program layout is as shown in Fig. 2.5. Subroutine ELINK is first entered to generate vehicles and move them down the street. The parameter following the jump to subroutine instruction contains the address of the data for the

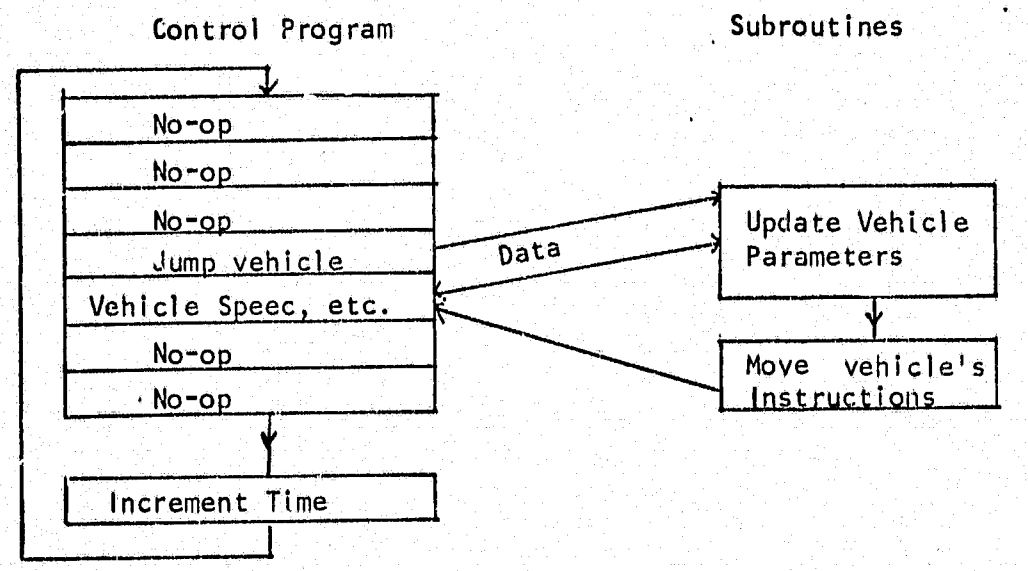

Figure 2.2. Control Program used by Bellby.

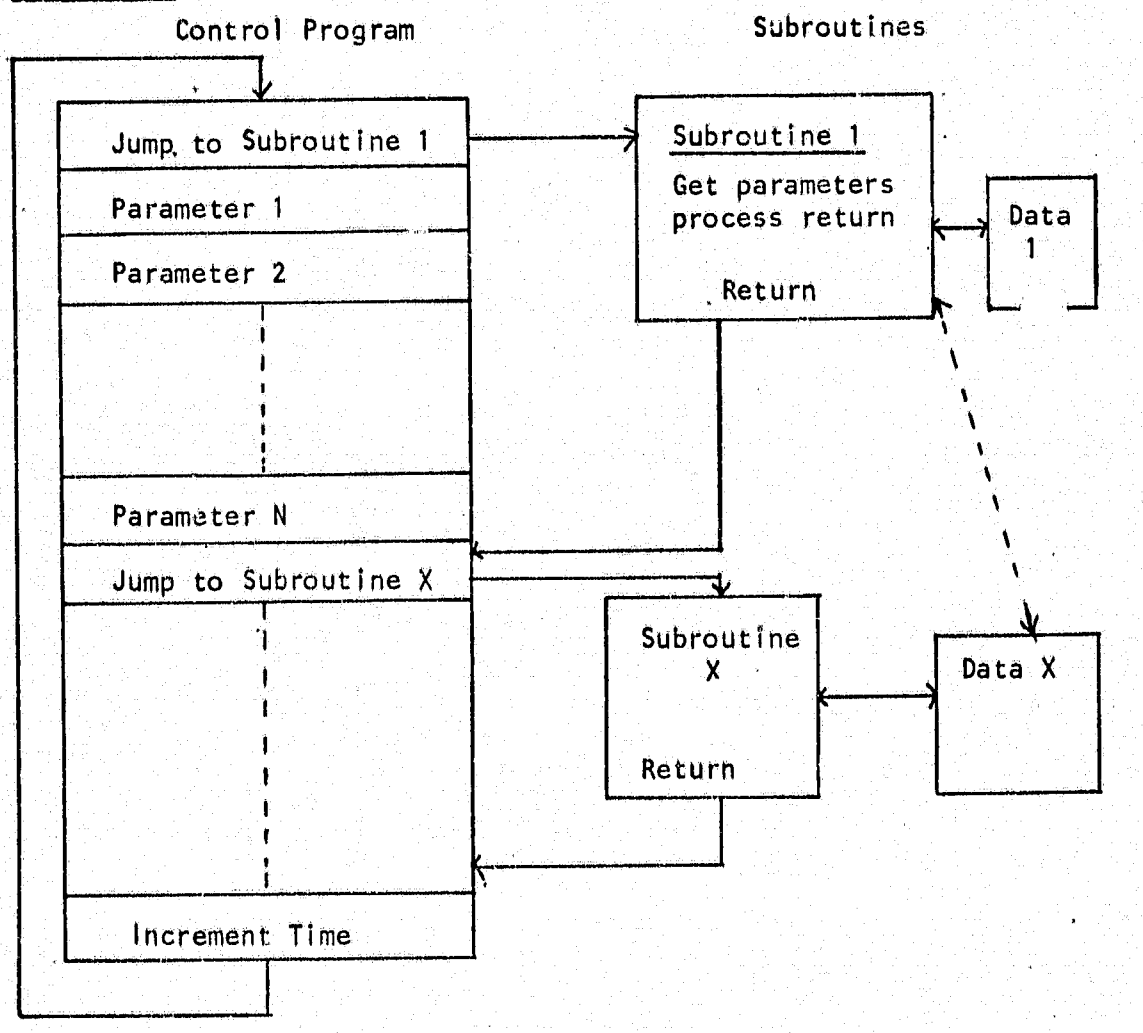

Figure 2.3. Control Program used in Present Work.

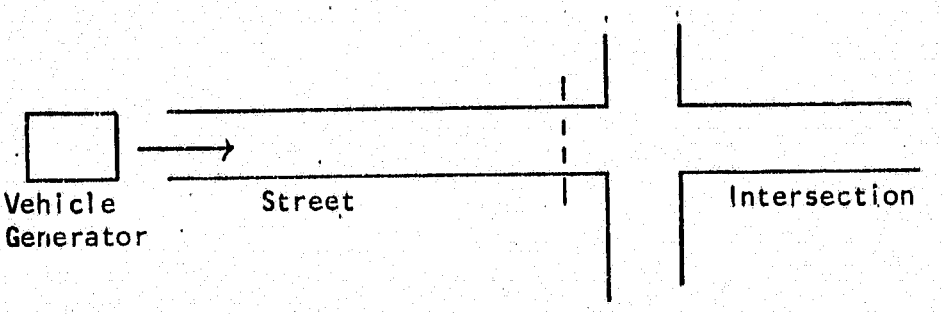

Figure 2.4. Example Network

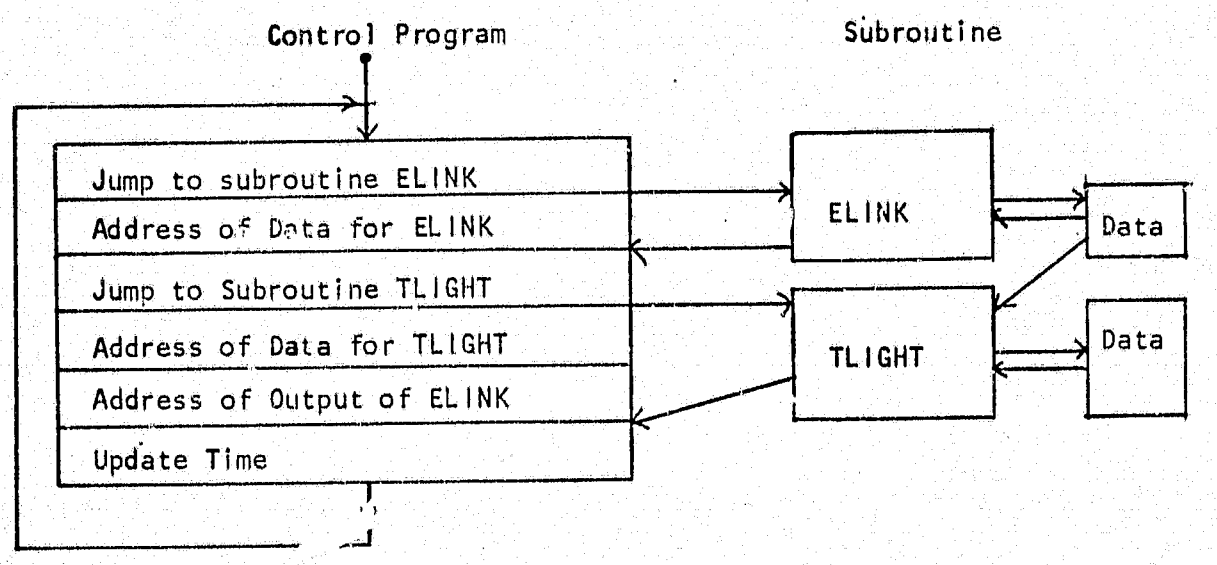

Figure 2.5. Control Program for Network given in Figure 2.4.

specified entry link. On exit from the ELINK subroutine, the intersection routine is entered. Two parameters are required here; the address of the data for the intersection routine and the address of the output of the ELINK routine. This output indicates whether a vehicle has been discharged from the street into the intersection.

In the case of a general network containing several elements such as streets and intersections the control program will consist of a number of calls to each subroutine. The number of calls correspond to the number of times an element appears in the network. For each call a unique data address is given. Thus each element in the network has a dedicated block of data and can, if required, access another elements' data as in the case of the TLIGHT routine (See 5.5).

2.3.3 System Operation

The system operates in two distinct modes - 'Program' and 'Run'. The program mode is used to set up the control program to correspond to the network being simulated. The run mode is used for entering run time data and for running the simulation. The two operations which are part of the run mode are described separately for clarity.

The program mode shown in Fig. 2.6 is initiated when the simulation system is loaded (see Appendix 1), or when a specific executive command (/P) is given (see 2.4.3). A special routine in FASP called the compiler is then entered. The compiler accepts network definition statements from either the console (see Appendix 2) or from a file stored on disk. If the statements are to be read from disk, the user types a load command followed by the appropriate file name (see 2.4.4). The compiler operation is unaffected by the source of the statements (console or disk). The compiler examines the input stream and the statements are compiled as soon as they are entered.

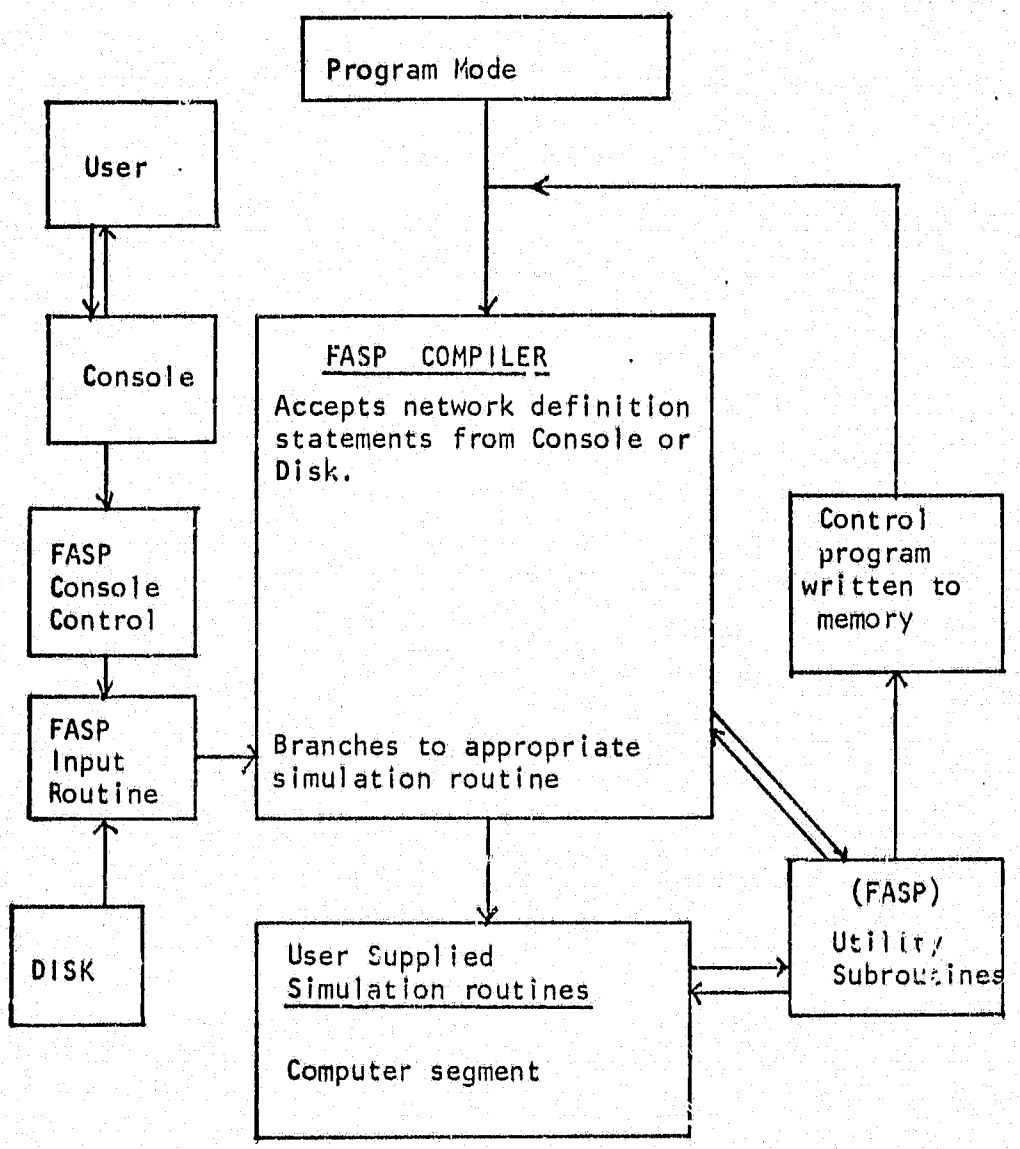

Figure 2.6. Program Mode

The compiler first checks if the statement entered is a system control statement, such as the end (/E) statement (see 2.4.3), and acts on it immediately. If it is not, the compiler calls a routine in utility subroutine library of FASP which scans through the simulation routine's names. If the first word of the statement does not match any of the simulation routine names an error message is printed and the statement is ignored. If a match is found the appropriate simulation routine compiler segment is entered. The compiler segment obtains the parameters (if any) from the definition statement by calling appropriate routines in the utility subroutine package. From the parameters, the simulation routine compiler builds up a section of the control program which is then written to the control program memory area by another utility subroutine. The above process is repeated for each statement entered until an end statement is encountered. A flow chart of this process is given in Fig. 2.7.

The FASP compiler does not have any control over the translation of the definition statements. The main function of the FASP compiler is to accept the definition statements and then pass them to the required simulation routines. The FASP compiler also checks that the number of parameters given in the statement corresponds to the number required by the simulation routine. If there is a difference in the number of parameters an error message is printed and the statement is ignored.

The translation of the definition statement into a section of control program is a function of the user supplied simulation routine (see 3.3.2). This requirement 2.2(vi) that all compilers be flexible, is satisfied.

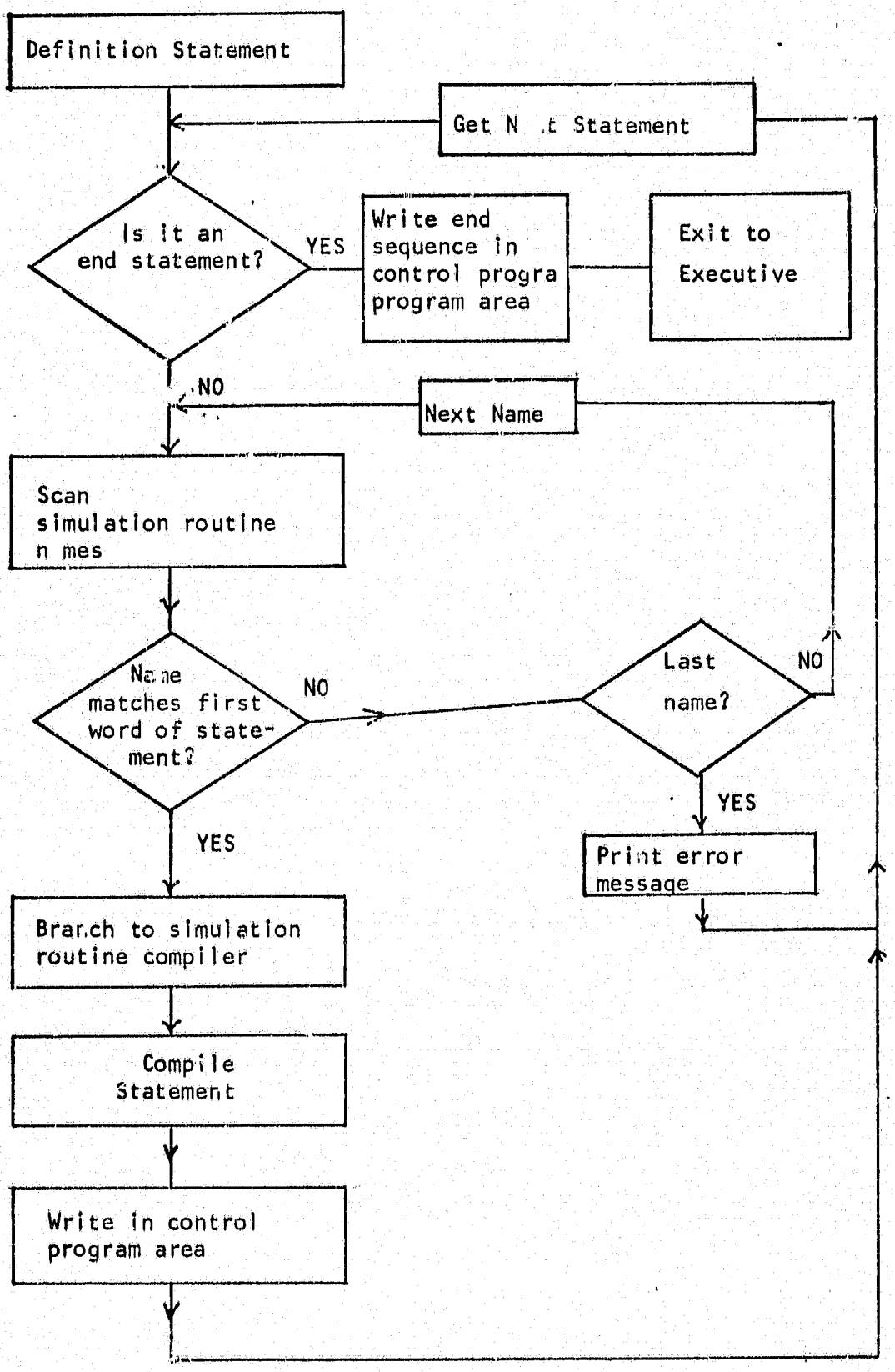

Figure 2.7. Flowchart of Definition Statement Interpretation

The end statement (/E) terminates the program mode and the run mode is started. The procedure for run time data entry is similar to the procedure used for network definition. Again the data may be entered from the console or from disk. A diagramatic representation of the process is given in Fig. 2.8 and a flow chart is given in Fig. 2.9. The run time executive first checks if the statement is a command and processes it (see 2.4.3). If not an executive command, the simulation routine names are scanned for a match with the first word of the statement. If no match is found an error message is printed. If a match is found the appropriate simulation routine data segment is entered. The data segment uses routines in the utility subroutine library to access the parameters in the statement. The data segment processes the parameters, if necessary, and stores the results in the data area. A return to the executive is then made and the next statement is processed. Again the user supplied simulation routine has full control over the interpretation, preprossessing and storage of the parameters in the data statement (see 3.3.2).

When a run command is given (see 2.4.3) the executive transfers execution to the control program which was built up in the program mode. The interaction of the various parts of the system is shown in Fig. 2.10. Generally the control program calls simulation routines in a required sequence. The simulation routines receive or send data to the peripherals, i.e. hardware random number generator, printer and disk. The control program and the simulation routines use and modify run time data. The heart of the control program is a loop which is automatically generated by the compiler. The number of times this loop is repeated is specified by the user when the run command is issued. On completion of the required number of loops,

