1981

SIMULATION OF A MULTIPROCESSOR COMPUTER SYSTEM

SALIH, A.M.

http://hdl.handle.net/10026.1/1965

http://dx.doi.org/10.24382/4987

University of Plymouth

SIMULATION OF A MULTIPROCESSOR

COMPUTER SYSTEM

A.M. SALIH

Submitted in partial fulfilment of the requirements for the Doctor of Philosophy degree.

Sponsoring Establishment : Plymouth Polytechnic,
Drake Circus,
Plymouth

Collaborating Establishment : The General Electric Company Ltd
GEC Hirst Research Centre
Wembley.

MAY, 1981

ABSTRACT

A.M. SALIH

SIMULATION OF A MULTIPROCESSOR COMPUTER SYSTEM

The introduction of computers and software engineering in telephone switching systems has dictated the need for powerful design aids for such complex systems. Among these design aids simulators - real-time environment simulators and flat-level simulators - have been found particularly useful in stored program controlled switching systems design and evaluation. However, both types of simulators suffer from certain disadvantages.

An alternative methodology to the simulation of stored program controlled switching systems is proposed in this research. The methodology is based on the development of a process-based multi-level hierarchically structured software simulator. This methodology eliminates the disadvantages of environment and flat-level simulators. It enables the modelling of the system in a 1 to 1 transformation process retaining the sub-systems interfaces and, hence, making it easier to see the resemblance between the model and modelled system and to incorporate design modifications and/or additions in the simulator.

This methodology has been applied in building a simulation package for the System X family of exchanges. The Processor Utility Sub-system used to control the exchanges is first simulated, verified and validated. The application sub-systems models are then added one level higher, resulting in an open-ended simulator having sub-systems models at different levels of detail and capable of simulating any member of the System X family of exchanges. The viability of the methodology is demonstrated by conducting experiments to tune the real-time operating system and by simulating a particular exchange - The Digital Main Network Switching Centre - in order to determine its performance characteristics.

ADVANCE STUDIES UNDERTAKING IN CONNECTION WITH THE PROGRAMME OF RESEARCH IN PARTIAL FULFILMENT OF THE REQUIREMENTS OF THE DOCTOR OF PHILOSOPHY DEGREE

1) Mini-Computers and their Applications
Plymouth Polytechnic, 7-18 June, 1976

2) Software Engineering for Computer Controlled Switching Systems
IEE Vacation School, Essex University, 11-16 July, 1976

3) Stored Program Control of Telephone Switching Systems
IEE Vacation School, Essex University, 11-16 Sept. 1977

4) Simulation with SIMULA and SIMON
Exeter University, Oct-Dec. 1977

5) Switching and Signalling in Telecommunications Networks
IEE Vacation School, University of Aston in Birmingham
10-15 Sept. 1978

6) Association of SIMULA Users' Seminar
University of Bradford, 19th Dec. 1978

7) Queuing Theory
Imperial College, 8th May - 5th June, 1980

8) Data Networks
Essex University, 13th Jan - 10th Feb. 1981

9) Traffic & Queuing Systems
Essex University, 19th Feb - 19th March, 1981

DECLARATION:

The author of this thesis hereby declares that:

(1) While registered as a candidate for the Ph.D. degree the author has not been a registered candidate for another award of the CNAA or a University during the research programme.

(2) No material contained in this thesis has been used in any other submission for an academic award.

....
A. M. SALIH

ACKNOWLEDGEMENTS

Acknowledgement is due to Plymouth Polytechnic whose financial support has made it possible to carry out this research and to GEC Telecommunications and GEC Hirst Research Centre whose interest in this work was motivating. My thanks and gratitude to the Director of Research, Dr.E. McQuade and to my Supervisors Mr. H. Gray and Mr. G.J.Owen for their continued encouragement, advice and interest which made it possible to complete this programme of research.

I would like to acknowledge the contribution made to my understanding of System X sub-systems and simulation approaches and languages by all those whom I met and have discussed these topics over the past four years. In particular, I would like to thank Messrs J. Fanstone, J. Nissen and G. Dent of GEC Telecommunications, F.M. Clayton, D. Bear and V.R. Barker of GEC Hirst Research Centre, and R.J. Cunningham of the Imperial College whose interest and provision of computing facilities made it possible to complete the last phase of this research.

My thanks are also conveyed to the other establishments, namely The Institute of Marine Environmental Research and Exeter University Computer Unit who generously provided us with the necessary computing facilities during the first three years.

Last but not least I would like to thank my wife Fatima for her patience and understanding during the course of this research.

CONTENTS

PAGE

CHAPTER 1

CHAPTER 2

CHAPTER 3

CHAPTER 3 continued.......

PAGE

CHAPTER 6

CHAPTER 7

PAGE

CHAPTER 7 continued....

CHAPTER 8

APPENDICES

REFERENCES

FIGURES

FIGURES FOR CHAPTER 1

FIGURES FOR CHAPTER 2

FIGURES FOR CHAPTER 3

FIGURES FOR CHAPTER 7

TABLES

PAGE

TABLES FOR CHAPTER 2

TABLES FOR CHAPTER 3

TABLES FOR CHAPTER 6

TABLES FOR CHAPTER 7

GRAPHS FOR CHAPTER 6

CHAPTER 1

INTRODUCTION

With the advent of digital computers, it was proposed back in 1955 that digital computer techniques could also be used to control telecommunication switching systems (KAWA 71). The work was initiated by the Bell Telephone Laboratories of the U.S.A. and resulted in the development of the Number 1 Electronic Switching System (No.1 ESS) which went into public service in 1965. There are now over twenty different designs of computer-controlled telephone exchanges and many more in the advanced development stage (HILL 76A). An increasing number of telecommunication administrations in various countries have been introducing or planning to introduce computer-controlled switching systems into their communication networks.

Although the present electromechanical telephone switching systems, such as the step-by-step and crossbar systems, are offering a reasonable and economical service, stored program controlled (SPC) switching systems possess more overall capability than conventional systems. This overall capability manifests itself in the following features: (Section 3.3).

1. A higher level of system security
2. The ability to interwork with the older existing network.
3. Labour saving as a result of easier administration and considerably reduced maintenance effort.
4. A range of new services to the customer.
5. Space saving, power saving and higher traffic capacity.
6. Flexibility to changes and ability to make use of advances in micro-electronics.

These features have been brought about by the application of software engineering in conjunction with the development of powerful telecommunication-oriented processors. The software qualities of richness of function and versatility have made it possible for example, to expand basic call processing with wide varieties of special features for business and individual subscribers. The telecommunication industry is in a position to make the most of the opportunities presented to it by software as the technologies of communications, computers and information management rapidly converge, bringing about what is often referred to as the 'information society'. However, the complexity of software and hence stored program controlled systems called for the provision of suitable design tools and performance evaluation aids. In the domain of performance evaluation aids modelling techniques such as empirical, analytical and simulation modelling assumed a special significance recently, particularly with the development of multiprogrammed, virtual memory multiprocessor systems (SVOB 76, UNGE72). The performance of such complex systems is a function of several parameters.such as the system configuration, the resource management policies of the operating system and the efficiency of the application programs. Performance evaluation is required during the design stage as well as during its operational life to aid in the design decisions, in fine-tuning and assessing the system capacity.

The most potentially powerful and flexible of the performance evaluation techniques of computer and SPC systems is simulation (CALI 67, SVOB 76). The concept of simulation is both simple and intuitively appealing,allowing the user to experiment with systems (real and proposed) where it would be impossible or impractical otherwise. Yet the application of simulation for the analysis and evaluation of computer system performance is an important and demanding field. It is important because performance is one of the prime considerations in evaluating a computer system and demanding because it requires a deep understanding of the inner system mechanisms, both hardware and software, knowledge of the processing requirements and the workload characteristics. As a result of the demand for more powerful and flexible computing systems and better performance per unit cost, computer systems have become increasingly complex, and system performance increasingly difficult to assess. To overcome this difficulty researchers resort to a combination of simulation and measurement techniques where the workload only is simulated (CASY 77), or using a combination of analytical modelling and simulation techniques (JOUB 78), or simulation of certain aspects of a system at a time (WEAT 73) or a coarse flat-level simulation of a class of computer systems that give an approximate indication of the overall performance characteristics (UNGE 72).

In this research we attempt to develop a simulation methodology that will allow us to develop a multi-level process-based simulation of the GEC Mark II BL microprogrammed, telecommunication-oriented multiprocessor system both in hardware and software as well as the members of the family of System X SPC switching systems, which are controlled by Mark II BL System, in a flexible level of detail. The aim is to provide a computer-aided design package that will enable the software engineers to assess their designs and strategies in the areas of the real-time operating system and the telephony applications software.

This is in contrast with the previous simulation work in the SPC field where the simulators were either real-time environment simulators or concentrated on one aspect or sub-system only,such as traffic capacity, networks and control sub-system studies (ANDE 72, JAME 78, POVE 65). Real-time environment simulators have been pioneered by a subsidiary of ITT, BTM of Belgium, since 1966. The technique relies on simulating the call types and the environment of the SPC switching system such as the cross-points of the switching network, the junctors, peripheral devices and subscribers and trunks in a separate processor. The simulator processor is then run in real-time with the system processor that houses the operating system and application software. The main use of environment simulators is for testing and debugging programs, though they have been used recently for traffic capacity studies (FONT 71, GRUS 76, DGWE 78). Environment simulators proved to be effective in reducing the costs of testing and debugging of application and diagnostic software and the technique has been adopted by some other SPC and computer systems manufacturers (BECK 73, GUIT 76, CHAR 78).