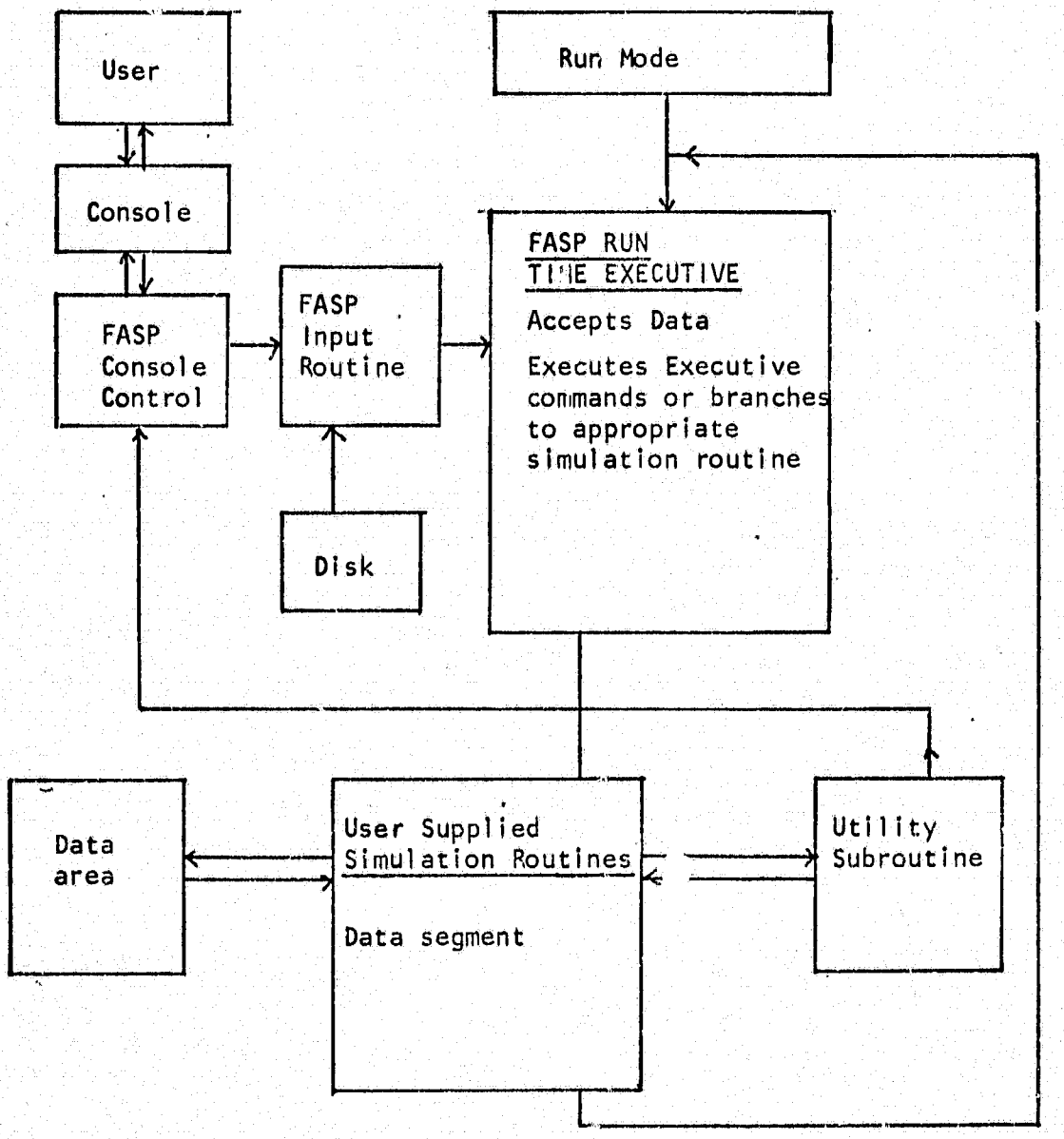

Figure 2.8. Run Mode - Data Entry

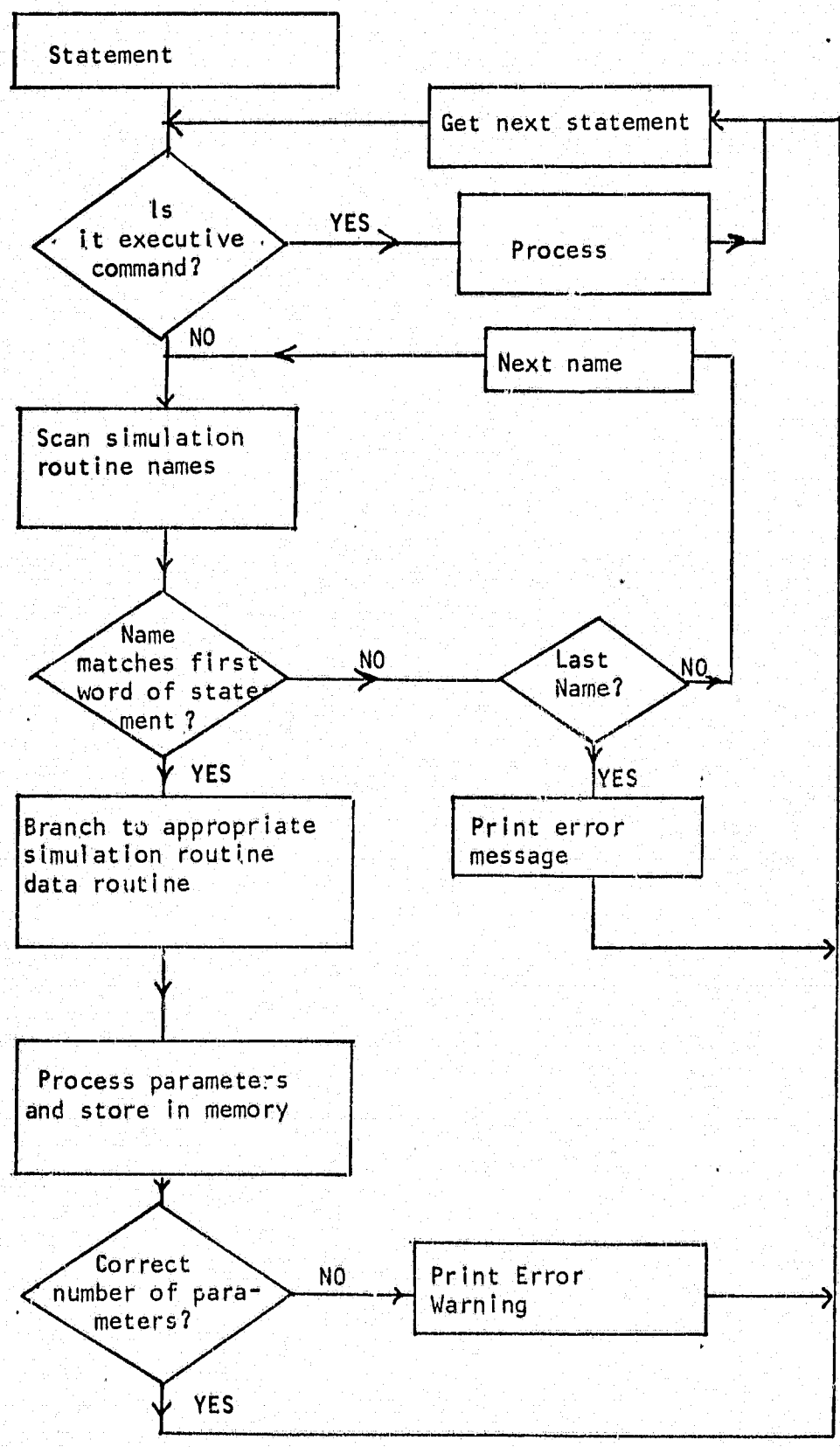

Figure 2.9. Flowchart of Run Mode Data Entry.

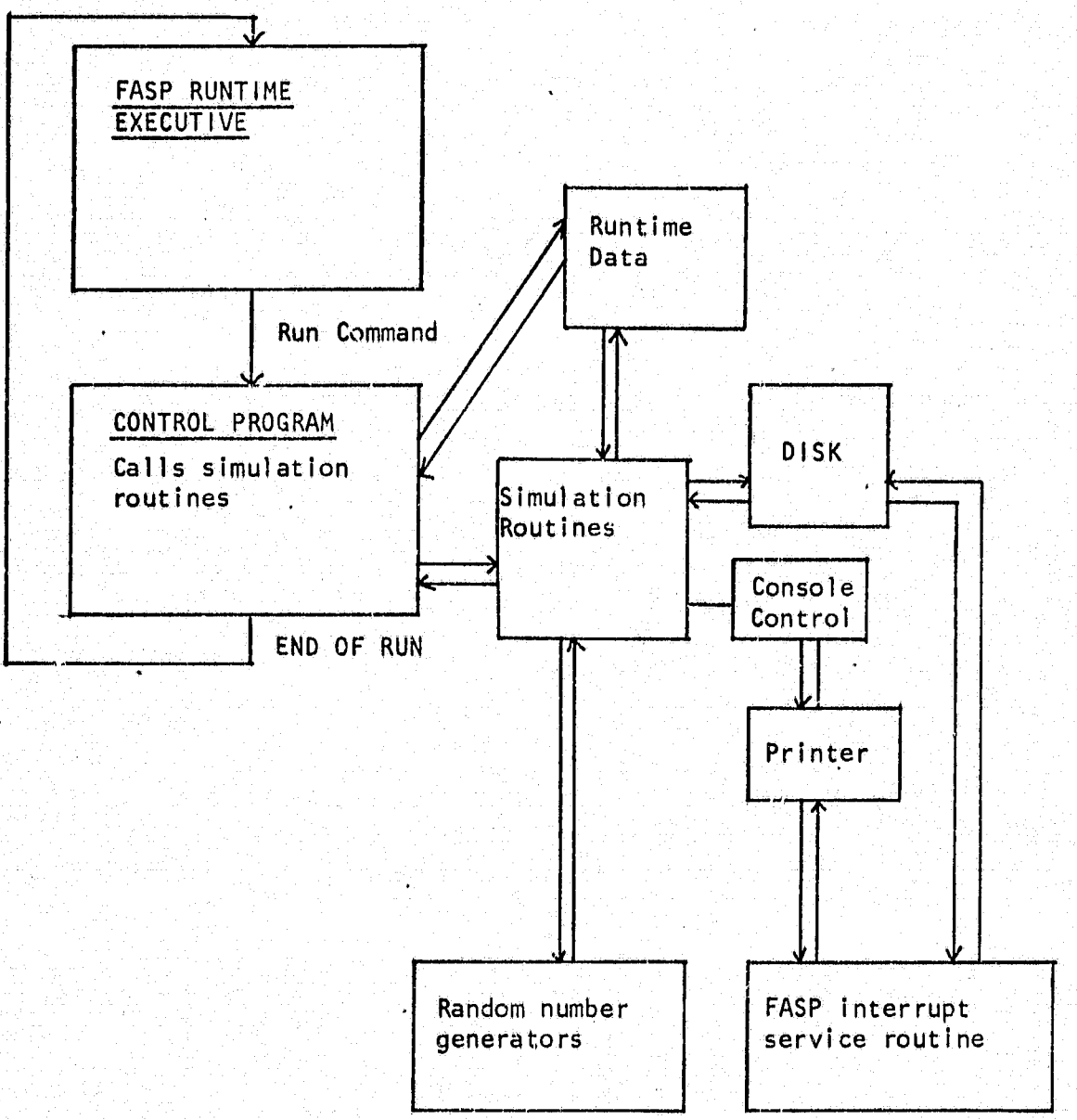

Figure 2.10. Run mode - Simulation Run

control is transferred back to the executive and more data or commands may be issued by the user. FASP has an interrupt service routine so that the disk and console can operate under interrupt control.

Disk input-output and printer output is buffered and the buffers are read and written under control of the interrupt service routine. This means that the program does not have to wait for these devices to become ready and can continue executing. The interrupt service routine also allows the user to halt a program before it has finished executing and return the control to the run time executive. Data may then be entered and program execution may be continued.

2.3.4 Storage Allocation

One of the features of FASP is the allocation and management of data storage. This simplifies the programming of simulation routines.

FASP has two types of storage areas. The first is called "Constant" storage and the FASP executive cannot modify data in this area. This area is used for storing constants such as traffic flow rates and signal timing data.

The second type of storage area called "Initial" is provided specially for traffic simulation. The data in this area can be initialized to zero. The simulation routines can use this area of storage for streets and queues which need to be emptied before a simulation run is started or restarted. This data area is only initialised when the initial command (see 2.4.3) is given. Thus the user has control over whether the network is to be emptied of vehicles before a rerun of the simulation.

2.4 Use of FASP

2.4.1 Network Definition and Run Time Data Statements

The network definition statements and run time data statements format is largely a function of the programming of the simulation routines. However the general Format of these statements is defined by FASP. The general form is as follows:

NAME, NUMBER, DATA ELEMENT 1, DATA ELEMENT 2, . . . DATA ELEMENT N.

The NAME is a name given to the simulation routine element and identifies the simulation routine to which the statement refers. NAME must appear exactly as it is defined in the simulation routine and must appear before the first comma.

NUMBER is an optional parameter and is used when several elements of the same type appear in the network. For example a network can consist of many streets. If the simulation routine for modelling streets is called LINK the different streets would be allocated numbers so that one can be differentiated from another. The statement LINK, 3, ... would refer to street number three, and the data elements entered would pertain to that street.

DATA ELEMENT is a number, alplanumeric string or another NAME which is to be passed to the simulation routine. The number of data elements in a statement depends on the simulation routine. Carriage return must be entered after the last data element.

FASP checks each statement to see that the number of data elements in the statement correspond to the number of data elements required by the particular simulation routine. The type of data element (NAME, number or string) is also checked for correspondence. FASP also provides facilities for the simulation routines to request a variable number of data elements (see 3.2.1). The comma which is

used as a delimiter between data elements can be replaced by a semi-colon to indicate options to the simulation routine if the simulation routine is programmed to expect it.

FASP places few restrictions on the format of the statements. By careful choice of the NAME and sequence of the data elements the programmer of simulation routines can ensure that the statement is meaningful to the user of the simulation system. Examples of statements for traffic simulation are given in Chapter 5.

2.4.2 Control Program Structure

The control program is built up from the network definition statements. Thus the network definition statements can be considered as a symbolic representation of the simulation program. In a simulation program it is desirable to have three phases. The first phase is used for initializing variables, resetting peripherals and opening files. The next phase is the actual simulation phase. The last phase is used for closing files, and printing final results.

FASP provides facilities for segmenting a simulation control program into the three phases and allowing the user control over the various phases. The program is segmented into the phases by using the keywords. INITIAL, DYNAMIC, and TERMINAL as shown in Fig. 2.11.

The INITIAL phase comprises all the statements which appear between the words INITIAL and DYNAMIC. These statements are executed when an initialise command is given. The statements are executed once only and control returns to the FASP executive allowing the user to issue a further command.

The dynamic phase comprises all the statements between the words DYNAMIC and TERMINAL. FASP sets up a program loop so that these statements can be executed repeatedly.

INITIAL

Statements executed unconditionally when an INITIALISE (/I) command is given

CONTROL RETURNS TO FASP EXECUTIVE

DYNAMIC

Main program

These statements are executed in a loop after a RUN Command (/R) is given. The number of loops is specified by the user.

TERMINAL

Statements executed unconditionally after the DYNAMIC loop has been completed.

CONTROL/RETURNS TO FASP EXECUTIVE

/E End of program

Figure 2.11. Basic Control Program Structure

Each execution of the loop represents one epoch of simulation. The number of times the statements in the loop must be executed (or number of epochs to be simulated) is specified by the user when the run command is given. On completion of the specified number of loops the TERMINAL section is executed.

The statements which appear between the words TERMINAL and the end statement are executed only once at the end of a simulation run. After the TERMINAL phase is completed control is returned to the FASP executive.

The three keywords INITIAL, DYNAMIC and TERMINAL must appear in the given order for the FASP system to operate correctly.

2.4.3 Executive Commands

FASP has several commands to control the running of a program. Commands consist of a / followed by an alphabetic code. Their actions are as follows:

/E This indicates the end of a user program and terminates the program MODE of FASP. FASP then enters the run mode. This command is undefined in the RUN mode.

/I Initialises all initial storage to zero and executes the initial segment of the program.

/R Runs the user program. The console types NO OF RUNS and the user enters the number of times he wants the DYNAMIC segment of the program to be repeated.

/P Returns FASP to the Program mode so that another program may be entered. Note: The previous control program in memory is destroyed.

/C Continues execution of program which was interrupted by the user (see below).

ESC A program that is executing may be interrupted by pressing the ESCAPE key. At the end of the current epoch the program halts and 'STOP AFTER N RUNS' is printed. Control is transferred to the executive, and data may be entered, or another command may be given.

2.4.4 DDOS 1.6 Disk Support

The present version of FASP is configured with support for the Decision Disk Operating System [15]. The disk support provides facilities for the user to store network definition statements and run time data on disk. These can be subsequently read from the disk and the simulation can be rerun. Data is stored on disk in ASCII code so that the data can be edited with the standard Data General Editor [16].

Various commands are provided so that the user can store data which is subsequently entered from the console, on the disk and read it back at a later stage. The commands can be entered whenever the FASP system is expecting data from the user.

The Commands conform to the standard format for statements (see 2.4.1).

Disk Commands

SAVE, FILENAME [,]

This command creates a file called FILENAME and all subsequent data typed on the console is written to disk. The same mode is terminated by an executive command (i.e. any call preceded by the character "/"). FILENAME must be a name that does not already exist on the disk or an error message will result. If the optional comma is present, the file "FILENAME" must exist and all subsequent data typed will be appended to the existing file.

GET, FILENAME[,]

This command reads a disk file and treats the data as if it had been entered from the console. If the optional comma is present the data read in it is not printed on the console.

KILL, FILENAME

This deletes the file called FILENAME from the disk.

BYE

Returns control to the DDOS operating system.

2.4.5 Error Messages

Error messages are printed in a standard format similar to that of BASIC. The format is:

ERROR N.

N is the error number and is in the range of 1 - 99. Error numbers in the range 1-19 are system errors and the remainder may be used by the programmer for errors which may occur in simulation routines. Certain errors (example memory overflow) are fatal to the system and require that the source of error be corrected and the system must be reloaded. All other errors merely result in the operation in which the error occurred to be ignored. A list of error messages and their meanings is given in Appendix 3.

2.5 Summary

The FASP executive provides facilities for the user to easily simulate a traffic network once the simulation routines have been provided. FASP provides the user with a means of defining the network, entering run time data and running the simulation.

FASP is flexible in that the programmer of the simulation routines has control over the format and interpretation of user statements and can thus tailor the system to meet his requirements.

CHAPTER 3

DETAILED STRUCTURE OF FASP

3.1 Introduction

The concepts and overall view of FASP was given in the previous chapter. In order to modify the system or to add simulation routines, a more detailed description is required.

The interfacing requirements for simulation routines is explained in this chapter. Since simulation routines are written in Nova Assembly Language, a knowledge of the Nova Extended Assembler [17] is required for a complete understanding of this chapter.

The programming of simulation routines is made easier by various routines contained in FASP. These routines perform input-output, storage manipulation and code conversions. The use of these routines is first described and then the layout of the simulation routines is given. Although probably not required for traffic simulation, a description on how to write interrupt service routines is given for completeness.

The modular structure of FASP, and the use of pseudo instructions is similar to the approach used by Gordon for extending the Data General BASIC language [18].

3.2 Pseudo Instructions

The utility routines which are defined in FASP are intended to create a more powerful or higher level Assembly Language for programming of simulation routines. The utility routines have been given symbolic names which effectively increase the instruction set of the

Nova Assembler. The symbolic names are translated by the Assembler and Loader into Jump to Subroutine instructions. Since FASP and the simulation routines are relocateable and are assembled separately, some means of linking the two together is required. This is done by using the facilities of the Nova Extended Assembler. The symbolic names of the routines are defined in FASP to be Entry points (.ENT) and in the simulation routines to be external references (.EXTN). This instructs the assembler that symbols which are defined as external references are to be resolved at load time. This is illustrated by the following example of FASP and simulation routine coding.

FASP

```
.ENT GET                    ;   define entry point
GET.= JSR @.GET             ;   define pseudo instruction
.GET:  GET1                 ;   indirect address
GET1:  . . . .              ;   subroutine for executing
                                pseudo instruction
```

Simulation Routine

```
.EXTN   GET                 ;   define external reference
                            ;   simulation routine coding


GET                         ;   pseudo instruction
                            ;   loads as JSR @ .GET
```

The pseudo instruction GET above is translated by the relocating loader [15] into a JSR @.GET instruction. The symbol @ in the instruction denotes indirect addressing and is used because only 256 words of memory can be directly addressed in the NOVA.

All the pseudo instructions use the above format. Some of the pseudo instructions require certain parameters. Those parameters are passed to the routine in FASP via the accumulators AC0 to AC3. Parameters are also passed back to the simulation routine via the accumulators as illustrated below.

```
LDA   0,PARM1          ;   LOAD PARAMETER  1
LDA   1,PARM2          ;   LOAD PARAMETER  2
LDA   2,PARM3          ;   LOAD PARAMETER  3
GET                    ;   PSEUDO INSTRUCTION
                       ;   EXECUTION RESUMES
                       ;   HERE AFTER EXECUTING THE
                       ;   PSEUDO INSTRUCTION
                       ;   WITH RETURN PARAMETERS IN
                       ;   ACCUMULATORS.
```

The pseudo instructions are divided into 4 main classes, i.e. input, output, FASP control and miscellaneous.

3.2.1 Input Pseudo Instructions

FASP has an input routine which reads network definition statements of run time data from the console or disk into a buffer. Input pseudo instructions are provided so that simulation routines may extract parameters from the buffer. Code conversion (e.g. ASCII decimal to binary) is also performed as required. The parameters are extracted sequentially from the beginning of the buffer. Thus the first pseudo instruction refers to the first parameter, the second instruction to the second parameter and so on. The general form of the data or definition statement was given in 2.4.1. Parameters are separated from one another by break characters which may be either a comma, a semi-colon or a carriage return. The semi-colon may be used to denote certain options to the simulation routine. Whenever a parameter is

extracted from the buffer, the type of break character following the parameter is indicated by a code value which is placed in AC3. The code may then be tested by the simulation routine and appropriate action taken. This feature may be used to test for the end of a statement if a variable length parameter list is required. The codes returned in AC3 are as follows:

AC3 = 1	Break character was a semi colon
AC3 = Ø	Break character was a comma
AC3 = -1	Last parameter in list.

Before FASP transfers execution to a simulation routine after a a or definition statement, a GETAD pseudo instruction is automatically executed. The accumulators are loaded with data as below. Thus the first pseudo instruction issued by the simulation routine will refer to the first parameter after the routine name and number.

The available input pseudo instructions are as follows:

GETAD Get the addresses of a simulation routine given a routine name and a number if necessary.

On completion of the instruction

ACØ = address of the constant storage area for the routine

AC1 = address of the initial storage area for the routine

AC2 = Address of 1st word of the routine following the routine name

AC3 = Number of the routine

GET Convert the ASCII single precision decimal number in the buffer to a two's complement binary number

On completion

ACØ = binary number

AC1, AC2 unchanged

AC3 = break code

GETAS Fetch an ASCII string from the buffer and store it starting at an address specified by AC0. The bytes are packed from left to right in the word and the string is terminated by a null byte or word.

On completion

AC0 = String starting address

AC1 = Length of string (words)

AC2 = Unchanged

AC3 = Break code

GETBR Return the last used break code in AC3.

GETC Get a character direct from the console (not buffer); This is not normally used by simulation routines.

On completion

AC0 = 7 bit ASCII character in lower byte of the accumulator word

AC1,AC2,AC3 = unchanged

3.2.2 Output Pseudo Instructions

Output pseudo instructions are used to print messages or numbers on the console. In all cases the constants of the accumulators are unaltered following the instruction. The following pseudo instructions are available:

PUTC Print the ASCII character contained in AC0 on the console. The character must be in the lower byte of the accumulator.

PUT Convert the binary number in AC0 to ASCII decimal and print it on the console. The word following the pseudo instruction must contain a number in the range 1 - 5, to indicate the number of decimal digits to be printed. Leading zeros are suppressed and a number that cannot be represented by the format specified is printed as $ $ $

CRLF — Print a carriage return and line feed on the console.

MES — Print a text string on the console. The string must follow the instruction MES and two ASCII characters must be stored per word. The characters must be left justified in the word (.TXTM 1 format).

ERROR — Print an error message on the console. The word following the pseudo instruction must contain the error number. The message is printed with the following format.

ERROR NN

After printing the error message, control is passed to the executive. Error numbers 1 to 19 are reserved for system usage and numbers 20 - 99 are available for simulation routine error messages.

3.2.3 FASP Control Pseudo Instructions

The FASP control pseudo functions are included here for completeness but are referred to later in the sections on simulation and interrupt service routines. The following instructions are available

RETURN — This must be the last instruction in a data segment of a simulation subroutine. It checks that all the parameters in the data statement have been used and returns control to the executive enabling more commands or data to be entered.

NONE — This is not an instruction but an indirect address. It points to a routine which will print an error message (ERROR 6). It is used for simulation routines that have no data segments (see 3.3.2).

PUTP — Write instructions into the control program. This is used in the program segment of a simulation routine. The program segment is entered when a network definition state-

ment is entered on the consol. The program segment builds up the appropriate NOVA Assembly Instructions to be written into the control program and stores them starting from the second word following the pseudo instructions PUTP. The word immediately following PUTP must contain the address of the last instruction to be written into the control program. Since the instructions following the pseudo instructions are moved to a different part of memory, they must be unlocated, i.e. they must be able to execute anywhere in memory.
Thus indirect, or direct addressing of page zero must be used for these instructions. After the pseudo instruction is executed control is transferred to FASP. PUTP is the last executable instruction in the program segment.

TRAP — This pesudo instruction is rarely used and is only used by some interrupt service routines. It is used by interrupt service routines which use any of the other pseudo instructions. These cannot be used without first using TRAP as they are non re-entrant. TRAP is the only re-entrant pseudo instructions and is re-entrant up to 8 levels.

TRAP dismisses the interrupt and simulation continues till the current epoch has been simulated. The service routine is re-entered with interrupts off and the instruction following TRAP is executed. After the TRAP instruction the service routine may use any of the pseudo instructions. The service routine must end with a JMP Ø,3 instruction.

3.2.4 Miscellaneous Pseudo Instructions

These pseudo instructions are included to simplify programming of simulation of traffic elements.

The following instructions are available.

MULTIPLY Multiplies 2 single precision unsigned binary numbers to produce a double precision unsigned binary number.

A third number is added to the result.

On entry:

AC0 = number to be added to result (C)

AC1 = multiplicant (A)

AC2 = multiplier (B)

On completion:

AC0 = product (A x B + C) low order word

AC1 = product (A x B + C) high order word

N.B. No overflow checking is performed.

DIVIDE Divide a double precision unsigned binary number by a single precision binary number to produce a single precision binary quotient and remainder.

On entry:

AC0 = dividend (low order word)

AC1 = dividend (high order word)

AC2 = divisor

On completion

AC0 = quotient

AC1 = remainder

N.B. No underflow-checking is performed.

3.3 Simulation Routines

3.3.1 Introduction

Simulation routines are programmed in Nova Assembly Language and pseudo instructions. The routines must be relocateable.

A generalized simulation routine has three entry points which correspond to three separate segments of the routine, i.e. program, data and simulation segments. The program segment is entered when a network definition statement is entered and is used to write a section of the control program. The data statement may be used to manipulate parameters entered in a data statement and store them for later use by the simulation segment. The simulation segment is the actual simulation subroutine which is called by an instruction in the control program.

3.3.2 Example of a Simulation Routine

The organisation and structure of a simulation routine is best explained by the use of the ELINK subroutine as an example. The ELINK routine models an entry street by generating vehicles randomly with a user specified probability and stepping the vehicles progressively down the street. A more detailed description of this routine from the traffic point of view is given in 5.4. The routine will be discussed by referring to page and line number in the listing given. The listing has been divided into 5 main groups - comments and assembler pseudo operations, routine header, program segment, data setment and the actual simulation subroutine.

Page 1, lines 1 to 18 consists mainly of comments on the routine which are ignored by the assembler. The only mandatory statements are the declaration of all the pseudo instructions as .FXTRN (lines 12 and 13). Line 17 defines an indirect address which is placed in page zero. This indirect address is used later for the control program which is generated by the program segment.

Lines 20 to 29 comprise the header. The header provides the interface between the routine and the FASP executive and a strict format must be adhered to for correct operation. The first word of

the header contains the address of the routine which interprets the run time data statement (see page 2, line 1). If no data routine is required the pseudo instruction "NONE" must be placed here. The second and fourth word (lines 21 and 23) contain the number of storage blocks that must be reserved for the simulation routine. In this case sufficient storage has been provided for $64(100_8)$ entry links in the network. The third and fifth words (lines 22 and 24) define the number of constant and initialisable storage locations to be reserved in each storage block (see 2.3.4). In this case a total of 320 constant and 1600 initialisable locations are reserved for the ELINK subroutine.

The sixth word (line 25) contains the address of the last instruction plus one of the simulation routine. This is used by FASP as a pointer to the next simulation routine. The pseudo operation in line 26 is an instruction to the assembler to pack the ASCII bytes of subsequent text data from the left to right in the computer memory words.

The seventh word of the header (line 27) contains the first two characters of the routine name. The routine name is the name used in network definition and data statements to identify the particular simulation routine. There is no restriction on the length of the name.

The program segment immediately follows the end of the name (line 31). When a network definition statement is entered with the routine name "ELINK", execution is transferred to the program segment. Before execution is transferred, a GETAD pseudo instruction is issued by FASP and the accumulators are loaded with the relevant data (see 3.2.1). In the example, the instructions in lines 32 and 33 store the addresses of the constant and initializable data areas for the referenced routine and routine number in locations EL2 and EL3

respectively. This in effect comprises the interpretation of the data statement into a section of control programs. The pseudo instruction PUTP writes the subroutine call (lines 36 to 38) into the control program area. The word following PUTP is the address of the last location which is to be written in the control program area.

The run time data segment is listed on page 2. When a data statement is entered from the console, a GETAD pseudo instruction is executed by FASP and execution continues from line 3. Line 3 stores the address of the constant storage block for later use. Loop constants are set up in lines 4 and 5. The GET pseudo instruction gets the first parameter (in this case a flow rate) in the data statement from the input buffer and converts the decimal value to binary. Lines 7 - 16 manipulate the data into a form most suitable for the simulation routine (see 5.4.2) and stores it in the buffer. The loop count is incremented and the pseudo instruction GETBR examines the break character in the definition statement. If the break character is a carriage return, the remaining lanes' flow rate constants are zeroed and the pseudo instruction RETURN in line 25 transfers execution back to FASP.

If the break character was not a carriage return, a check is made to see if more than 5 parameters are specified (lines 31 - 34). If not the above process is repeated for all the parameters specified in the data statement. If more than 5 parameters are specified, the ERROR pseudo instruction (line 35-36) is executed and ERROR 20 is printed.

The actual simulation segment is given on page 2, line 44 to line 22, page 3. When the simulation program is run and the section of the control program generated by the program segment is executed, a jump to subroutine ELINK is made using the indirect address

```
---
PAGE   1
07/04/75
 1                              .TITL   ELINK
 2
 3                      ; ENTRY LINK OF UP TO 5 LANES
 4                      ; CALLING SEQUENCE:
 5
 6                      ; PROGRAM:-
 7                      ;    ELINK,NO.
 8
 9                      ; DATA:-
10                      ;      ELINK,NO.,FLOW RATE 1,FLOWRATE 2,.....FLOW RATE 5
11
12                              .EXTN   PUTP,GET,RETURN,GETEP,MULTIPLY
13                              .EXTN   DIVIDE,ERROR
14                              .HEAD           *ELINK ROUTINE*
15
16                              .ZREL
17  00000-000064' .ELINK: ELINK
18
19                              .NREL
20  00000'000020'               EL1             ; ADD OF DATA ROUTINE
21  00001'000100                100             ; 64 ELINKS
22  00002'000005                5               ; CONST STORAGE REQUIRED
23  00003'000100                100
24  00004'000031                25.             ; INITIAL STORAGE REQUIRED
25  00005'000131'               NEXTE           ; POINTER TO NEXT ROUTINE
26          000001              .TXTM   1       ; LEFT TO RIGHT PACKING
27  00006'042514                .TXT    *ELINK* ; NAME OF ROUTINE
28  00007'044516
29  00010'045400
30
31                      ;       PROGRAM AREA
32  00011'040405                STA     0,EL2   ; CONST ADD
33  00012'044403                STA     1,EL3   ; INITIAL ADD
34  00013'177777                PUTP            ; WRITE PROG
35  00014'000017'               EL3
36  00015'005000-               JSR     @.ELINK
37  00016'000001    EL2:        .BLK    1
38  00017'000001    EL3:        .BLK    1
39
```

```
---
PAGE  2                          *ELINK ROUTINE*
07/04/75
 1                                ; DATA ROUTINE
 2
 3  00020'040775  EL1:    STA     0,EL2
 4  00021'102400          SUB     0,0
 5  00022'040775          STA     0,EL3    ; COUNTER
 6  00023'177777          GET              ; GET FLOW RATE
 7  00024'105000          MOV     0,1      ; SET UP ACS FOR MULTIPLY
 8  00025'102400          SUB     0,0
 9  00026'030434          LDA     2,C10K   ; 2^12
10  00027'177777          MULTIPLY
11  00030'030431          LDA     2,C100
12  00031'177777          DIVIDE
13  00032'030754          LDA     2,EL2
14  00033'024764          LDA     1,EL3    ; DISPLACEMENT
15  00034'133000          ADD     1,2
16  00035'041000          STA     0,0,2    ; STORE FLOW RATE CONSTANT
17  00036'010751          ISZ     EL3      ; BUMP DISPLACEMENT
18  00037'177777          GETBR            ; GET BREAK CODE
19  00040'174014          COM#    3,3,SZR  ; TEST FOR CR (-1)
20  00041'000412          JMP     EL6      ; NO, CONTINUE
21  00042'024755          LDA     1,EL3    ; ZERO REMAINING LANES
22  00043'020420          LDA     0,C5
23  00044'176400          SUB     3,3
24  00045'122415  EL5:    SUB#    1,0,SNR  ; FINISHED ?
25  00046'177777          RETURN           ; YES
26  00047'151400          INC     2,2
27  00050'125400          INC     1,1
28  00051'055000          STA     3,0,2
29  00052'000773          JMP     EL5
30
31  00053'020410  EL6:    LDA     0,C5     ; CHECK FOR MORE THAN 5 PARMS
32  00054'024743          LDA     1,EL3
33  00055'122414          SUB#    1,0,SZR
34  00056'000745          JMP     EL1+3    ; NO
35  00057'177777          ERROR
36  00060'000024          20.              ; MORE THAN 5 LANES
37
38  00061'025000  C100:   10000
39  00062'010000  C10K.   10000
40  00063'000005  C5:     5
41
42                        ;       ACTUAL USER SUBROUTINE
43
44  00064'054711  ELINK:: STA     3,EL2
45  00065'102400          SUB     0,0
46  00066'031400          LDA     2,0,3    ; GET CONSTANT ADD
47  00067'113000          ADD     0,2      ; ADD IN DISPLACEMENT
48  00070'025000          LDA     1,0,2    ; GET FLOW RATE
49  00071'125005          MOV     1,1,SNR  ; IS IT ZERO ?
50  00072'000411          JMP     EL7      ; YES , NEXT LANE
51  00073'071,70          DIBP    2,70     ; GET RANDUM NO
52  00074'146020          ADDZ    2,1      ; 1 TO CARRY IF VEHICLE GENERATED
53  00075'126560          SUBCL   1,1
54  00076'031401          LDA     2,1,3    ; GET INITIAL ADD
55  00077'115000          MOV     0,3
56  00100'177120          ADDZL   3,3      ; *4
57  00101'173000          ADD     3,2
58  00102'125200          MOVR    1,1
```

```
---
PAGE   3
07/04/75
 1   00103' 025005
 2   00104' 125100
 3   00105' 045005
 4   00106' 025006
 5   00107' 125100
 6   00110' 045006
 7   00111' 025007
 8   00112' 125100
 9   00113' 045007
10   00114' 025010
11   00115' 125100
12   00116' 045010
13   00117' 125500
14   00120' 034676
15   00121' 031401
16   00122' 113000
17   00123' 045000
18   00124' 101400   EL7:
19   00125' 031716
20   00126' 112415
21   00127' 001401
22   00130' 000736
23
24          000131'
25
26
```

```
*ELINK ROUTINE*

        LDA     1,5,2   ; GET 1ST LENGTH OF LANE
        MOVL    1,1     ; SHIFT
        STA     1,5,2   ; STORE
        LDA     1,6,2   ; NEXT LENGTH
        MOVL    1,1
        STA     1,6,2
        LDA     1,7,2
        MOVL    1,1
        STA     1,7,2
        LDA     1,10,2
        MOVL    1,1
        STA     1,10,2
        SUBCL   1,1     ; SAVE CARRY
        LDA     3,EL2
        LDA     2,1,3   ; INITIAL ADD
        ADD     0,2
        STA     1,0,2   ; STORE OUTPUT
        INC     0,0     ; BUMP COUNTER
        LDA     2,C5
        SUB#    0,2,SNR ; FINISHED ?
        JMP     2,3     ; YES RETURN
        JMP     ELINK+2 ; CONTINUE

NEXTE=.

.END
```

.ELINK (page 1, line 17). The address of the constant and initializable data storage areas are loaded by the instructions in line 46 and 48. Vehicles are randomly generated and shifted one position down the street (see 5.4). On completion a return to the control program is made by the instructions on page 3 line 21 and the next simulation subroutine is then called in a similar fashion.

3.4 Interrupt Service Routines

Users may write routines to service special devices that they may have in their system. The master interrupt routine in FASP identifies all interrupts and transfers control to the service routine On entry to the service routine interrupts are on but devices have been masked out according to the user supplied mask which is inclusively ORed with the current mask. Thus higher priority devices may interrupt while a device is being serviced. The user service routines may use all accumulations without saving them but must not use any pseudo functions unless TRAP has been used (see 3.2.3).

The layout of service routines is as follows:

WORD	FUNCTION
1	Device code
2	Mask to be used while servicing device
3	Pointer to next routine
4	Start of service routine.

The service routine may be exited from by either a JMP Ø,3 if AC3 has not been destroyed or by a JMP @ INTR. If the second return is used, INTR must be declared as an external displacement (.EXTD).

3.5 Memory Organization of FASP

The memory map of a FASP system is given in Fig. 3.1. FASP uses very little of the first 256_{10} words of memory (page zero) and 175_{10} words are available for simulation routine ZREL. FASP is divided into 2 programs - FASP and FASPEND. The first is the main program of FASP and the second is the initialization area, disk support and routines to interpret the keywords INITIAL, DYNAMIC and TERMINAL (see 2.4.2).

The core area required for interrupt service routines and simulation routines (A) is variable depends on the memory requirements for the routines used when the system is generated. The remaining area for data storage and the control program (B) may be calculated from expression

$$B = M - (A + 4201)$$

where M is the size of memory used and all numbers are decimal.

3.6 Conclusion

The format for simulation and interrupt service routines has been described and the procedure for generating a FASP system with the routines is given in Appendix 4.

The use of the FASP executive eases the programming effort in writing a simulator. This can be seen by referring to the simulation routines described in Chapter 5. The simulation routines for the complete traffic simulator required less than 1500_{10} words of memory.

It is hoped that the FASP executive will be used in a large number of different areas of simulation and will not be confined to road traffic studies.

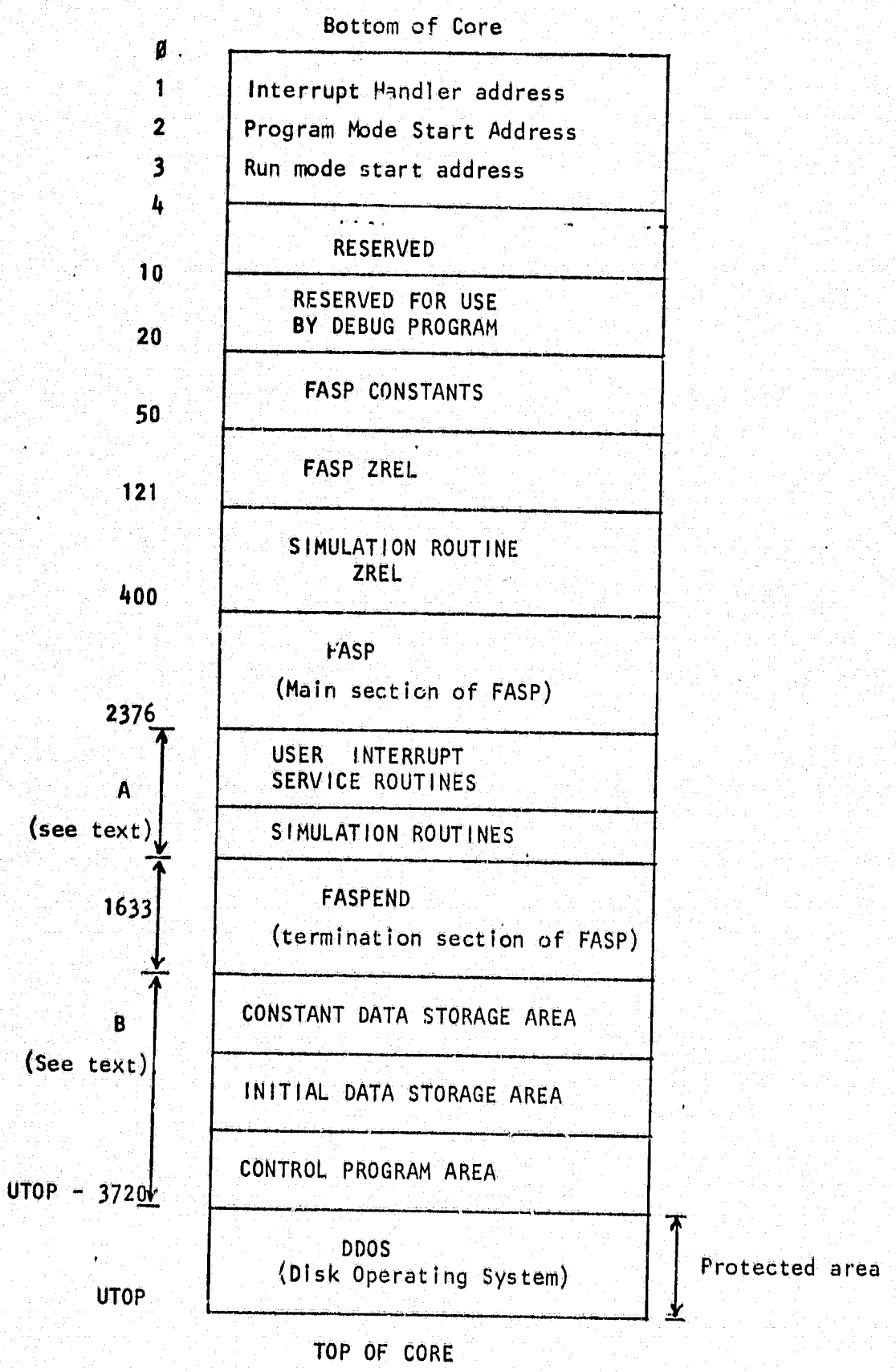

Figure 3.1. Memory Map of FASP

CHAPTER 4

DATA SOURCES FOR A TRAFFIC SIMULATOR

4.1 Introduction

All simulations of stocastic systems rely heavily on the use of random numbers for the data input to the model. In the particular case of a macro traffic model, the close statistical correspondence between the random data sources and the system being modelled largely determines the overall accuracy and validity of the model. Many techniques exist for generating random numbers in both hardware and software. Software random number generators are available as standard subroutines for many computers and are widely used in simulators [19]. These routines are usually sufficiently accurate for traffic simulators but are very slow and considerably degrade the speed of the simulation. Hardware data sources were used in the UMIST [10,11,12] traffic simulator and resulted in high speed data generation (see 4.5.2). These were dedicated generators and separate generators were required for generating different frequency distributions. This method is inflexible and expensive in terms of hardware cost.

Many different frequency distributions are encountered in the description of traffic phenomena, some of which can be described mathematically, others are totally arbitrary. Thus specialised, dedicated random number generators are unsuitable and a highly flexible random number generator is required.

Two related special types of data are encountered so often in

traffic work that an additional, far simpler data source is justified. This first case is the Poisson distribution which is used for modelling vehicles at the periphery of a network. The second case is the division or allocation of vehicles to lanes in specified proportions. It will be shown that a geometric distribution which is very easily generated in hardware can be used for both these cases.

4.2 Requirements on a Traffic Source

Data sources used for traffic simulations should have the following properties:

(i) Accurate representation of the required distribution

The validity of a simulation is largely influenced by the correspondence between the data source and the real life situation.

(ii) Fast Operation

A large amount of data is required for simulating a traffic network and the speed of the simulation model will be related to the speed of the data sources (see Chapter 7).

(iii) Good Statistical Properties

The output of the data source must pass the usual tests for randomness (frequency test, auto and serial correlation test) as any correlation or deviations from the required distribution will effect the reliability of the simulation.

(iv) Repeatable

The data source should be repeatable so that results of simulations using different network parameters can be compared under identical input conditions. This implies the use of a pseudo random number generator which can be preset to some arbitrary initial conditions.

(v) Easily Programmed

The distribution and parameters of the distribution should be programmable by some means so that they may be easily changed.

4.3 A Flexible Pseudo Random Number Generator with Arbitrary Frequency Distribution

4.3.1 Introduction

Various methods for generating random numbers with arbitrary frequency distributions are described in the literature [21,22,23]. The majority of these methods are software orientated but a few specialised cases have been realised in hardware [24,25,26]. In general these techniques involve the manipulation of a set of uniformly distributed random numbers to produce the required distribution. The most common methods use simple decision making processes and a lookup table.

One approach [27] uses a large lookup table. A uniform random number in the range 0-n (where n is the length of the table) is generated and is used as an index to address a number in the table. The addressed number becomes the output number. The relative frequency of each number in the set of output numbers is proportional to the number of times that number appears in the table. This method is fast as it only requires one decision process to generate a number. The smallest relative frequency of a number ..ving the system is inversely proportional to the size of the table so this system is rarely used when high accuracy is required.

A second more common approach is a modification of the above and requires less storage capacity. A uniform random number is compared with a set of breakpoints stored in a table. The output number is dependent on the range in which the random number lies.

For example, consider a set of random numbers (0,9) and a set of output numbers in the range (0,3). If the required relative frequencies of the output numbers 0, 1, 2, 3 are 0,2, 0,4, 0,1, 0,3 respectively, the breakpoints would then be 1, 5, 6, 9. Let n be the random number generated. If $0 \leq n \leq 1$ the output number would be 0. If $1 < n \leq 5$ the output number would be 1 and so on. Thus for a set of 2^N output numbers, only 2^N-1 entries are required in the table. This method is slow as N decisions on average are required to generate each number.

An alternative method proposed by Walker [26], requires 2^N entries in the table and requires only one decision to generate a number. A hardware generator using this technique was used in the present work. The generator was designed and constructed by Walker and is described below.

4.3.2 Basic Description of the Data Source

The method used manipulates numbers produced by a uniform random number generator to produce the final output number in one decision step involving only as much storage as number of output numbers. In this case the set of uniform random numbers has the same range as the set of output numbers. The method is most easily explained by the use of an example. Consider the case of a set of output numbers (0, 1, 2, 3) with a corresponding s of relative frequencies (0,1, 0,25, 0,3, 0,35). The set of uniform random numbers is (0, 1, 2, 3) and the relative frequency of each number is 0,25. Clearly, the number of occurrences of the number zero must be decreased, the number one must emerge unchanged, and the occurrences of the remaining numbers must be increased. Since the sums of the relative frequencies of the input and output sets are equal to unity, a system which could convert a fraction of the numbers entering the system as zeros to a 2 or a 3

would produce the required distribution. In the example, $\frac{10}{25}$ of the zeros must remain unchanged, $\frac{5}{25}$ must be changed to a 2 and $\frac{10}{25}$ must be changed to a 3. In order to reduce the number of decision levels required, in practice, $\frac{15}{25}$ of the zeros entering the system will emerge as a 2 and $\frac{10}{25}$ of the twos entering the system emerge as a 3. Thus only one level of decision is required. This decision is performed by comparing a second random number (independent of the input number) with a fixed number which is associated with the input number. The fixed number is proportional to the probability of the input number leaving the system unchanged. If the second random number is less than the fixed number, the input number emerges unchanged. Otherwise the alternate number which is associated with the input number becomes the output number. Thus the storage table consists of 2 numbers per number in the output set. A diagram of the method is given in Fig. 4.1.

The smallest increment in relative frequency that can be obtained with this method is proportional to the precision of the fixed number but can be made extremely small by using floating point representation for the fixed number.

4.3.3 The Hardware Data Source

Although the algorithm described above is inherently fast, software implementation was not used as the speed is dependent on the time taken to generate two uniform random numbers. In the NOVA computer the time to generate a uniform random number is 230 μs or the equivalent of approximately 100 instructions [19]. In hardware this this operation requires approximately 50 ns. The algorithm was therefore implemented in hardware.

In a hardware implementation the most costly single item is the memory for storing the constants and a separate memory is required for

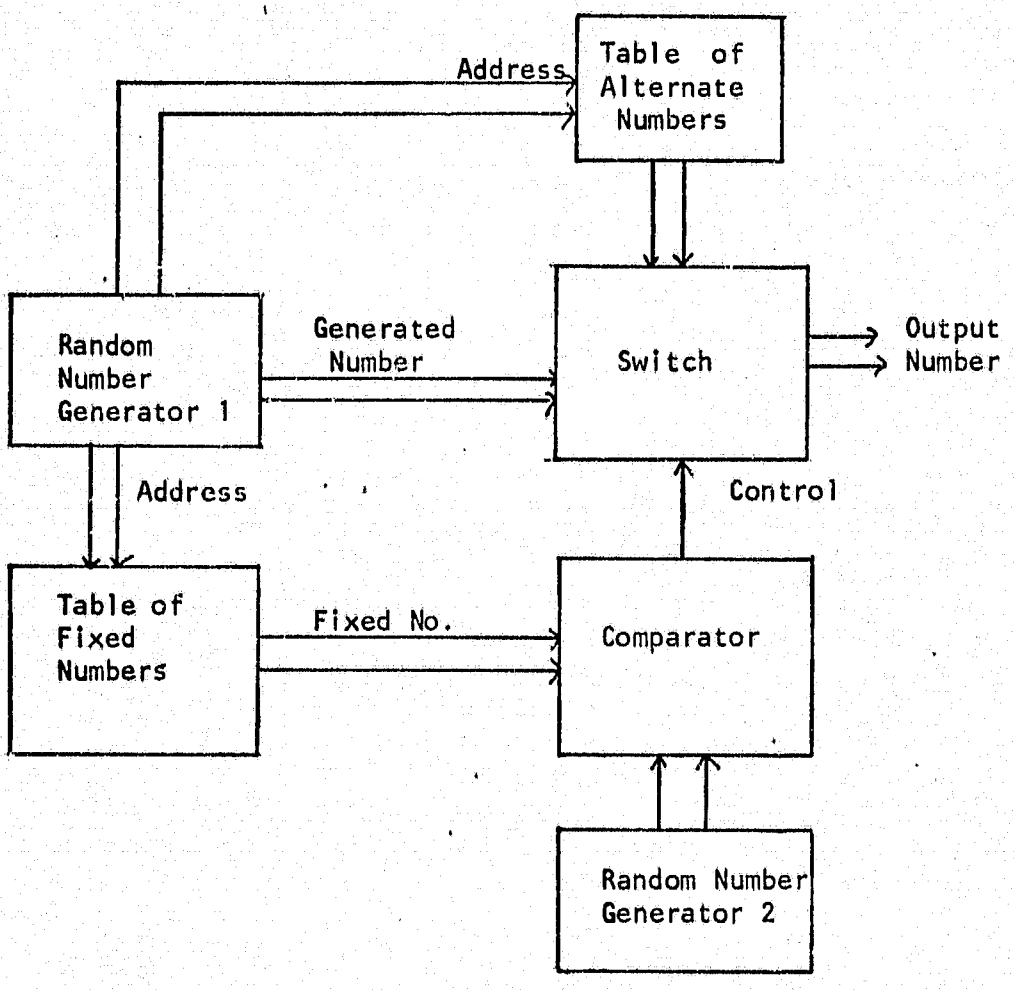

Figure 4.1. Basic System for Generating Arbitrary Frequency Distribution.

every distribution. For traffic simulation a large number of different distributions may be required so a large amount of memory would be needed. This can however be avoided in the case of a hybrid traffic simulator by using the computer memory for storing the various sets of constants. The generator is then connected to the direct memory access (DMA) channel of the computer which enables very high speed transfer of the constants between the computer memory and the peripheral generator memory (Fig. 4.2).

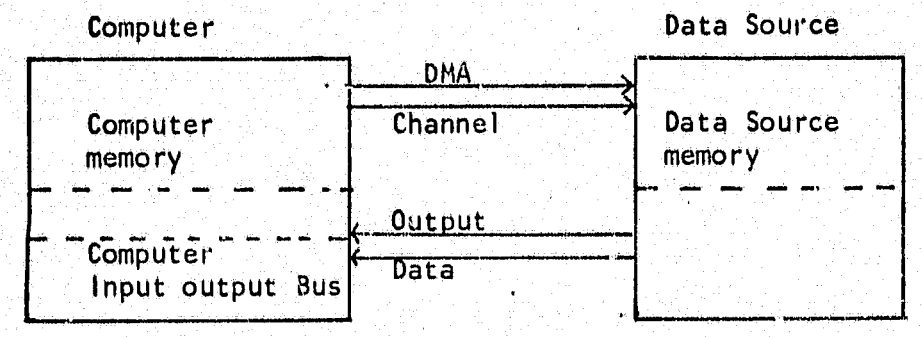

Figure 4.2. Computer-Data Source Connections

This technique results in a far less costly implementation and the number of different distributions that can be generated is only limited by the amount of available computer memory. The loss in speed due to the time required to transfer the constants from computer to periferestal is negligible.

4.3.4 Specifications of the Data Source

(a)	Number of bits in output word:	Variable 1 - 8
(b)	Number of computer instructions required to generate a new number:	2
(c)	Amount of computer memory required per distribution:	2^N words (N = No. of bits in results)
(d)	Smallest increment in relative frequency:	10^{-5}

4.3.5 Statistical Tests on the Generator

Detailed statistical tests for randomness were performed on a prototype of this generator and are detailed in the literature [18]. These tests included frequency tests and serial and auto and correlation tests. This generator used fixed point representation for the fixed numbers and was not as accurate as the generator used in the present work which was also tested in a similar manner. The results obtained from these tests were within expected statistical limits.

4.3.6 Conclusions

The generator described above is fast, flexible and has good statistical properties. It is easily programmed using an algorithm by Walker [26] and the results are repeatable as hardware random number generaors are used. Thus the generator satisfies all the requirements for a traffic data source.

4.4 A Method for Obtaining a Distribution with Arbitrary Mean and Standard Deviation from a Base Distribution

4.4.1 Introduction

From the description of the data source it is clear that a unique set of constants is required for generating each distribution. In simulation work normally only a few basic types of distributions (e.g. Normal, Poisson, Binomial) are required. These basic types of distribution are used repeatedly. Thus several distributions of a single type (e.g. normal distribution) may be required with different means and standard deviations. If a separate set of constants is used for each mean and standard deviation, a large amount of memory space would be required. An alternative approach is to use a single set of constants for each type of distribution only and mathematically manipulate the resultant random numbers to produce a

new set of numbers with a specified standard deviation and mean. The term "base distribution" is used here to denote the distribution produced by the set of constants. The mean and standard deviation of the base distribution is chosen to accurately define the required type of distribution, and the manipulations described below are used to produce the required mean and standard deviation.

4.4.2 Description of the Method used for Changing the Mean and Standard Deviation of a Distribution

The method used depends on the following manipulation of the expressions for the mean and standard deviation of a discrete distribution.

The mean of a discrete distribution is given by

$$\mu = \frac{1}{n} \sum_{i=1}^{n} a_i \qquad (1)$$

when μ = mean of the distribution

n = No. of samples taken

a_i = Value of sample i.

Adding a constant b to each side of equation (1) gives

$$\mu + b = \frac{1}{n} \sum_{i=1}^{n} (a_i + b)$$

Multiplying equation (1) by a constant c gives

$$c\mu = \frac{1}{n} \sum_{i=1}^{n} ca_i \qquad (1b)$$

The standard deviation of a discrete distribution is given by

$$\sigma = \frac{1}{n} \sum_{i=1}^{n} (a_i - \mu)^2 \qquad (2)$$

where σ is the standard deviation

n is the number of sample

a_i is the value of sample i

μ is the mean of the distribution

From (1a) and (2) it can be seen that the standard deviation is not changed if a constant b is added to each sample a_i.

Multiplying equation (2) by a constant c gives

$$c\sigma = \frac{1}{n} \sum_{i=1}^{n} (ca_i - c\mu)^2 \qquad (2b)$$

From the above results, it is clear the mean and standard deviation of a base distribution can be changed by single arithmetic manipulations of each sample in the distribution. The constants are calculated from equations (3) and (4) below

$$c = \frac{\sigma_1}{\sigma} \qquad (3)$$

$$b = \mu_1 - c\mu$$

$$= \mu_1 - \frac{\sigma_1}{\sigma} \qquad (4)$$

where c is the multiplying constant

b is the additive constant

μ is the base distribution mean

σ is the base distribution standard deviation

μ_1 is the required mean

σ_1 is the required standard deviation

4.4.3 Conclusion

In the present simulator a set of constants is calculated for each type of distribution using the algorithm of Walker [26]. These constants provide the definition of the base distributions and are loaded into the computer memory as part of the simulation program. By suitably programming the simulation routines the simulator user can be provided with a facility for choosing the type of distribution and specify the mean and standard deviation at run time.

4.5 A Specialised Traffic Data Source

4.5.1 Introduction

One of the most important requirements of a traffic simulator is a traffic data source which can accurately model the arrival of vehicles at the periphery of a network. Adams [28] has shown that the number of vehicles arriving at an isolated intersection in a given time follows a Poissonian distribution. A binary representation can be used with a "1" corresponding to a vehicle arrival. Thus any data source giving a binary sequence of ones and zeros of the required distribution and probability can be used.

The Poissonian distribution is given by [29]:

$$\phi(x) = \frac{e^{-\alpha T}(\alpha T)^x}{x!} \qquad (5)$$

where $\phi(x)$ = the probability of x occurrances of an event in time T

α = probability of an even occurring

= average number of events per unit time.

Equation (5) can be rewritten in terms of time as:

$$\psi(t) = \begin{cases} \alpha e^{-\alpha t} & t \geq 0 \\ 0 & t < 0 \end{cases} \qquad (6)$$

where $\psi(t)$ is the probability that the time between two successive events will be t.

If the number of zeros between two successive ones in a binary sequence is defined as the gap length (G) equation (6) can be rewritten as

$$\lambda(G=i) = \begin{cases} \alpha e^{-\alpha i} & i \geq 0 \\ 0 & i < 0 \end{cases} \qquad (7)$$

The above negative exponential distribution is difficult to generate but it can be approximated by an easily generated geometric distribution of the form:

$$\Theta(G=i) = \alpha(1-\alpha)^{i} \qquad (8)$$

if α is small.

This is shown by the comparative graphs (Figs. 4.3 and 4.4) of equations 7 and 8 for different values of α and in the subsequent tests of the generator (See 4.5.3(b)).

The basic epoch of the simulator was chosen to be 1/3 seconds which corresponds to 5,56 m at 60 k.p.h. This is approximately the length of an average car so good positional accuracy can be achieved with the simulator. This choice also ensures that for flow rates below saturation flow (1300 vehicles per hour) the probability of a vehicle arriving is sufficiently small for the geometric distribution to be a good approximation to the negative exponential.

The length of the epoch is the same as that used by Green [11] for a hardware simulator. Micro model simulations [1,4] usually use 1 second as the basic epoch time as the modelling is more accurate and the effective simulation speed is increased by using a longer time between model updates.

4.5.2 Basic System Design

Hardware generators with a geometric frequency distribution were designed by Redshaw [30] and Robinson [31] for use with the UMIST simulator. Two maximal length chaincode generators were cross coupled

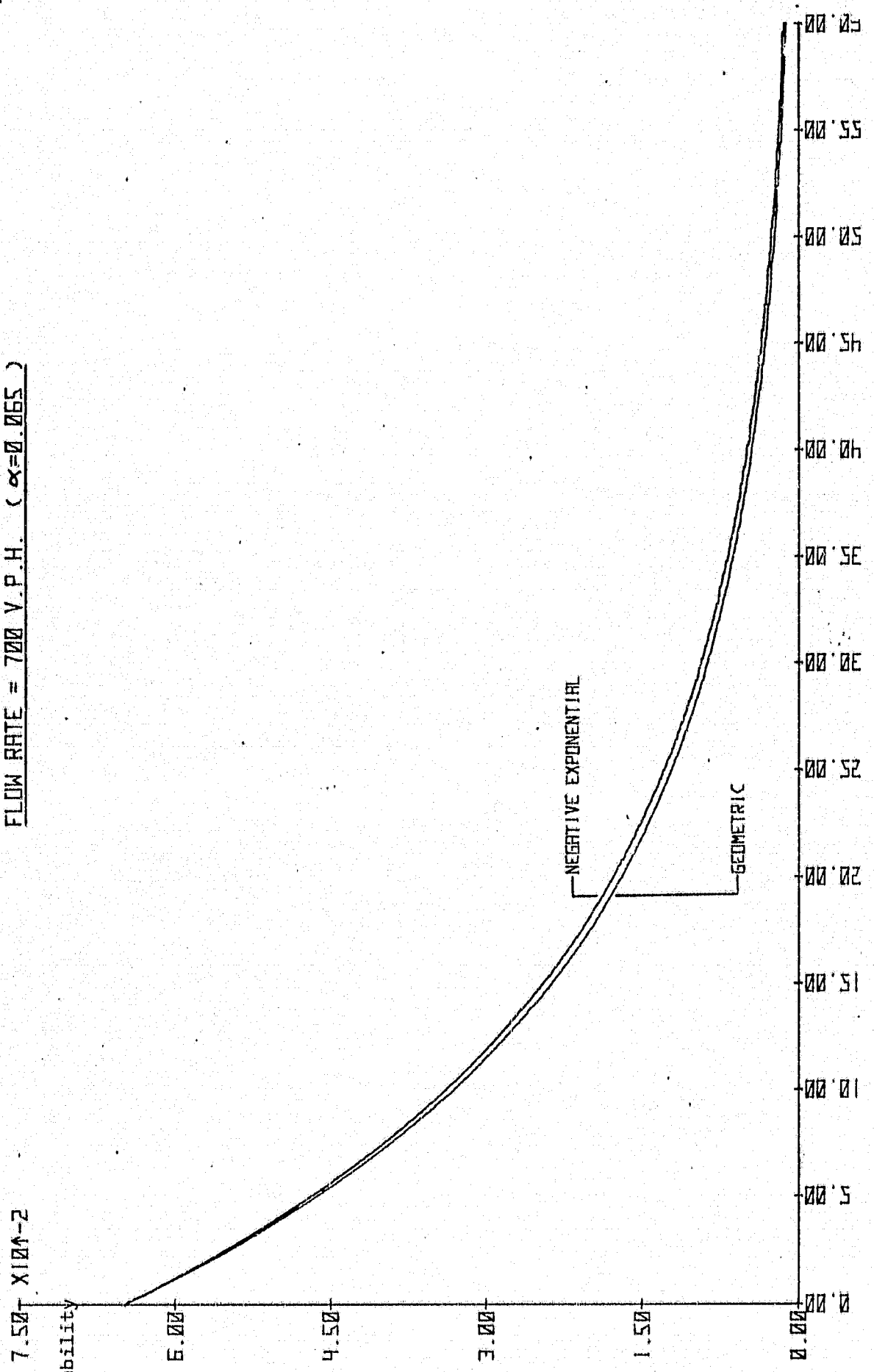

FIG. 4.3 COMPARISON BETWEEN GEOMETRIC AND EXPONENTIAL DISTRIBUTIONS

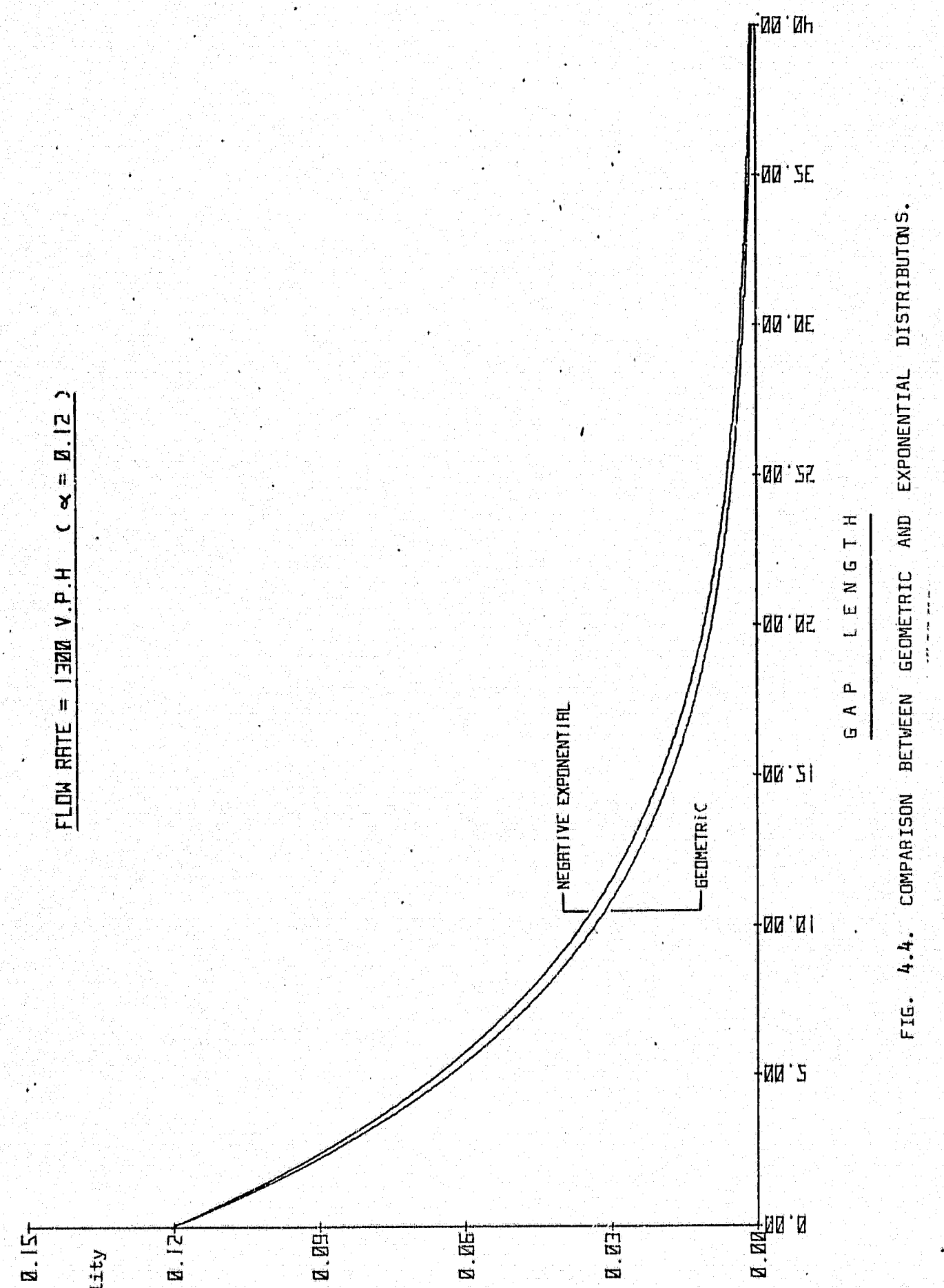

FIG. 4.4. COMPARISON BETWEEN GEOMETRIC AND EXPONENTIAL DISTRIBUTONS.

with half adders to produce several independant outputs. The probability of a 1 occurring is 0,5 for each of these outputs. If N of these outputs are connected to an AND gate the probability of a 1 occurring at the output is $0,5^N$. Thus a coarse control of probability could be obtained. Further fine control was provided by deriving a number with F bits from the chaincode generators and comparing this number with a manually programmed fixed number f. If the generated number is less than or equal to the fixed number a 1 is produced at the output of the comparator. The outputs of the first AND gate and the output of the comparator are then fed into another AND gate to obtain the final output. The probability of a 1 at the output is $(0,5)^N \left(\frac{f+1}{2^F}\right)$

The basic system is illustrated in Fig. 4.5. This system cannot be programmed easily because of the relatively complicated formula for calculating the probability. Although several independent outputs can be obtained from the two chaincode generators, care must be exercised in the selection of the pickoff points to avoid correlation between the outputs [32].

An alternative approach is shown in Fig. 4.6. An N bit uniformly distributed binary number is generated and impared with the programmable fixed number F. If the random number is less than the fixed number a 1 is produced at the output. The probability of a 1 is $2^{\frac{F}{N}}$. This system is much simpler than the first method and more easily programmed. By choosing a large value for N, an almost stepless control of probability is possible. This method can be used for software generators but the time to generate the uniform random number is of the order of 230 µs.

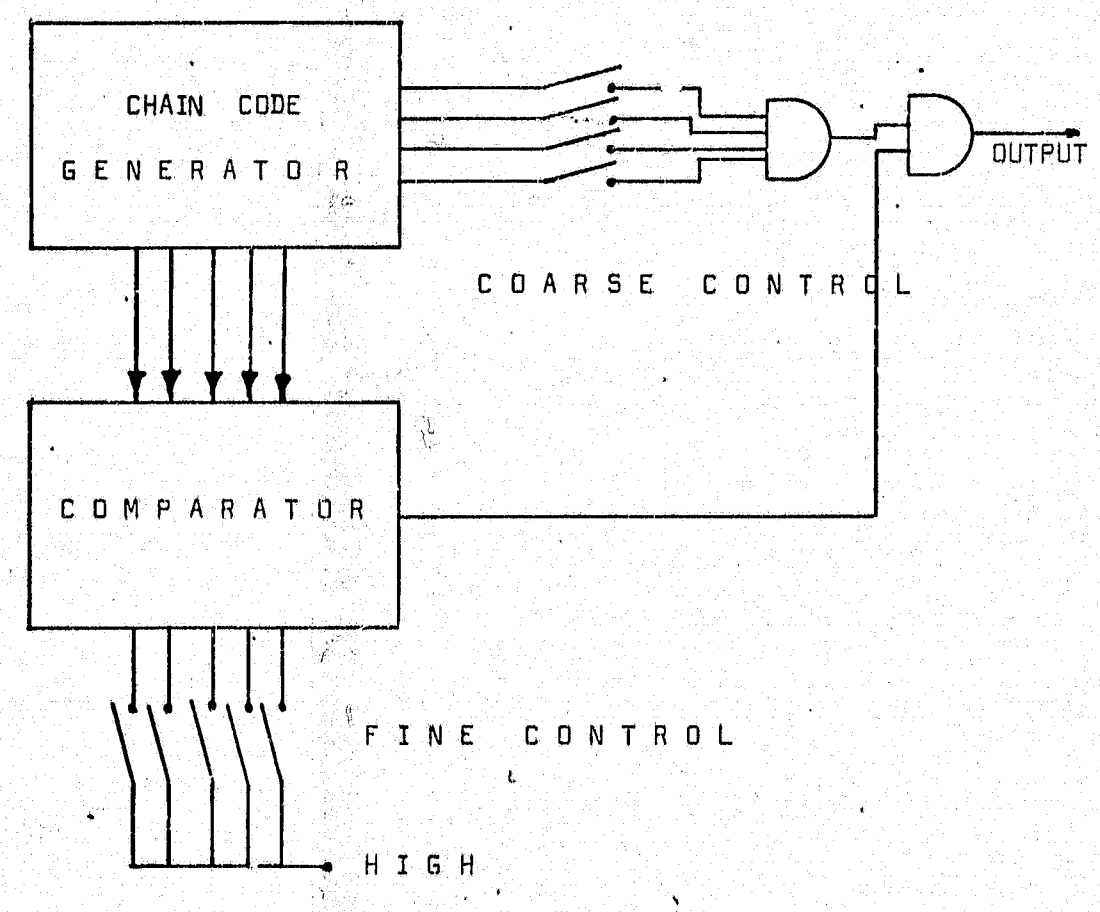

FIG. 4.5 BASIC BLOCK DIAGRAM OF REDSHAW GENERATOR

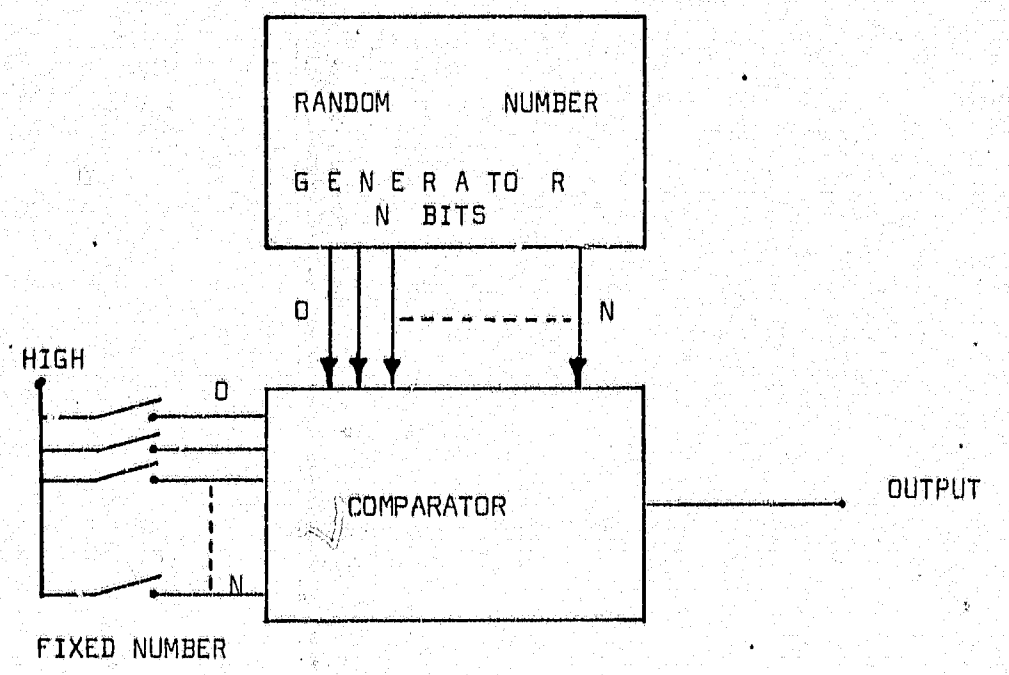

FIG. 4.6 BASIC BLOCK DIAGRAM OF SYSTEM USED

The system used for the hardware generator is shown in Fig. 4.7. Sixteen independent channels are provided, but only one random number generator and comparator is used. A twelve bit random number generator was chosen giving the smallest increment in flow rate of 2,64 vehicles per hour (V.P.H.). Ten bits would have given sufficient accuracy (10,5 V.P.H.), but the additional accuracy could be obtained at little extra cost as the integrated circuit packages used for the memory and comparator each contained four bits.

Sixteen fixed numbers of 12 bits each specify the flow rate in each channel and are programmed into a memory. On applying a clock pulse at the input, the internal clock sends out 16 pulses to the random number generator, address counter and output shift register. The address counter addresses a new word in the memory every time it is clocked. Thus 16 random numbers and 16 fixed numbers are applied sequentially to the two inputs of the comparator. The output of the comparator is fed into a shift register which stores the 16 channel result.

The system described above is suitable for simulating single intersections or for use with a hardware simulator. For multiple intersections it complicates the software design and is not particularly fast as the memories have to be reprogrammed for each intersection. Thus the full facilities of the generator were not used. The method adopted was to read the uniform random number from the generator and perform the comparison in the computer resulting in only a minimal loss in speed (approximately 4 μs). The design of the random number generator is given below and the remaining design is given in Appendix 5.

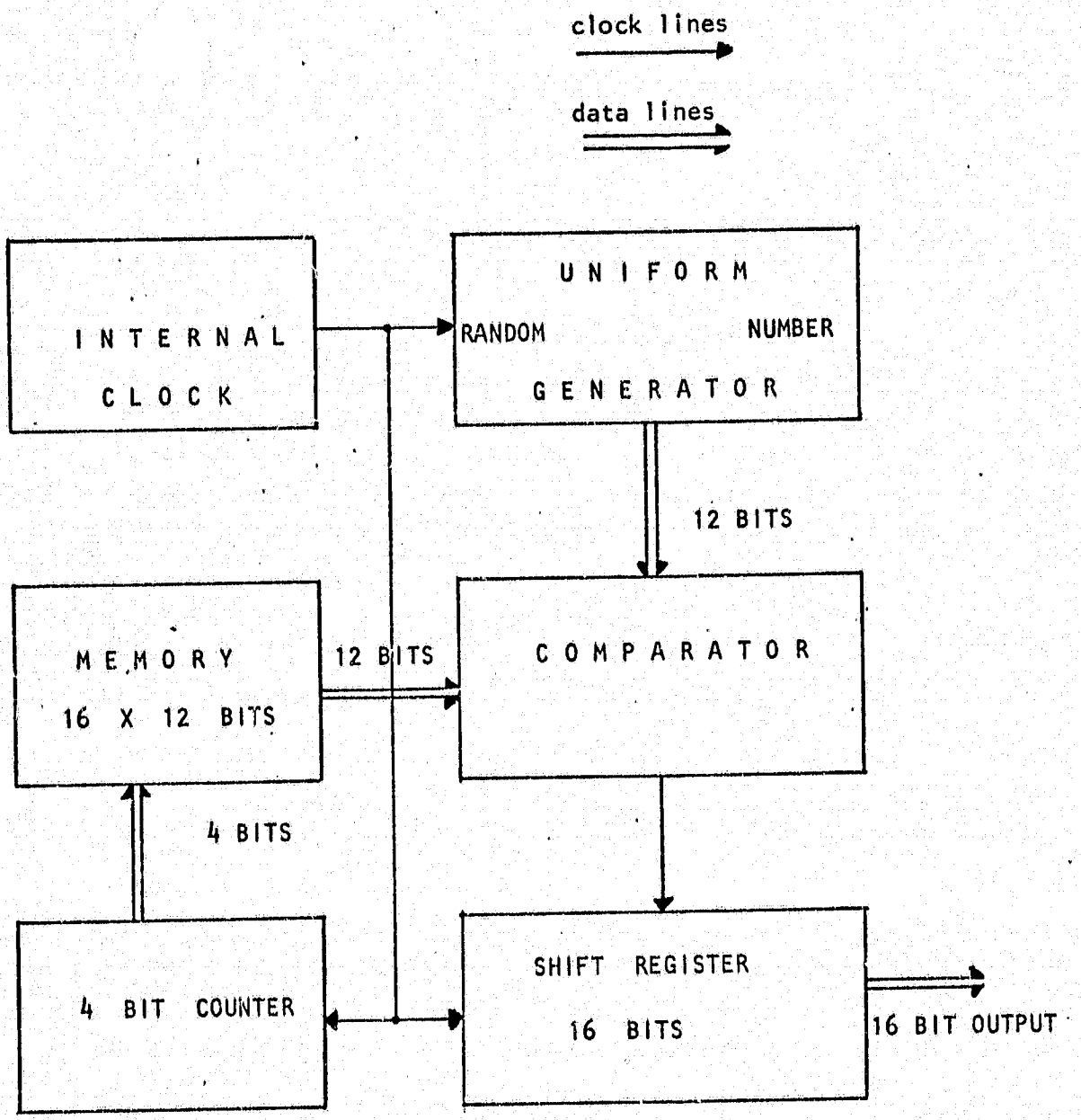

Figure 4.7. System used for the Generator

4.5.3 Design of the Random Number Generator

A method widely used for generating random numbers is to connect a shift register with feedback. The generalised circuit is shown in Fig. 4.8. If the logic network used is a modulo 2 adder (otherwise known as an EXCLUSIVE OR gate) a linear chain code is produced (Fig. 4.9). A binary chain code is "a sequence of 2^n or fewer bits arranged such that a pattern of n adjacent digits locates the position of those digits uniquely". [42]

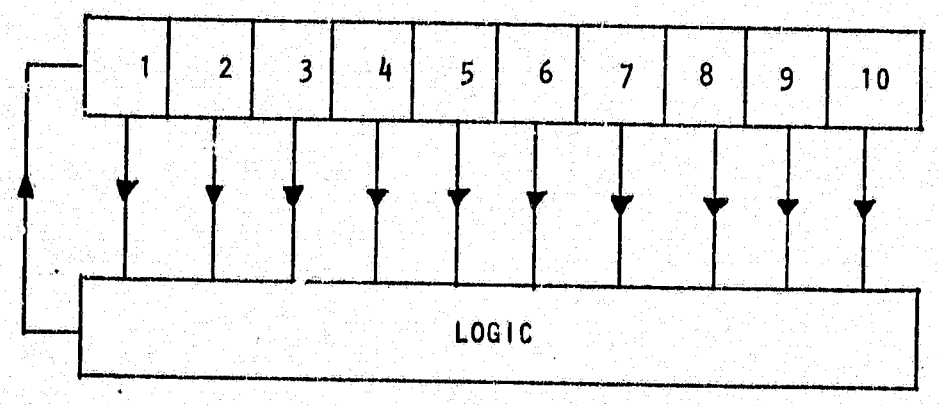

Figure 4.8. Generalised Feedback Shift Register

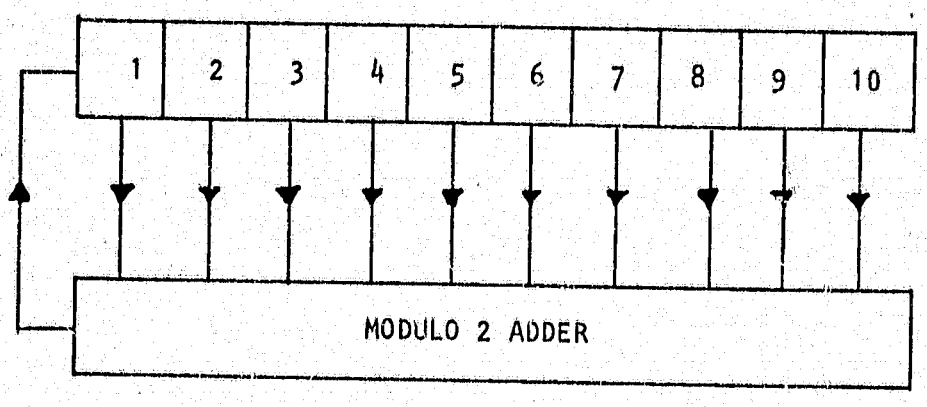

Figure 4.9. Linear Feedback Shift Register

By suitable connection of the modulo 2 adders a prime chain code of maximal length is produced. The number of ways of choosing n binary digits is 2^n, however the pattern containing all zeros cannot occur as it is self perpetuating. Thus a maximal length chain code has $2^n - 1$ states and a total of 2^{n-1} ones and $2^{n-1} - 1$ zeros. For a

large value of n the frequency of occurrence of ones and zeros can be considered equal. It would appear that an n bit (or longer) feedback shift register would be suitable for use as an n bit random number generator as each number in the set $(1,2^n)$ occurs only once. Pratt [33] has shown however that there is large serial correlation between successive numbers. This correlation can be avoided if the shift register is clocked n or more times so that all the bits of the previous number are removed and a completely new number is contained in the register. If the number of clock periods between two generated numbers and the sequence length $(2^n - 1)$ are relatively prime the sequence length is unchanged by this technique. This method was chosen in preference to the two cross coupled shift registers used by Walker [26] as it is simple to design, implement and has good statistical properties without having to select the pickoff points.

The integer n was chosen to be 23 which is prime and hence a maximal length sequence (m-sequence) would be produced. The sequence length is approximately 8 million which in the case of a single intersection would be equivalent to about 32 days of simulated traffic before the sequence repeats itself. The number of clock pulses between numbers was chosen empirically to be 24 as this configuration has less serial correlation (see 4.6.2b) than 23 pulses did.

Clocking the register N times between successive numbers is equivalent to jumping N states in the state diagram. Thus it is not difficult to arrange the feedback in such a manner so as to produce this effect with a single clock pulse.

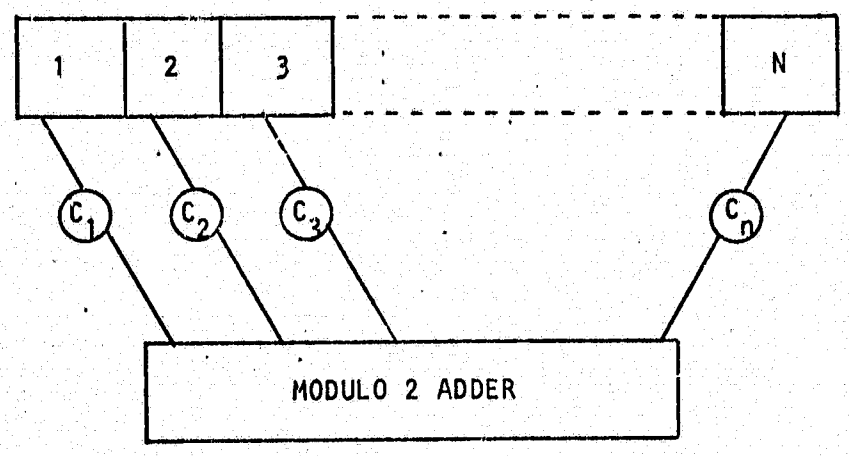

Figure 4.10. Generalised Linear Feedback Shift Register

For any feedback shift register there exists a characteristic polynomial.. For the shift register in Fig. 4.10 it can be written as

$$f(x) = 1 + C_1 x + C_2 x^2 + \ldots\ldots C_n x^n \qquad (9)$$

where the weighting factor $C_i = 0$ or 1,

and x^i is the ith bit in the shift register.

For a m sequence to be produced it is necessary that f(x) be irreducible. Reference to the table of irreducible polynomials modulo 2 by Watson [34] gives the following polynomial for n = 23.

$$f(x) = 1 + x^5 + x^{23} \qquad (10)$$

The complementary polynomial

$$f(x) = 1 + x^{19} + x^{23} \qquad (11)$$

is also prime and it allows a faster hardware implementation of the generator. Using Polynomial 7 there are 2 levels of modulo 2 adders instead of 6 if polynomial 6 was used. Rigorous methods have been formulated for the design of a feedback shift register which shifts states [35], but an intuative approch is far simpler. Representing the state of bit 1 after 24 clock pulses by 1', it is clear that the

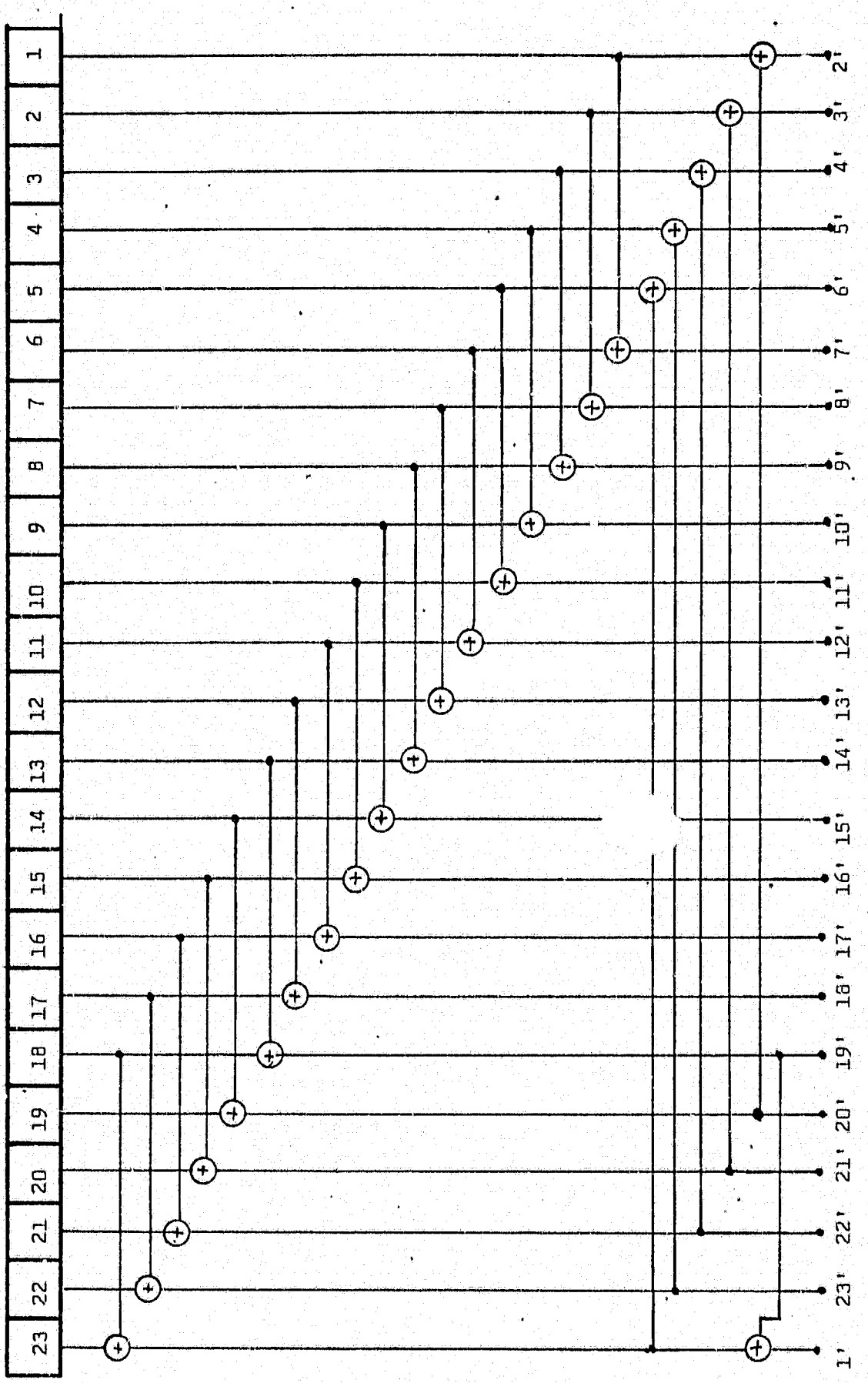

FIG. 4.11 CONNECTIONS OF THE SHIFT REGISTER FOR THE RANDOM NUMBER GENERATOR

new value 23' of bit 23 is obtained by modulo 2 addition of bit 22 and bit 18. Similarly 21' = 21 ⊕ 17 etc. This process is most easily performed by drawing a diagram as in Fig. 4.11 which shows the connections used.

A requirement of the generator was that it could be reset to some initial condition. A starting point of all 1's was chosen. This could be achieved by shifting a '1' serially into the register but 23 clock pulses are required. By using 23 cascaded flip-flops with clear inputs the resetting could be achieved by a single pulse. This requires that the $\bar{Q}$ outputs of the flip-flops be used and the data entering the flip-flops must be inverted. The data is inverted by using EXCLUSIVE NOR gates. These gates are not standard but the function can be implemented by using EXCLUSIVE OR gates and inverting one of the inputs. The method used is illustrated in Fig. 4.12.

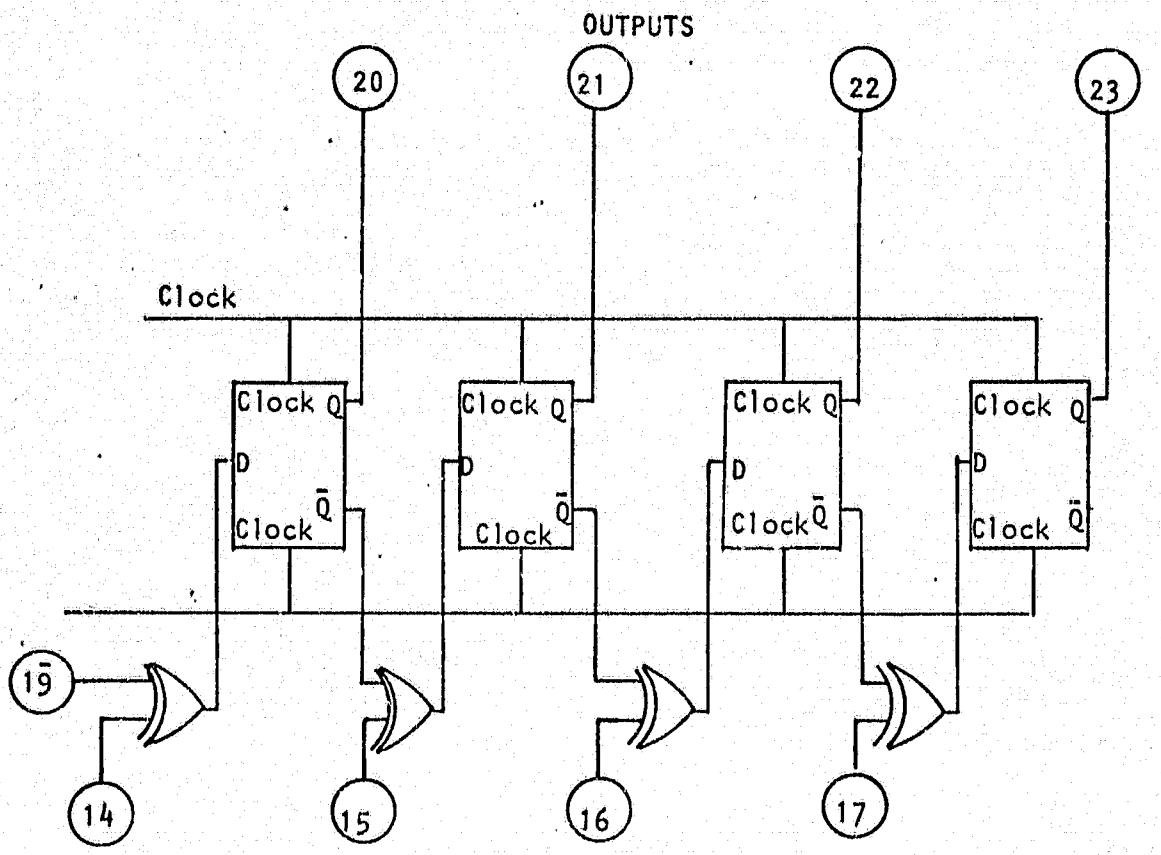

Figure 4.12. Design of the Feedback Shift Register.

4.6 Statistical Tests on the Specialised Traffic Generator

4.6.1 Introduction

The accuracy and trustworthiness of any simulation is only as good as the data source used, an extensive testing of the data source is necessary. The data source must closely model the situation being studied without introducing characteristics of its own. A large number of tests have been used to test the statistical properties of random numbers, e.g. frequency, runs, bunching, Gruenberger d^2, gap, serial autocorrelation and crosscorrelation tests. Walker [26] pointed out that it is not necessary for a data source to satisfy all these tests, but should rather pass those tests which indicate that it is suitable for the required application. The tests considered important for a traffic data source are frequency test, gap test, and serial, auto and cross correlation tests.

The uniform random number generator was first tested as the properties of the individual output channels are largely determined by its performance. These tests were not as extensive as those on the final output as they were only required to indicate the suitability of the first generator. The tests on the final output were performed using the hardware comparison and were therefore limited to 16 channels. However, since the uniform random number generator was shown to have good statistical properties, and a long sequence length, it is clear that an extension of the system to more channels will not affect the statistical properties of the individual channels.

4.6.2 Statistical Tests on the Random Number Generator (RDI)

(a) Frequency Test

The frequency test compares the number of occurrences of each number or group of numbers (class) in a set with the theoretical number of occurrences. The Chi-square "goodness of fit" test is

used for the comparison. The Chi-square value is computed with the following expression:

$$X^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \qquad (12)$$

where n = number of frequency classes

O_i = observed frequency of class i

E_i = expected frequency of class i

It is impossible to test the 12 bit number directly as there would be 4096 frequency classes. The approach used was to test 4 adjacent bits at a time. One thousand consecutive blocks of 320 samples were tested and their X^2 values computed. The number of samples in each block was chosen so as to produce on average 20 occurrences in each class as suggested by Kendall [36]. Choosing 5% levels of significance should result in 100 rejections of the null hypothesis, i.e. 50 rejections below $X^2_{.95}$ and 50 above $X^2_{.05}$. The values of $X^2_{.95}$ and $X^2_{.05}$ can be found in table 2 of Paradine and Rivett [37]. In this case for 15 degrees of freedom $X^2_{.95} = 7{,}261$ and $X^2_{.05} = 24{,}996$.

The generator was tested with a computer program by Walker [26] and a summary of the results is given in Table A6.1. The test was considered to be satisfactory.

(b) Serial Correlation Test

The serial correlation tallies the occurrence of each number following a given number. If numbers are statistically independent the probability of the number N_1 occurring after the given number N has occurred is simply, the probability of the number N_1 occurring, $P(N_1)$. Thus the expected distribution is the same as that for the frequency test. Again the 12 bit word was divided into three words

of four bits. Four hundred consecutive blocks of 320 samples were tested. The limits of the X^2 value are the same as in 3.2.1 and 40 rejections of the null hypothesis are expected. The results in Tables A6.2 show that on average this is true, and the test was considered to be satisfactory.

(c) Autocorrelation Test

The normalised autocorrelation coefficient $C(\tau)$ is given by

$$C(\tau) = \frac{\sum_{n=1}^{v} (A_n - \bar{a}).(A_{n+\tau} - \bar{a})}{\sum_{n=1}^{v} (A_n - \bar{a})^2} \qquad (13)$$

where v = the number of samples used

A_n = the nth number in the sequences

$\bar{a}$ = the mean value of the sequence

τ = the delay between the reference and the delayed sequences.

The value of $C(\tau)$ is the unity for zero delay and should be close to zero for all other delays if the sequential numbers are statistically independent. The correlation coefficients should also be normally distributed about a mean value of zero. To test this hypothesis, 100 coefficients are computed and are divided into 5 equal frequency classes giving on average 20 occurrences in each class. The mean and standard deviation of the coefficients is calculated and are used to calculate the expected frequency distribution, thus there are only 2 degrees of freedom in the Chi-square test.

The 12 bit word was not tested directly as double precision multiplication would have been required. The word was therefore split into the four most significant bits and the remaining eight and the two portions of the word were tested separately.

The coefficients were computed for 20 consecutive blocks of 320

numbers using a program by Walker [26]. Using 5% levels of significance 2 rejections of the null hypothesis are expected. The results of the test are shown in Tables A6.3 - A6.4 are considered to be satisfactory.

4.6.3 Statistical Tests on the Geometrical Distribution

(a) Frequency Test

Comprehensive frequency tests were performed, each channel being tested at different flow rates varying from 100 to 1300 V.P.H. in steps of 100 V.P.H. In each case 100 consecutive blocks were tested. The number of samples used is computed each time so that the expected number of occurrences in the two classes is at least 20. Ten rejections of the null hypothesis are expected in each case as 5% levels of significance were used. The number of upper bound failures are on average, as expected but the number of lower bound failures is nearly twice the expected value. As there are only two frequency classes and the expected frequency is an integer an examination of equation 12 shows that there is a high probability of the Chi-square result being zero, giving a lower bound failure. The results obtained are shown in Tables A6.5 and A6.6 and the test was considered satisfactory. The computer program used is shown in Appendix 7.

(b) Gap Test

The gap test is used to determine whether the lengths of the gaps (i.e. number of zeros) between arrivals (ones) conform to the required distribution. It was shown in 4.5.1 that the required distribution of the gaps is a negative exponential.

The probability of a gap of length i is given by:

$$\lambda(G = i) = e^{-\alpha i}$$

The probability of occurrences of gaps in the range (0,b) is:

$$\lambda(0<G<b) = \sum_{i=0}^{b} \alpha e^{-\alpha i} \qquad (14)$$

$$= \alpha(1 + e^{-\alpha} + e^{-2\alpha} + \ldots + e^{-b\alpha})$$

$$\lambda(0<G<b) = \left(\frac{1 - e^{-\alpha(b+1)}}{1 - e^{-\alpha}}\right) \qquad (15)$$

To test the generator the expected distribution was divided into 10 frequency classes each having equal probability. Chi-square testing was used to compare the actual distribution with the theoretical distribution.

As in the frequency test each channel was tested at different flow rates. In all cases 40 consecutive blocks of 200 gaps (i.e. 201 arrivals) were used giving an average of 20 occurrences in each class. Using 5% significance levels, 4 rejections of the null hypothesis are anticipated. Reference to the results given in Tables A6.7 and A6.8 shows that the distribution conforms to the required distribution for flow rates up to 1100 V.P.H. This was expected as the approximation discussed in section 4.5.1 does not hold for high values of probabilities. The test was considered satisfactory. The program used for this test is shown in Appendix 8.

(c) Serial Correlation Test

Serial correlation was tested under the same conditions as for the frequency test, i.e. 100 consecutive blocks and the number of samples adjusted to give at least 20 occurrences in each class. The serial correlation was tested for numbers following a one and for numbers following a zero. The results are shown in Tables A6.9 to A6.12 Again, for numbers following a one the results are satisfactory. In the case of numbers following a zero more upper bound failures occurred

than anticipated. This could be due to the fact that a shorter length of the sequence than in the first case is tested. The program used is shown in Appendix 7.

(d) Autocorrelation Test

The method used in 4.6.7(c) for testing cannot be applied to a binary sequence of ones and zeros as the coefficients produced by equation 13 are not normally distributed about the mean. Also the results are difficult to interpret and it is impossible to say whether a large coefficient occurred by chance or because of significant correlation.

The equation used here is of the form:

$$C(\tau) = \sum_{n=1}^{V} (A_n).(A_{n+\tau}) \qquad (16)$$

The product $(A_n.A_n + \tau)$ can only take the values 0 or 1. If the probability that $A_n = 1$ is p, it can be shown that the probability that the product term $(A_n.A_{n+\tau}) = 1$ is p^2. Denoting this product term by X_n, equation (16) can be written:

$$C(\tau) = \sum_{n=1}^{V} X_n \qquad (17)$$

where $\phi(X_n = 1) = p$ and $\phi(X_n = 0) = (1 - p) = q$

Equation (17) is the sum of independent Bernoulli random variates and the sum random variable $C(\tau)$ has the binomial probability distribution: [38]

$$\phi|C(\tau) = K| = \binom{V}{K} p^K q^{V-K}$$

$$= \frac{V!}{K!\ (v - K)!}\ p^K q^{V-K} \qquad (18)$$

From the aforementioned equation the values of the lower and upper limits of K can be calculated so that on average, 10% of the

coefficients will be less than the lower limit and 10% will exceed the upper limit. In addition the expected mean and standard deviation of the coefficients can be calculated by the following formulae [37]:

$$\text{mean} = Vp$$

$$\text{Standard deviation} = Vpq$$

The above test was implemented in a computer program shown in Appendix 9. A large number of samples were taken (V = 10 000) as in the case of low flow rates the probability is extremely small. Coefficients were evaluated for the first 100 delays. Each channel was tested with flow rates varying between 200 and 1300 V.P.H. in steps of 100 V.P.H. and the results are shown in Appendix 10. The results which were considered failures in terms of too many coefficients in the upper ten percentile are underlined. Examination of these results show that the standard deviation in these cases is less than the expected value and there are far fewer coefficients in the lower 10 percentile than expected. The mean in these cases is higher than expected. This seems to indicate that the actual distribution is just shifted from the expected positions by a small amount. This would be the case if the actual probability was higher than expected. An examination of the largest coefficient confirm that there is no significant correlation in these cases.

(e) Crosscorrelation Test

In any system where several channels are derived from a single source, it is important that no correlation exists between the different channels. The method of testing is similar to the method used for autocorrelation except that 2 sequences are used.

The equation used is:

$$C(\tau) = \sum_{n=1}^{V} a_n \cdot b_{n+\tau} \qquad (19)$$

Where a_n is the value of the nth element of sequence and $b_{n+\tau}$ is the value of the $n+\tau$ th element of sequence b.

The probability p used in the expressions for calculating the expected distribution is the combined probability $p_a p_b$ where p_a and p_b are the probability that $a_n=1$ and $b_n=1$ respectively. Every combination of channels was tested for flow rates varying between 200 and 1200 V.P.H. in steps of 200 V.P.H. and examples of the results are shown in Appendix 3. The large number of lower 10% failures in the case of very low flow rates is due to the small number of occurrences of 1's. The comments given in the discussion of the results for the autocorrelation test apply in this case as well and the various channels are considered to be sufficiently statistically independent.

4.7 Conclusion

Two hardware data sources have been described. It has been shown that both sources satisfy the requirements for traffic modelling. The use of combined hardware and software techniques is considered to provide the best combination of flexibility, high speed and low cost.

The data sources are used by the simulation routines described in Chapter 5.

CHAPTER 5

SUBROUTINES FOR SIMULATING TRAFFIC

5.1 Introduction

This chapter describes simulation routines, which, used with FASP, form the basis of a traffic simulator. The routines are designed to simulate a road network commonly encountered in the centre of large cities and not arterial road networks.

The road network is considered to consist of links and intersections. A link is a road which connects one intersection to another. The intersections are four leg, light controlled intersections. In the present work, fixed time control was used but provision has been made for the routine to be modified for dynamic control policies. A method of entering vehicles into the network is required and a third element has been provided. This element can be considered to be a source of vehicles feeding into a link. The three elements, streets, intersections and entry streets have been named LINK, TLIGHT and ELINK respectively.

The technique of dividing the network into elements and some of the modelling of the elemnts is an adaptation of the work done at UMIST [8,9,10,11,12,13]. The major differences being in the lumping of several elements together to form a composite element. For example, the intersection element consists of queues, signal timing and vehicle extraction control.

The criteria in designing these simulation routines were: ease of network definition; ease of run time data entry; high speed; flexibility

and economical utilisation of computer memory. Ease of definition of the network and data entry is simplified by the use of composite elements and is also one of the facilities provided by the FASP executive. High speed is obtained by using efficient assembler language programming and macro modelling. High run time speed is also achieved by using the facility provided by FASP to preprocess data, when it is entered, into the form most convenient the actual simulation section of the routine.

Two output routines are provided which print the queue lengths at intersections. One of these routines is designed for interactive simulation and the other writes the information onto disk for later formatting and printing.

The routines are economical in the use of memory and a network consisting of 40 intersections with 64 entry links, 40 links and 40 output routines of both type can be simulated by a computer with 16K words of memory.

The average real time to simulation time ratio is approximately 200 : 1 per intersection (see Chapter 6).

5.2 General Simulation Routine Layout

A consistent structure of the routines has been adopted. Each routine consists of three subsec[illegible] The first subsection interprets the network definition st[illegible] into the required control program sequence.

The second section preprocesses the run time data and stores it in a form convenient to the simulation section.

The simulation section is the last section and is a subroutine which is called by the control program generated by the first subsection.

The generated control program is of the following general form (see Chapter 3).

```
JUMP TO SIMULATION SUBROUTINE 1
 PARAMETER    1
 PARAMETER    2
     |
     |
     |
 PARAMETER    N
```

The simulation subroutine returns to the statements following PARAMETER N (which would be the start of the next subroutine call sequence).

The actual control program sequence generated by each simulation subroutine can be found by referring to the program listings in Appendix 12.

5.3 Link Simulation Routine

5.3.1 Introduction

The link routine models a street connecting one inte section to another. The link routine accepts vehicles which are discharged from an intersection or from another link or entry link. The vehicles are randomly assigned to a lane according to a user specified proportion. A random travelling time, sampled from a user defined distribution is assigned to each vehicle and the vehicle is placed in the appropriate position in the street. The positions of all the vehicles in the street are updated every epoch.

A link is unidirectional i.e. two links are required to simulate a two way road. Each link may have up to five independent lanes.

5.3.2 Modelling of Streets

The method of modelling streets is based on the shift register technique used by Green [11] in a hardware simulator. A vehicle is represented by a binary one in the shift register. Each epoch the register is shifted once. Thus in a single operation, the position

of all the vehicles in the street are updated. This approach was used in preference to the circular file technique used by Grigg and Hartley [8] in a software simulator as it is easily implemented in Assembly Language, fast and requires less storage.

The choice of shift register length is a compromise between minimising data storage requirements and representation of the longest street which may be simulated. A street length was chosen so that it would cater for all but the longest streets likely to be encountered in city network. If a longer street length is required, it may be simulated by joining several links together to simulate the required length.

The length of the shift register for each lane was chosen to be 64 bits. This gives a travelling time of 21 1/3 seconds, using the basic epoch of 1/3 sec. This corresponds to a length of 356 m at 60 kph.

The shift register is constructed using four computer words of 16 bits. The words are shifted individually by one position for every epoch simulated. Use is made of the carry bit facility of the NOVA computer to link the words together as shown in Fig. 5.1.

5.3.3 Allocation of Vehicles to Lanes

The LINK routine examines the output of the element feeding it every epoch. This output can be a single vehicle or a queue of vehicles in the case of an intersection element. Vehicles are removed from this queue and randomly allocated a lane. Associated with each lane i is a fixed number FN_i which is related to the proportion of vehicles to be assigned to that lane. A 12 bit uniformly distributed random number is read in from the hardware random number generator and is compared with the fixed numbers. The vehicle is then assigned to the highest numbered lane whose fixed number is

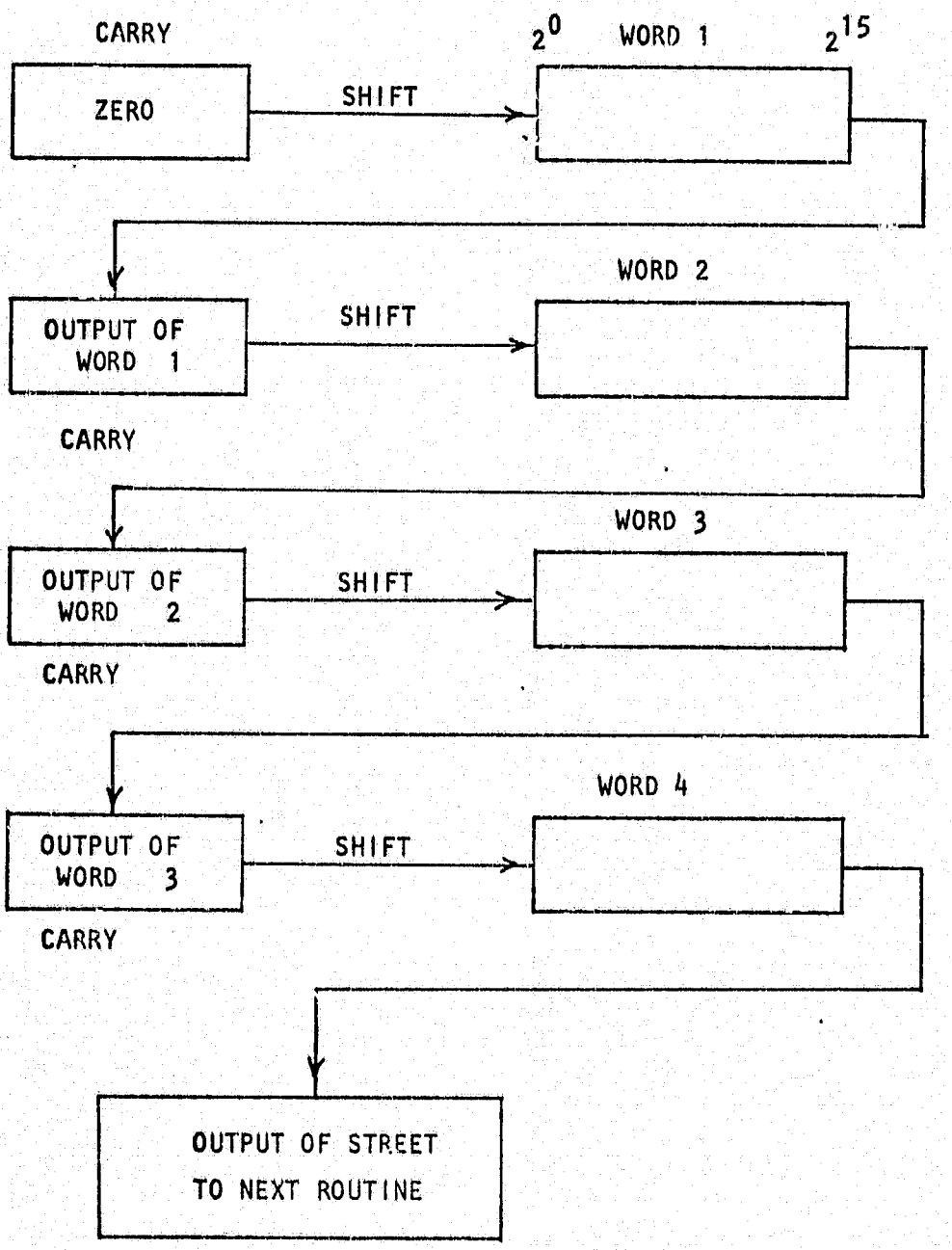

Figure 5.1. Modelling of Streets

greater than the random number. For example consider a uniformly d distributed random number in the range 1 - 100 and the percentage of vehicles to be assigned to lanes 1, 2 and 3 to be 20, 60 and 20 per cent respectively. The fixed numbers would then be 20, 80 and 100. Thus, if the generated number was less than 20 the vehicle would be assigned to lane 1. If the number was between 20 and 80 the vehicle would be assigned to lane 2 and so on.

For the case of a 12 bit binary random number, the fixed number for lane i is calculated using the following formula:

$$FN_i = \%FR_i \div 100 \times 2^{12} + FN_{i-1} \qquad (20)$$

where $\%FR_i$ is the % of vehicles to be assigned to lane i

and FN_{i-1} is the fixed number for the previous lane.

Note $FN_0 = 0$

The method of lane assignment effectively simulates lane changing, as a vehicle entering a LINK may exit via any of the lanes.

5.3.4 Allocation of Travelling Times

The hardware data source described in Chapter 4 is used to generate travelling times. Use is made of the results of 4.4.1 to enable the distribution, mean and standard deviation of the travelling times to be specified by the user. Any distribution that has been defined by a set of constants in the computer memory may be used. A 6 bit random number is used to give 64 possible travelling times. This choice is related to the length of the street. The vehicle is placed in a position in the street proportional to its travelling time. This is done by setting the appropriate bit in the appropriate word. If the position is already occupied, an attempt is made to place the vehicle in the preceeding position which corresponds to increasing its travelling time by one epoch. If all the preceeding positions are

occupied, the vehicle is placed in a "blocked queue" and an attempt to discharge the vehicle will be made during the next epoch. This statistically allows overtaking, caters for the forming of platoons within the street and takes saturation of the street into account.

The allocation of travelling times is fast due to the speed of the random number generator and a table lookup method which is used for setting the appropriate bit in the street words (see LINK Listing, Appendix 12).

5.3.5 Use of the LINK Routine

The network definition statement for the LINK routine is

LINK, NO1, INPUT ELEMENT, NO2, [LEG NO]

NO1 is the number identifying the particular link in the network. INPUT ELEMENT, NO2 identifies the element in the traffic network which feeds into the LINK. LEG NO. is an optional parameter which is used to identify which leg of an intersection is the input element.

This statement results in appropriate statements being written into the control program to call the LINK simulation subroutine when the simulation is run.

The run time data statement is of the following form :

LINK, NO, DIST. NAME, MEAN×100, STANDARD DEV×100, %FLOW LANE 1, %FLOW LANE 2 . . . . . % FLOW LANE 5)

DIST NAME is the name given to a simulation routine which consists of 64 constants required to generate a particular type of distribution (base distribution). Because integer representation is used throughout in FASP, the mean and standard deviations are multiplied by 100 when they are specified in the data statement. This means that they can be specified to an accuracy of 2 decimal places. The flow rate for each lane is specified as a percentage of the total flow i..

the link. If less than 5 lanes are used it is not necessary to specify the percentage flow in the unused lanes as the flow rates will automatically be set to zero.

The run time data statement is processed by a subroutine labelled L1 in the LINK simulation routine listing (Appendix 12). This subroutine calculates constants to be used to convert the mean and standard deviation of the base distribution to the specified values. The fixed numbers are also calculated and are stored to be used by the simulation segment.

5.3.6 Storage Requirements

Each LINK defined in the network requires 8 constant and 25 initial storage locations. Four additional storage words are required for the control program statements. The simulation routine is 192 words long.

Flow charts for the data statement conversion subroutine (L1) and the actual simulation segment (LINK) are given in Fig. 5.2 and Fig. 5.3 respectively.

5.4 Entry Link Simulation Routine

5.4.1 Introduction

The simulation routine for modelling an entry street has been named ELINK. The function of this routine is to generate vehicles at a specified flow rate and step them progressively down a street. At the end of the street the vehicles are available for processing by other simulation routines.

The ELINK routine provides for up to 5 lanes. Each lane is independent of the others. If less than 5 lanes are used the additional lanes are not processed so that the speed of simulation is a linear function of the number of lanes used.

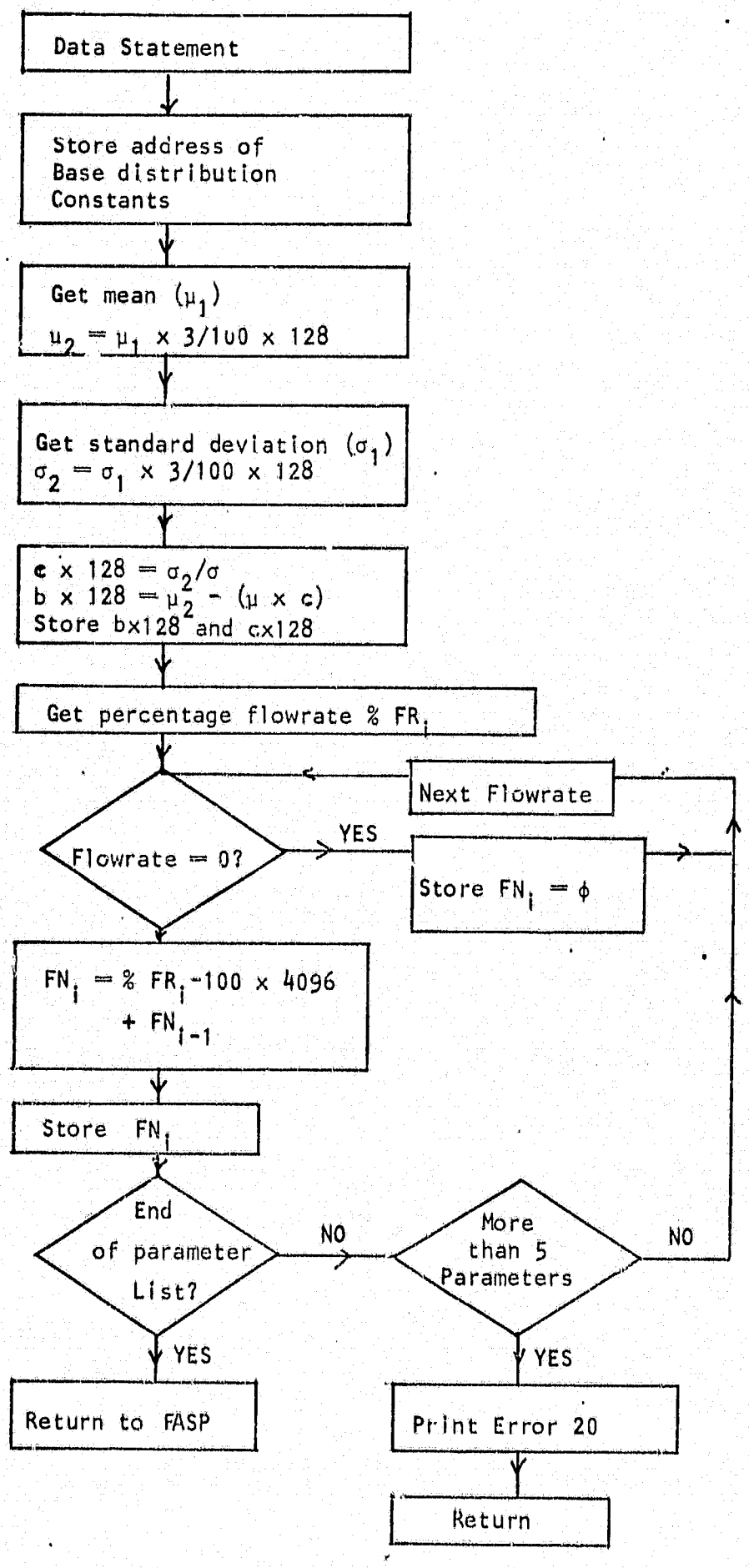

Figure 5.2. Flow Chart of LINK Data Routine L1

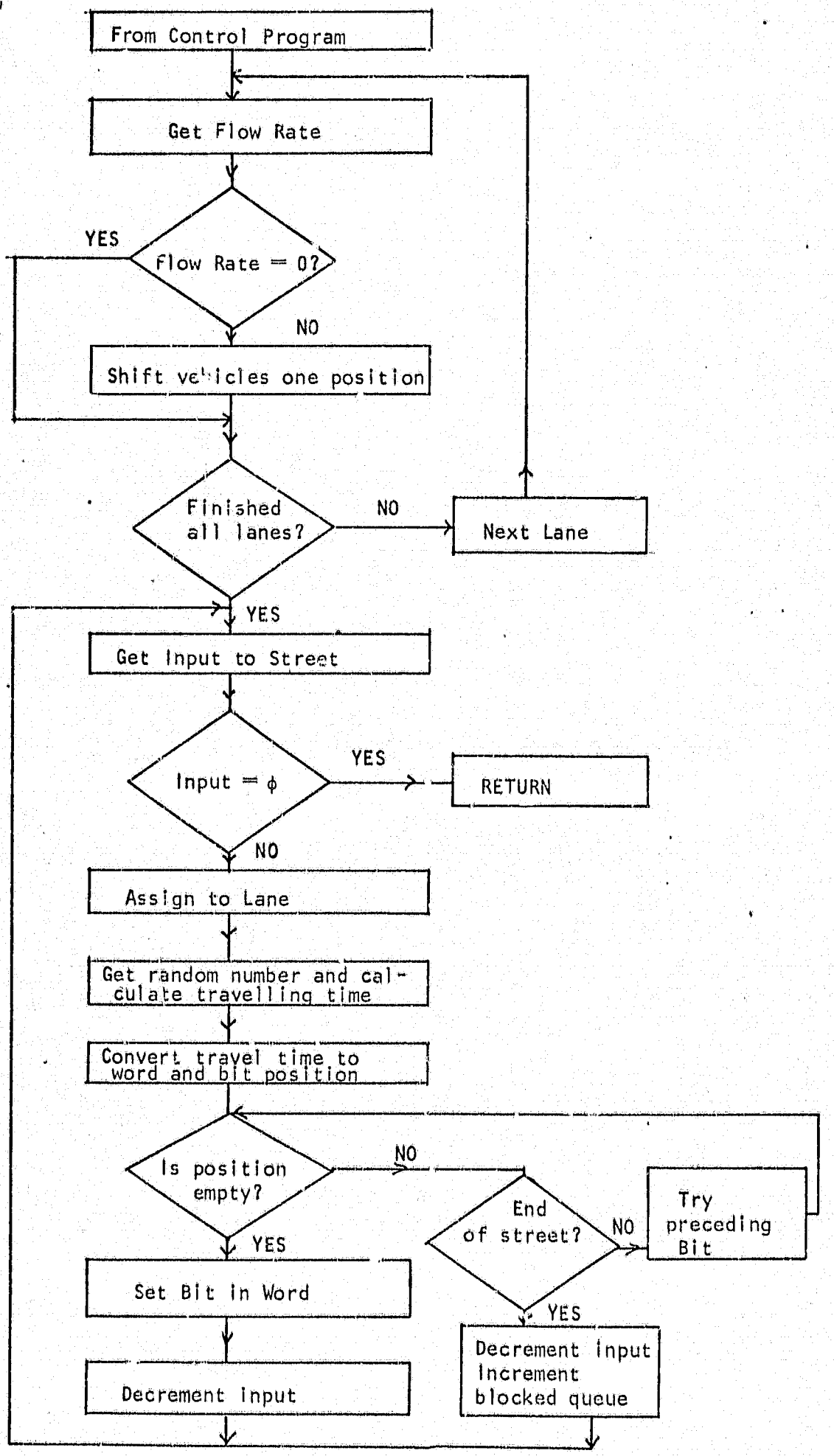

Figure 5.3. Flowchart of Link Simulation Subroutine

5.4.2 Vehicle Generation

Vehicles are generated with a Poissonian distribution. This is achieved by using the geometric approximation to the Poissonian distribution (see Chapter 4).

A 12 bit uniformly distributed random number read in from the hardware random number generator is compared with a fixed number. If the generated number is less than the fixed number, a vehicle is generated. The fixed numbers for each lane is calculated from the flow rate with the following formula:

$$FRC = FR \div (3600 \times 3) \times 2^{12} \qquad (21)$$

$$= FR - X\ 4096/10800$$

where FRC is the flow rate constant
FR is the flow rate in vehicles/hour.

5.4.3 Modelling of Streets

The street is modelled in the same manner as for the LINK routine. This means other simulation elements which are designed to interface with the LINK routine (e.g. TLIGHT routine) interface with the ELINK routine as the data areas are identical.

5.4.4 Use of the ELINK Routine

The network definition statement for the ELINK routine is

ELINK, NO

NO is the number identifying the particular entry link in the network.

This statement results in appropriate statements being written into the control program to call the ELINK simulation subroutine when the simulation is run.

The run time data statement is of the following form:

ELINK, NO, Flow rate lane 1, flow rate lane 2, ..., low rate lane 5

The flow rates are specified in vehicles/hour. If less than 5 lanes are being simulated, the flow rates of the lanes not used may be omitted and will be automatically set to zero. If more than 5 flow rates are specified, ERROR 20 will be printed out and the extra flow rates will be ignored.

The data statement is processed by a subroutine labelled EL1 in the ELINK simulation routine listing (Appendix 12). The subroutine converts the flow rates into flow rate constants to be used by the ELINK simulation segment. The flow chart for EL1 is given in Fig. 5.4.

5.4.5 Storage Requirements

Each ELINK routine in the network requires 5 constant storage and 25 initial storage locations. Three additional locations are required for the control program statements. The ELINK simulation subroutine is 88 words long. The flow chart of the actual simulation segment (ELINK) is given in Fig. 5.5.

5.5 TLIGHT Simulation Subroutine

5.5.1 Introduction

The intersection routine, named "TLIGHT" models a four leg intersection with up to 5 lanes per approach. Vehicles from the outputs of the ELINK or LINK routines are pl ced in queues by the routine. Vehicles are extracted randomly from the queues, as required by the signal conditions, at the user specified saturation rate of the intersection. A Poissonian distribution is used. The technique used for generating this distribution is the same as that used for generating vehicles in the ELINK routine (see 5.4.2).

Once a vehicle is discharged it is assigned a direction, i.e. straight, left or right turning. The direction is randomly assigned

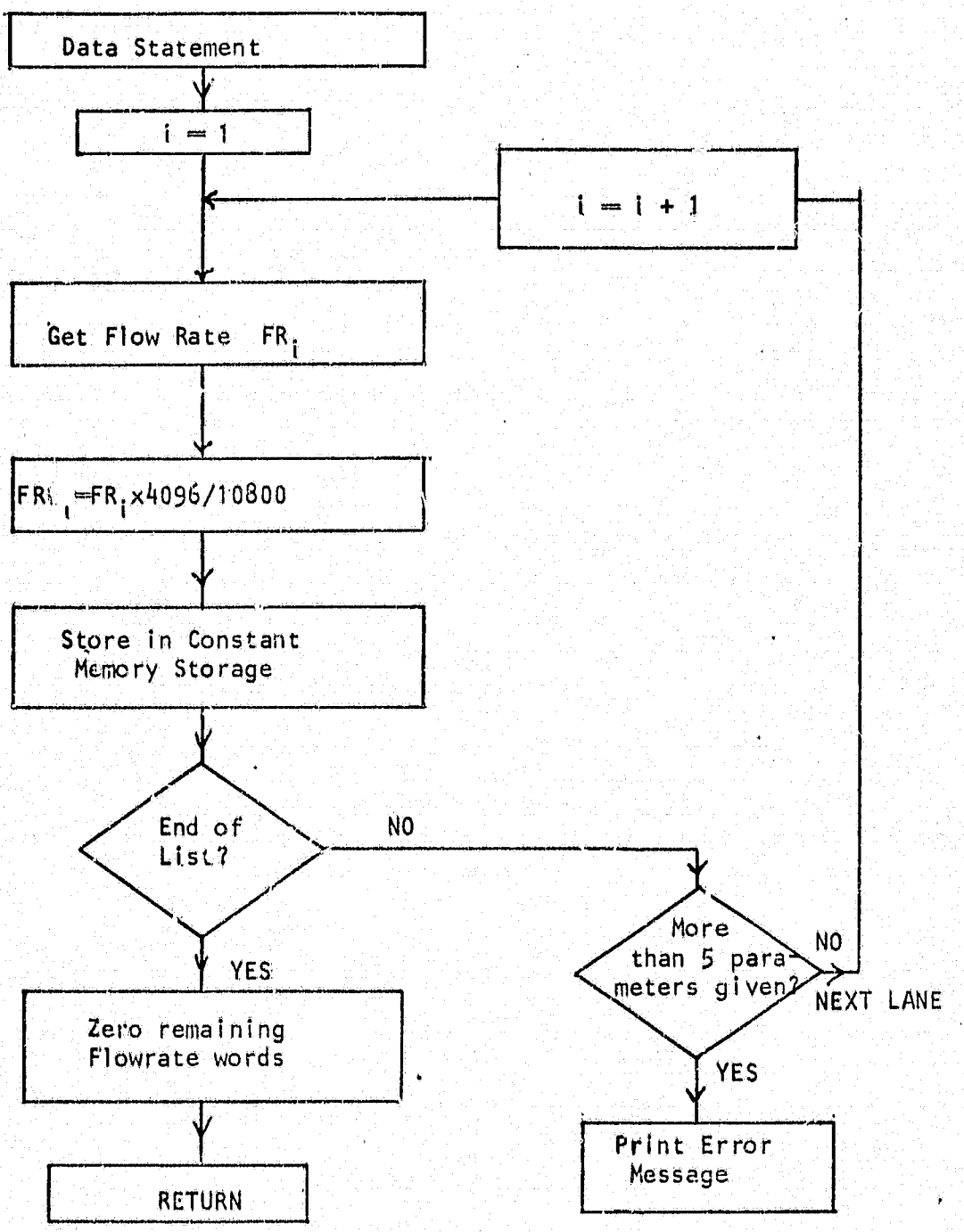

Figure 5.4. Flow Chart of Data Statement Interpretation (EL1 Routine)

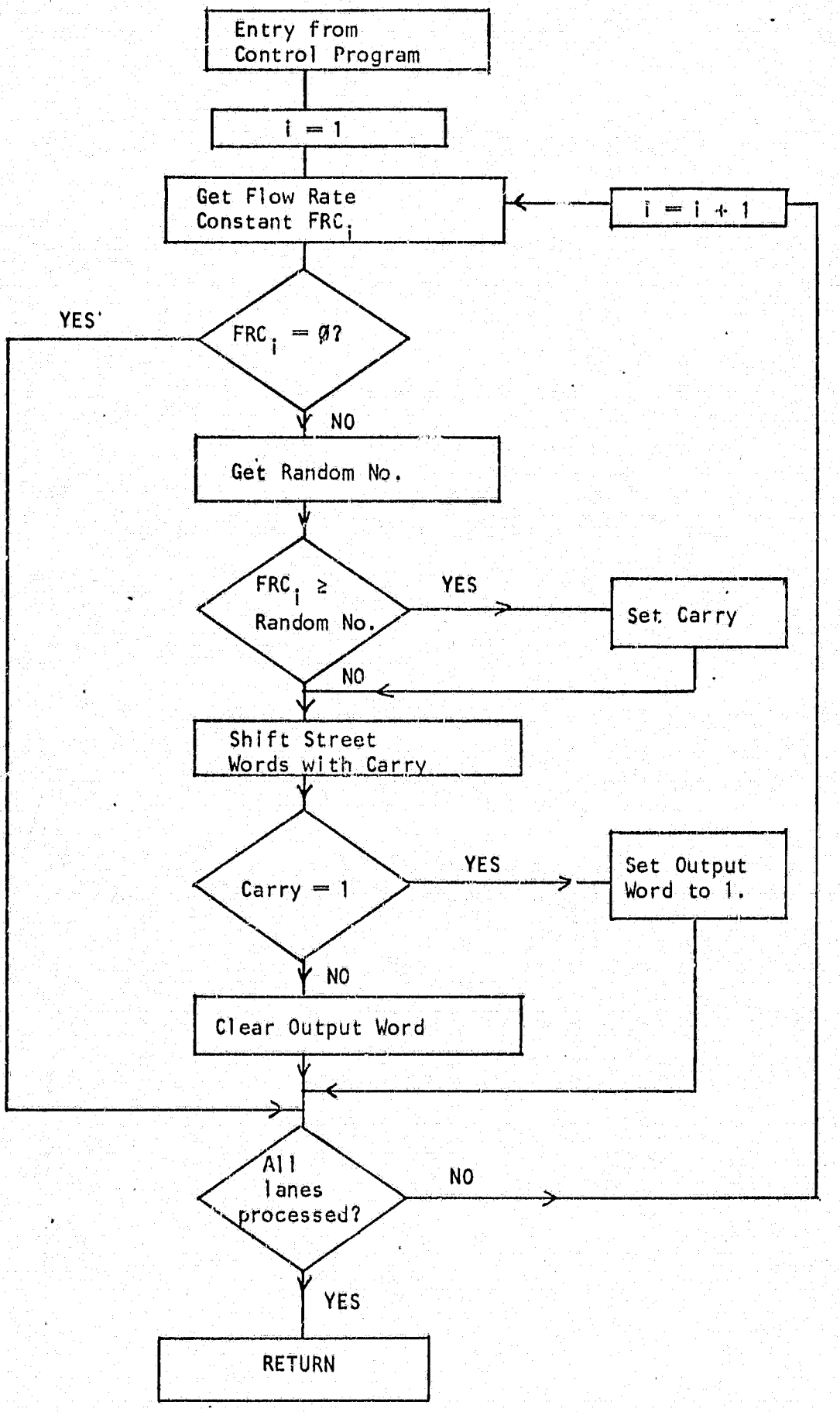

Figure 5.5. Flow Chart of ELINK Simulation Segment

according to user specified probabilities. This is done by using the same technique as is used in the LINK routine for assigning vehicles to lanes (see 5.3.3). If a vehicle has been assigned a straight on or ieft turning direction, it is immediately put into the appropriate output queue and the input queue is decremented by one.

The above modelling of intersections is based on the techniques used by Green [11] for a hardware simulator and is therefore not described in great detail. In the present work, an attempt was made to model right turning situations and signal light changes more accurately.

5.5.2 Simulation of Right Turning Vehicles

In real traffic situations a vehicle may only turn if there is a sufficient gap in the opposing stream of traffic enabling it to turn safely. This gap, often called the acceptance gap is a function of driver behaviour, speed of the vehicle and time gap in the opposing traffic stream.

The hardware simulator of Green used a constant minimum acceptance gap in determining whether a vehicle may turn. Clearly this over-simplification results in unrealistic simulation of turning traffic.

In the present work the acceptance gap is sampled from a random distribution. The type, mean and standard deviation of this distribution can be specified by the user in a similar manner to that used for specifying travelling times in the LINK routine (see 5.3.4).

Once a vehicle has been assigned a right turning direction, the opposing queues are checked. If they are not empty the vehicle cannot turn. The vehicle is then returned to its queue and the queue is marked with a flag to indicate that it is "blocked", i.e. the leading vehicle is a right turner. If a queue is blocked, vehicles cannot

be discharged from the queue until the leading vehicle has been permitted to turn right.

If the opposing queues are empty, a random gap acceptance time is assigned to the vehicle. The opposing lanes are then inspected to determine whether a sufficient gap exists in the opposing traffic to permit the vehicle to turn. Testing lanes for a sufficient gap is facilitated by the arrangement of vehicles in the street. The vehicles are entered into the least significant bit position of the street word and are successively moved to the most significant position by the LINK or ELINK routines. Thus the magnitude of street word is a function of the position of the leading vehicle in the street word. To determine whether a sufficient gap exists, it is only necessary to test if the appropriate lane word has a numerical value less than the gap time word. If insufficient gap length is detected in any of the lanes, the remaining lanes are not tested and the blocked queue flag is set. This, coupled with the speed of the random number generator, results in a fast implementation of the right turning function.

In the work by Green [11], the amber period was regarded as dead time, i.e. no vehicles were permitted to be discharged during this time. This does not exactly correspond with real situations. Although in real situation some vehicles do proceed straight on or turn left during this period, the number that do so are insignificant compared to the number that do so during the green period. Under saturated conditions however, the number of vehicles that turn right during the amber period is large compared with the green period. Thus in the present work, right turns are allowed at the saturation rate during the amber period. This has the additional advantage that a right turning filter phase may be simulated by a suitable choice of

of the amber period.

5.5.3 Signal Cycle

The signal cycle used is fixed time, two phase. Each phase is split into three periods - green, amber and dead. During the dead period, no vehicles are discharged. This is provided to simulate intersections which have a period during which the lights of all legs are red. When the lights of one phase are amber or green the lights of the second phase are red.

In addition to specifying the lengths of the periods for each phase, the user may specify an offset period in order to synchronize the phases of linked intersections. During the offset period the lights are held in the first phase state. At the end of this period the normal green period of the first phase (which is the starting period for all intersections) is started.

The 4 legs of the intersection are numbered for reference purposes The legs are numbered in a clockwise direction as shown in Fig. 5.6.

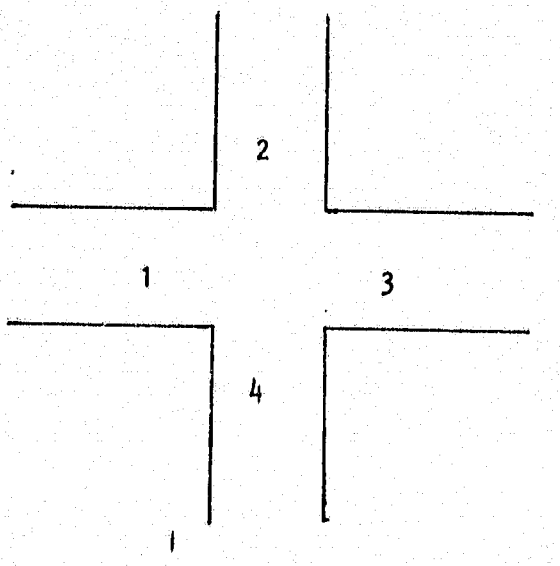

Figure 5.6. Numbering of the Legs of an Intersection

The two phases mentioned above refer to the signal lights for legs 1 and 3 and 2 and 4 respectively. The first phase refers to legs 1 and 3.

A single computer word is used to denote the state of the signal lights and the representation of the various states are shown in Table 1. A single period counter is used and at the end of a period the counter is reset and the signal word is updated to represent the new state. Changing periods is accomplished by shifting the signal word one position to the left. Phase changing is achieved by complementing the most significant bit (bit 0) of the word. The representation uses a minimum of storage and facilitates checking and updating the signal periods.

Table 1.

Signal word (octal)	State
0	green phase 1-3
1	amber phase 1-3
2	dead phase 1-3
100000	green phase 2-4
100001	amber phas 2-4
100002	dead phase 2-4

The TLIGHT routine has been designed to facilitate changing of the signalling. The signalling is performed by the routine labelled UPSIG in the listing. Different control policies may be simulated by changing this routine.

5.5.4 Use of the TLIGHT Routine

The network definition statement for the TLIGHT routine is.

TLIGHT, NO, INPUT ELEMENT FOR LEG 1, NO, INPUT ELEMENT FOR LEG 2, NO, ... INPUT ELEMENT FOR LEG 4, NO.

INPUT ELEMENT is the name of the routine that provides the input to each leg of the intersection, e.g. ELINK or LINK.

Six data statements are provided for the TLIGHT routine. The first four are similar and provide the turning data for each leg of the intersection. The fifth contains traffic light turning information and the last gives information pertaining to the distribution of the acceptance gaps for right turning vehicles. Each statement is independent and can be changed at any time without affecting the data previously specified by the others.

The data statement for each leg is as follows:

TLIGHT, NO, LEG NO, Saturation flow rate, % left lane 1, % right lane 2 . . . . . % left lane 5, % right lane 5.

The saturation flow rate is the maximum flow rate the intersection can handle per lane. It is determined by factors such as the width of the lane, gradient and geometry of the intersection. The percentage of turning vehicles per lane must be specified in pairs as shown for every lane that is used. The percentage flow rates for unused lanes need not be specified.

The traffic light setting data statement is as follows:

TLIGHT, NO, 5, OFFSET, GREEN PHASE 1-3, AMBER 1-3, DEAD 1-3, GREEN 2-4, AMBER 2-4, DEAD 2-4.

OFFSET is the delay before starting the cycle used to synchronize linked intersections.

Phase 1-3 is specified first by the lengths of the periods of green, amber and dead. This is then repeated for phase 2-4. The signal cycle is executed in the order given above and all periods are specified in seconds.

The distribution used for the acceptance gaps is specified by:
TLIGHT, NO,6, DISTRIBUTION NAME, MEANx100, STANDARD DEVIATIONx100.

DISTRIBUTION NAME is the name given to the set of constants used to generate the base distribution. The mean and standard deviation of the required distribution are specified in seconds.

5.5.5 Storage Requirements

Each TLIGHT requires 54 constant, and 27 initial storage words. The control program generated requires 6 words. The TLIGHT simulation routine is 461 words long. Flow charts for the data and simulation subroutines are given in Figs. 5.7 and 5.8 respectively.

5.6 Output Routines

Two output routines are provided which print the queue lengths at intersections. The number of epochs before commencing the print can be specified so that printing will only start after the network has been filled with vehicles. The number of epochs between successive prints can also be specified. The first output routine called "PRINT" prints out the queue lengths of intersections at run time. It is meant to give an immediate picture of an intersection at run time and is meant for interactive working. If more than one PRINT is specified, the queue lengths of different intersections will be interspersed with one another. Each line printed gives the epoch number and the number of the intersection followed by the queue lengths.

The second output routine is called "WRITE" and the queue lengths are not printed at run time but are written onto disk. This information can then be processed further or printed at the end of the run using the COPY routine. This routine sorts the data according to intersection number and prints a heading and the queue lengths at the specified epochs. Each intersection is printed on a new page and

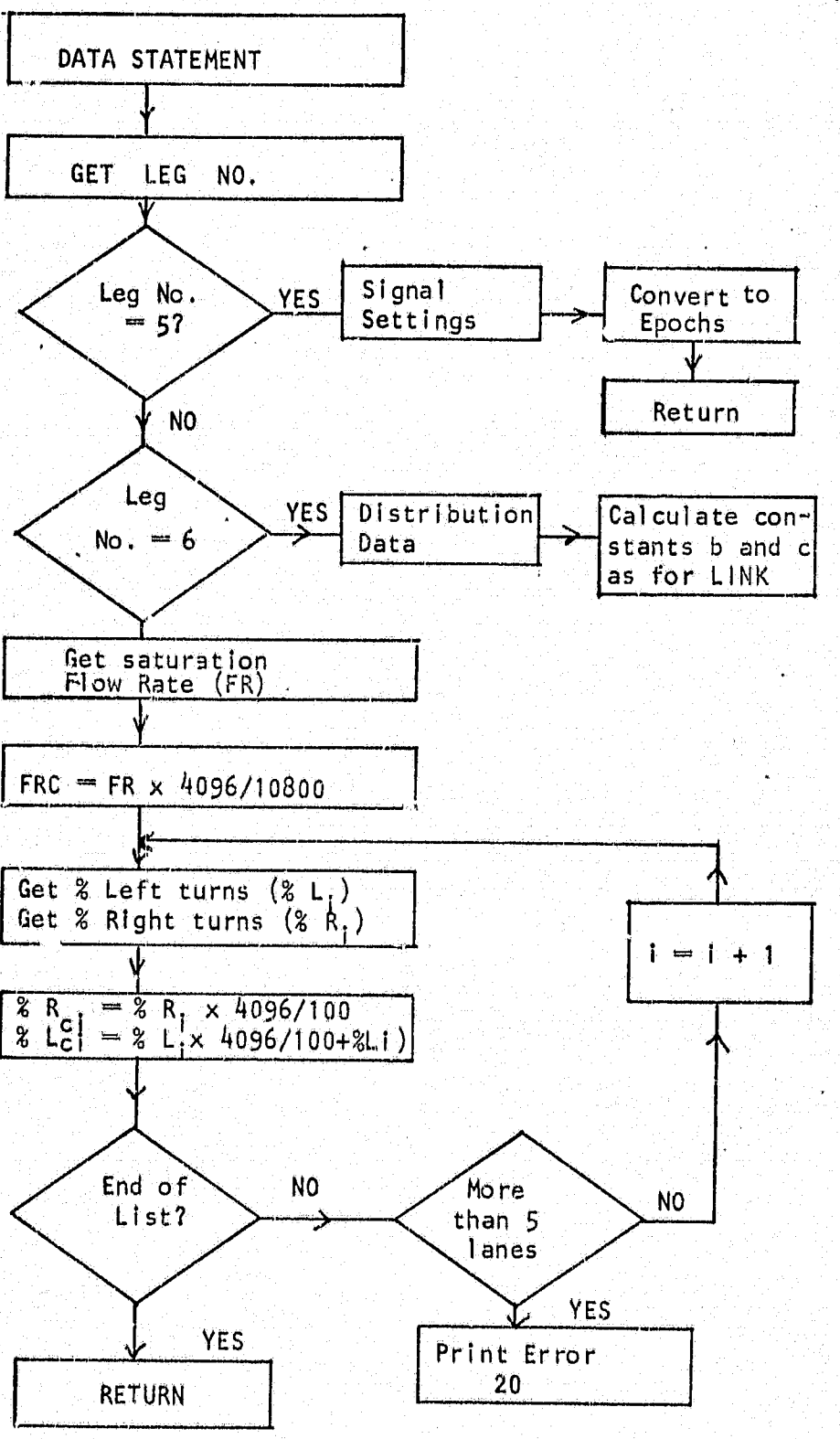

Figure 5.7. TLIGHT Data Routine

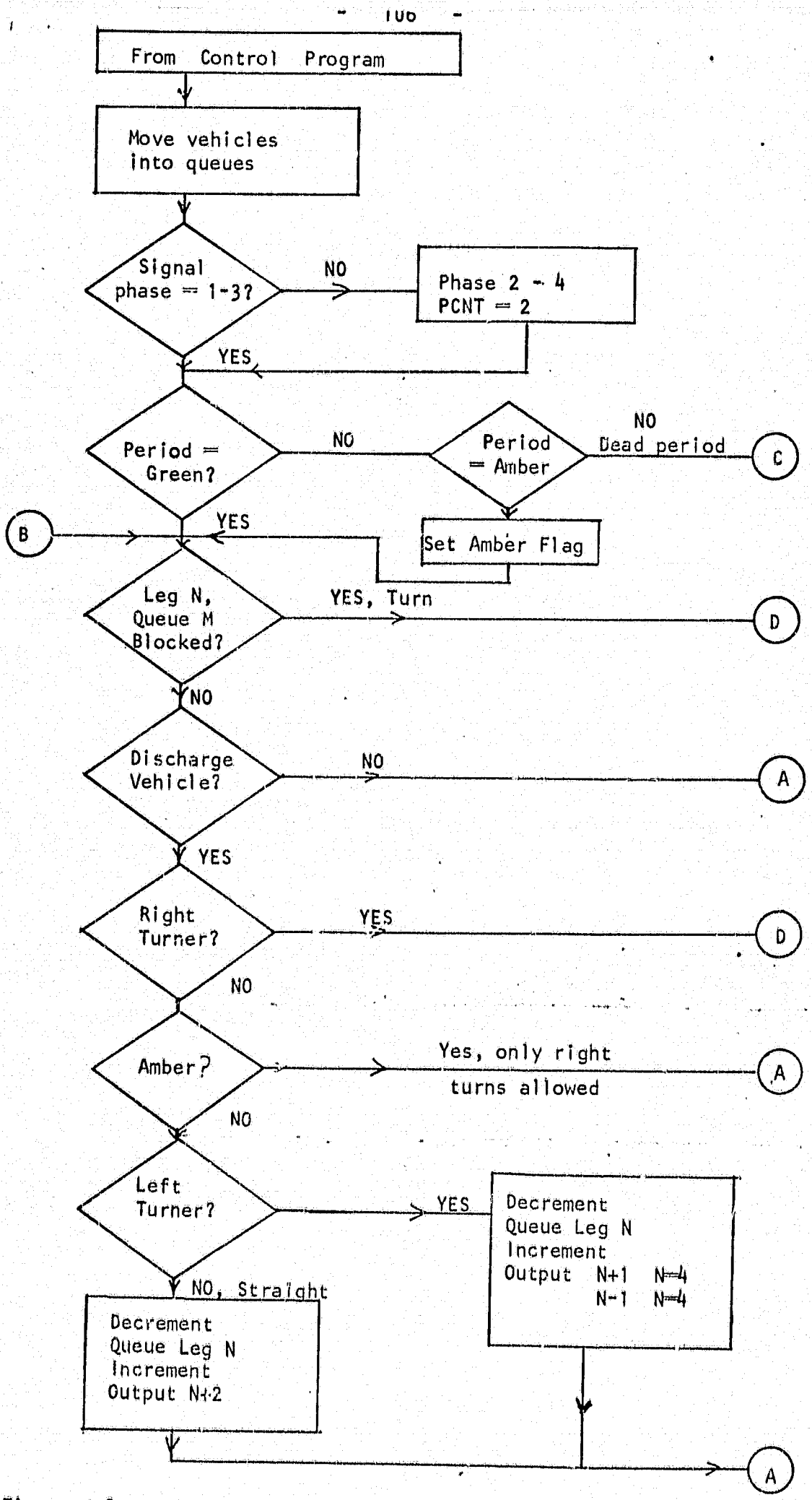

Figure 5.8. Flow Chart of TLIGHT

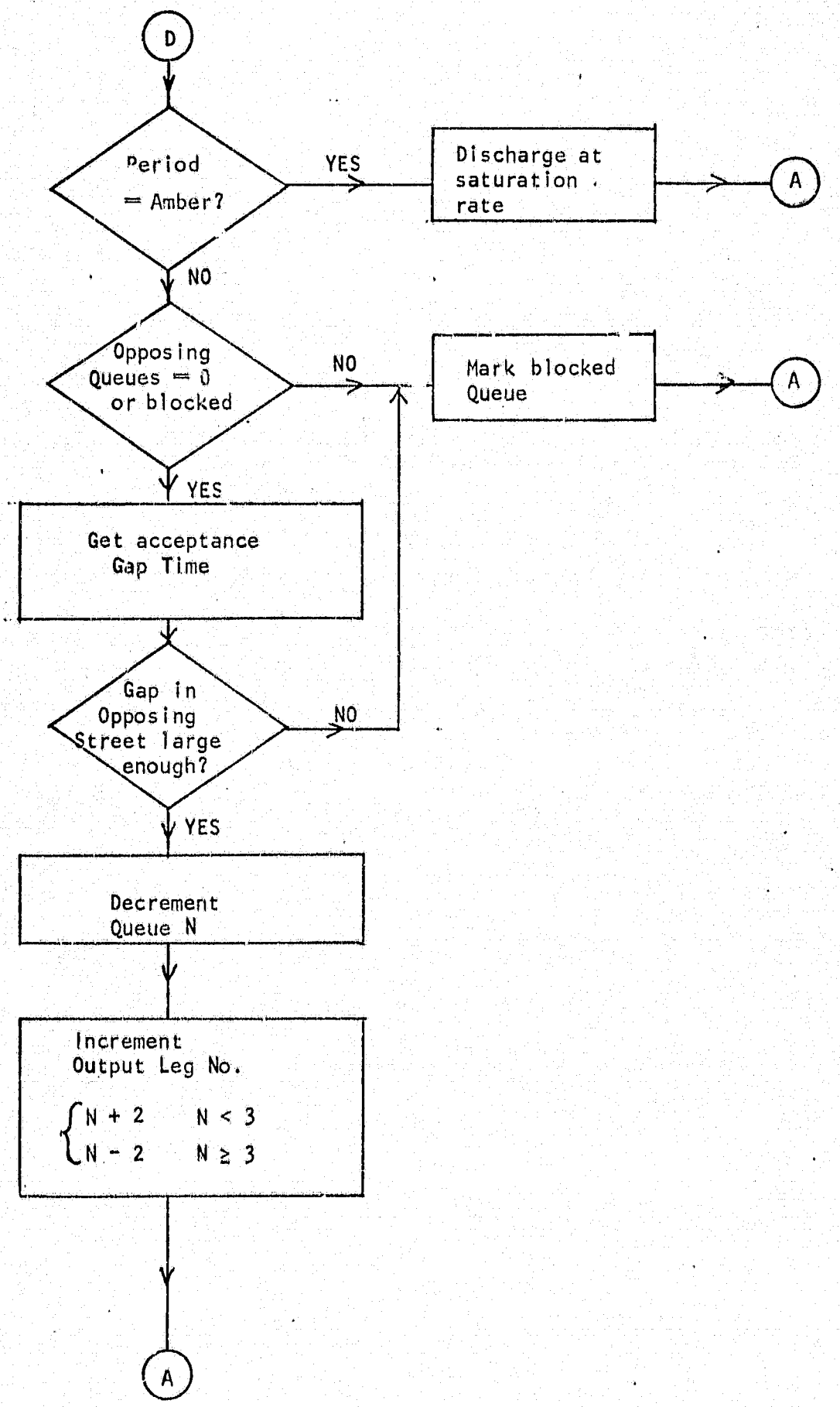

Figure 5.8. (Cont.)

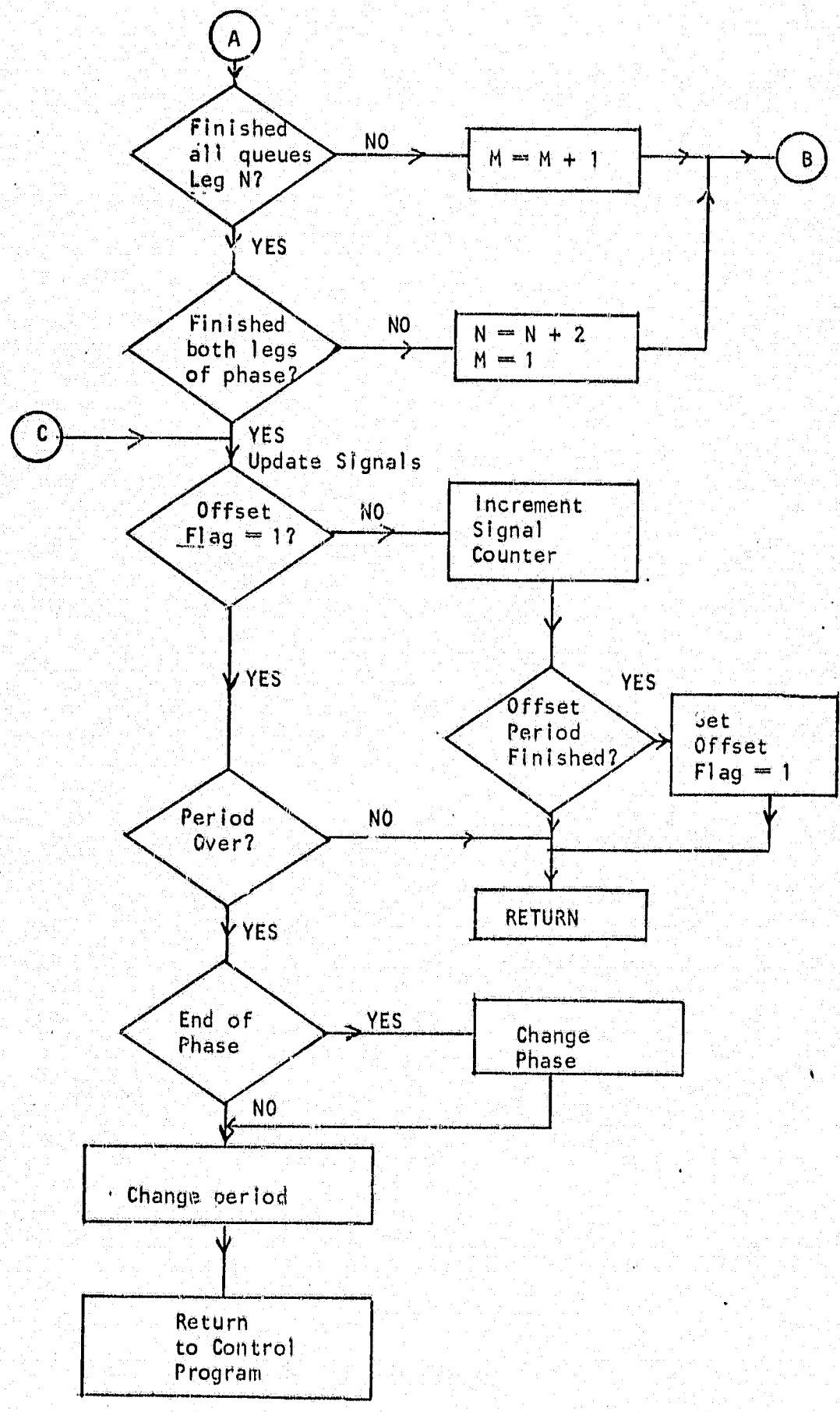

Figure 5.8. (Cont.)

every page has a heading. The WRITE and COPY routines are used to provide tabulated final results for record and presentation purposes.

5.6.1 PRINT Routine

The definition statement for the PRINT routine is

PRINT, NO, TLIGHT, NO

The first NO parameter is the reference number for the print statement. The second number is the number of the intersection whose queue lengths must be printed. The data statement is

PRINT, NO, NO OF EPOCHS BEFORE FIRST PRINT, NO OF EPOCHS BETWEEN PRINTS.

The data subroutine stores this information in the constant storage area for later use by the simulation subroutine.

Each print requires 3 constant storage, 2 initial storage words. The constant storage words are used for storing the TLIGHT NO, and the data specified by the data statement. The initial storage words are used for a counter and a flag which indicates whether the initialisation period before printing starts, is over. The subroutine call requires 4 words and the PRINT user routine is 84 words long. The flow chart of PRINT is given in Fig. 5.9.

5.6.2 WRITE Routine

The structure, storage and execution of the WRITE routine is the same as for PRINT with the exception that data is written onto disk rather than being printed on the console. The definition and data statements are the same and produce the same results. The word WRITE is used in place of PRINT.

The WRITE routine requires two other routines to operate correctly. The first is called IFILE which opens a disk file called ///TRDAT/ on which the data will be written. The program statement IFILE must appear in the initial segment of the program. The second routine

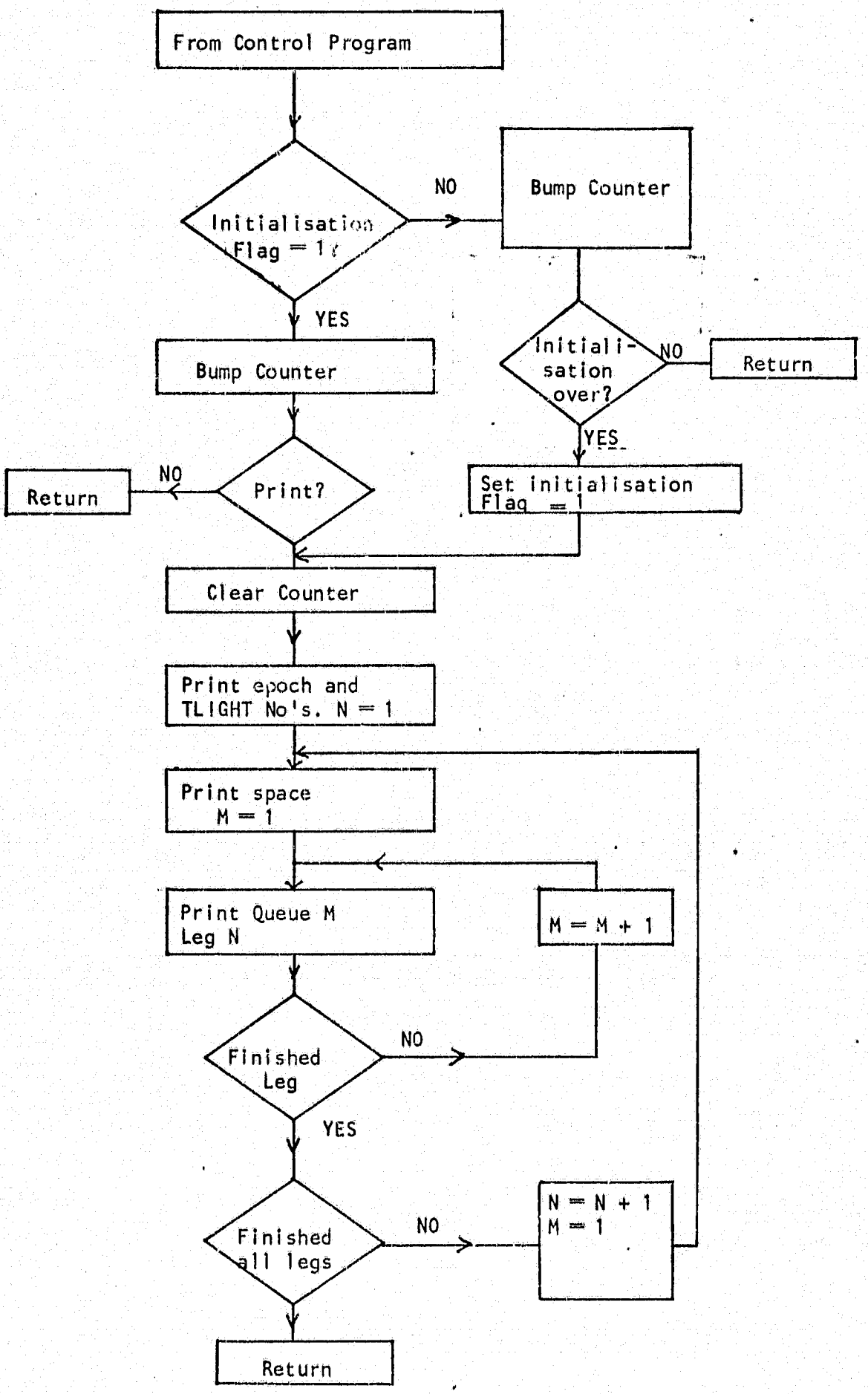

Figure 5.9. Flowchart of Print

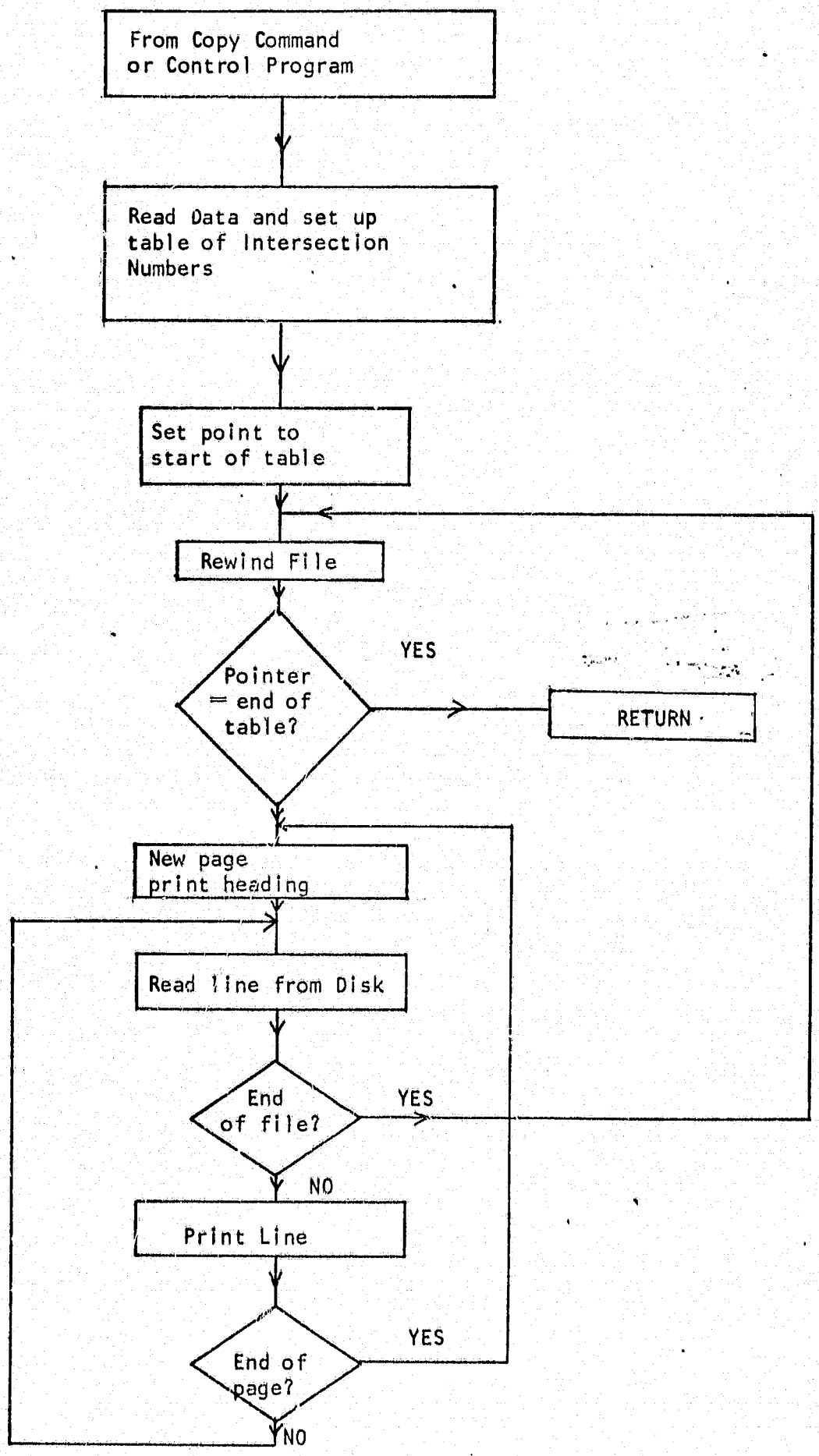

Figure 5.10 Flowchart of COPY

called CFILE writes an end of file mark on the file ///TRDAT/ and closes it. The program statement CFILE must only appear in the TERMINAL segment of the program. The routines IFILE and CFILE use no storage and have no data routines.

5.6.3 COPY Routine

The COPY routine uses the data written onto disk by the WRITE routine to produce tables of queue lengths at intersections. The data on disk is read once to compile a table of intersection numbers that are to be printed. Using this table the data is re-read and the data for each intersection is printed in the order specified by the table. Each intersection is printed on a new page with a heading inserted at the top of the page. If the data for an intersection requires more than one page, a new page with a heading is started.

COPY uses no storage and can be executed either from the keyboard or can be included in the TERMINAL segment of the program. The flow chart is given in Fig. 5.10. The simulation routine containing WRITE, IFILE, CFILE and COPY and a 264 word disk buffer requires 674 words of memory.

5.7 Initialisation Routine

In most simulations it is necessary to be able to change a parameter and rerun the simulation with the same input data. This can be done in this simulator by always resetting the hardware random number generators to an initial state. A routine called IGEN has been provided to accomplish this.

IGEN resets both generations to their starting position (all 1's in both cases) and then shifts the generators from this starting position by a number of pulses specified by the user. In addition the generator of Walker [26] is set up into a mode which uses the computer

memory for storage of the required constants. The definition statement:

IGEN

should only appear in the INITIAL segment of the program.

The data statement

IGEN, NO OF PULSES

specifies the number of pulses the generators must be shifted from their starting positions. IGEN uses no storage and is 34 words long.

5.8 Constants for Arbitrary Distributions

The generator used for generating random travelling times and acceptance gaps requires a set of constants to generate each type of distribution. In order to allow new distributions to be added easily, and to retain the continuity of the FASP system, these constants are arranged in a format similar to a user routine. This permits the user to add or delete distributions easily and allows the distribution to be referred by name.

The constants are calculated using an algorithm of Walker [26] from the relative frequencies of each number. The present traffic routines accept random numbers in the range of 0-63 (i.e. 6 bits). The standard deviation and mean of the base distribution must be chosen carefully so that the distribution is fully described by the relative frequencies of the set of numbers (0,63). For example in the case of the normal distribution, a mean of 32 and standard deviation of 8 was chosen which resulted in the smallest relative frequency being 2×10^{-5}. Thus the tails of the distribution were included and well defined. There are no restrictions on the choice of mean and standard deviation constants b and c that are referred to in 4.4.2.

A program called DIST has been provided to calculate the constants from the relative frequencies. The program outputs the constants onto

disk in the format of a user routine source program which can then be assembled directly with no further user intervention. If all relative frequencies of the distribution can be represented by a mathematical equation, a BASIC program called DISTB can be used to calculate the relative frequencies and punch them onto paper tape in a format acceptable to the DIST program. If the relative frequencies are arbitrary they must be entered manually from the keyboard. The listings of DIST and DISTB are given in Appendix 14.

5.9 Addition of New or Alternative Subroutines

The routines described are not meant to provide the last word in traffic simulation. Rather, they illustrate the principles used in simulating traffic using the FASP executive and are intended as the basis for future work.

The general format for writing simulation subroutines is given in Chapter 3. If new subroutines are required to work in conjunction with existing routines attention should be paid to the data storage arrangements used by the present routines. Inter subroutine communication is provided via the storage handling facility of FASP and new routines should be designed so that their storage areas are compatible with the existing routines. Storage maps for the ELINK, LINK and TLIGHT routines are given in Appendix 9.

If subroutines are added, the traffic simulation system must be reconfigured. This involves linking all the subroutine modules and the FASP system modules together. This process is explained in Appendix 3.

CHAPTER 6

TESTS AND VALIDATION OF THE SIMULATION ROUTINES

6.1 Introduction

Before a simulator can be used it is important to check the validity and accuracy of the assumptions made in formulating the model. This is often a formidable task as simulations are usually used to model highly complex situations. Validation is also complicated by the fact that while a simulation model may accurately simulate a given situation, it may be totally useless for many others.

Two methods are usually employed in testing traffic simulators. A theoretical approach may be used to compare simulation results with results obtained from existing traffic simulators or statistical theory. An example of this approach is the method used by Green [11] to test the simulation of arrival and discharge of vehicles at an intersection. This situation is a deterministic queueing problem and can be described by the Polluczek-Khintchine formula [39]. For the evaluation of the theoretical results certain assumptions had to be made concerning the frequency distribution of arriving vehicles and the rate of discharge of vehicles. Since these assumptions are the same as those made in formulating the simulation model, this type of test can only verify that the simulator is operating correctly. It does not prove that the simulator is accurately modelling real traffic. Thus the use of theoretical verification is of limited value as it can only be applied to isolated, idealised cases for which theory exists and does not necessarily guarantee correspondence with real life situations.

A second and more reliable approach is to select a representative real life traffic network, simulate it and compare the results with the real life situation. This approach was used by Bruggerman et al.[1] and is very clearly described in the literature. It was used to calibrate, validate and test the sensitivity of the UTCS micro traffic model. A large amount of data needs to be gathered for this approach and in the example aerial and roof-top photography was used in addition to the normal manual methods of counting traffic. This method is by far the most satisfactory but is extremely costly, tedious and time consuming. The traffic network must be carefully chosen so that it contains most of the features normally encountered in road networks.

An alternative to the above approaches is to validate a new simulation model by comparing it to another simulation model which has been previously validated. This however, relies on the integrity of the first simulation.

For the validation of the simulator described in this dissertation, the first approach was rejected as only small elements of the model could be checked rather than the operation of the complete network. In addition this approach would only verify that the software is correct but would say nothing about the assumptions made in formulating the model. The second approach is the most suitable, however, it is completely beyond the scope of this work. It should be considered seriously for future work. The alternative approach of comparison was considered to be the most suitable for this work but unfortunately sufficient input and output data from other simulators could not be found in the literature. Thus no attempt was made to rigorously validate the model. Instead the elements of the simulator were checked for correct software operation and a fictitious network

consisting of four linked intersections was simulated. This demonstrates the use of the simulator and was used for estimates of run time. It must be stressed that this does not constitute a validation of the simulator and extensive work must be carried out before it can be said that traffic is being accurately modelled.

6.2 Tests on the Software Routines

6.2.1 Introduction

Various tests were performed on the traffic subroutines to check that they were operating correctly. This was facilitated by the use of a standard Debug program [41] which allowed the flow of the routines to be monitored and permitted examination of locations in core. In certain cases a quasi statistical approach was used in which flow rates were specified and the routine was run for a known number of epochs. The outputs of the various routines were then examined to assertain whether they contained values in an expected range.

6.2.2 Tests on the ELINK Routine

The ELINK routine is relatively simple and therefore easy to test. Preliminary testing was performed on the data routine to confirm that the flow rate constants were being calculated correctly. The program was then run with various flow rates and the initial core storage was examined after every epoch to verify that vehicles were being generated and then stepped down the street. After these tests were satisfactory a simulation program was written to simulate a single entry link feeding into one arm of a traffic light.

The traffic light was programmed so that the signal facing the link was red. The simulation was run for 1000 epochs and the queue lengths were then examined. The queue lengths were found to be close to the value which was calculated from the flow rates in each