The major problem with environment simulators, however, is that they can not be used for designing and evaluating the real-time operating system. They assume the presence of a proved design of one. Their use for testing and debugging applications and diagnostic software comes at a later stage of the design process and their development is costly, considering the cost of hardware, software and the synchronization circuitry involved. The alternative of an all-software flat simulator is often difficult to verify and validate and does not lend itself to a non-simulation specialist because of the lack of resemblance between the real and simulated system. This is particularly true if the simulation approach adopted is that of the event-scheduling approach (FISH 73). Examples of such simulations are the British Telecommunications simulations of the Mark II BL system, the Digital Main Network Switching Centre and the

Pre-processor Utility (SINC 80).

Here we are suggesting a different approach to both environment and flat simulations - an all-software multi-level process simulation. We will show how and why this alternative methodology of simulating SPC systems is attractive, cost-effective and capable of simulating both the real-time operating system and the applications software. The simulator developed in this research is a powerful tool for computer aided design and evaluation of a class of SPC systems, that is the System X family of exchanges. The real-time operating system level of the simulator can be used to evaluate the present system design, tune the system and assess any proposed modifications. The applications software level can be used to evaluate a particular SPC exchange from the family of System X of exchanges.

The immediate objectives of the simulator developed in this research are:

1. To provide a tool for the designers of the real-time operating system of the System X processor utility sub-system, that is the GEC Mark II BL multiprocessor system, which would permit

   a) tests of the performance of the prototype design

   b) evaluation of possible modifications and extensions.

2. To provide a tool for evaluating the performance of applications software for the System X Digital Main Network Switching Centre (DMNSC), which uses the processor utility sub-system and other sub-systems.

The second objective is concerned with a coarser level of detail than the first, suggesting that the use of a hierarchical multi-level process simulation may be appropriate. This approach has proved to be both straightforward to implement and adaptable in its application since:

1. The amount of detail at each level in the hierarchy is kept relatively low, reducing the tasks of verification and validation at each level.

2. The simulator is structured so that it corresponds closely with the structure of the system, making it easier to understand and adapt to system design changes.

3. It is possible to exploit the confidence in the simulator, once established for one hierarchical level by using the features of this level as a base for simulating applications.

Multi-level simulation has been suggested before (ZURC 68). However, Zurcher et al used the term multi-level modelling in a different context and meaning. What they suggested was an iterative method with the concurrent existence within a single model of several representations of the system being modelled, at different levels of detail using an activity-based simulation language (a collection of Fortran sub-routines). The methodology and philosophy we suggested in this research for SPC systems simulation is primarily concerned with the inexperienced user who can build up his model of a particular SPC system from a library of models, where no more than one representation of a module exists. The objective of the Zurcher et al methodology is to provide a facility to design a computer system from the outside inwards such that each level of abstraction consists of a simulation program, constructed of a hierarchy of procedures. The simulation program is controlled by the program on the level above, which is making more global decisions based on its own variables. These next-higher level variables are an abstraction of those on the current level and, hence, are continually updated when the values of variables on the current level change.

The goal of the Zurcher methodology is to use the lowest-level model to produce the actual system, by replacing the basic algorithms in the lowest level of abstraction and the facilities which are provided by the simulation system, such as sequencing and list processing facilities, by the real-life mechanism from which the system is to be built. This is in contrast to the objective and methodology we adopted. The objective here is to provide an integrated computer-aided design package oriented towards the community of software engineers developing the system as the main users. The application programs can be tested and debugged individually using an existing emulator program. Then the performance of individual application programs when interacting in a model of a particular System X exchange can be assessed and modified using the simulator. The transformation of the real system modules into their corresponding models in a 1-to-1 transformation process preserving the interfaces in the real system is very valuable in increasing the understanding and confidence of the software engineers in the simulator. No attempt is made to use the simulator to produce designs of the application programs from their models in the manner followed by Zurcher et al. This is because of the great volume of software in SPC systems. For example, the call processing sub-system amounts to about 100K statements. The objective of the simulator is to assess and tune the present design and not to produce new designs altogether. Applying Zurcher et al methodology would produce a simulator approaching half a million of statements! However, the top-down design implied by Zurcher et al methodology has been used by some simulation analysts in the field of SPC systems (JAME 78). Others have used a variation of their methodology where the environment is simulated in a simulation language such as SIMULA as well as the system model. The simulation of the environment and the system is run on a big computer, tested and debugged. The system model then becomes the implementation (LOVD 77, BELS 78)

The design philosophy of System X simulator called for a careful selection of a simulation system to implement the simulator. Portability was very much in mind, and led initially to the use of an available FORTRAN-based simulation language, CSL (BUXT 62). The processor sub-system,including the real-time operating system kernel and periodic application processes were simulated in CSL, an activity-based simulation language, but the resulting model did not fulfil all of the above stated requirements. From this first attempt,it was apparent that out of the three discrete event simulation approaches (event-scheduling, activity scanning and process interaction), the last approach is the most appropriate one. The notion of a process and a process instance in a simulation sense is very close to the notion of a software process as used in SPC switching systems.

SIMULA (DAHL 70) is an ALGOL-based general-purpose language with a structural concept,called the CLASS which made it very convenient for developing self-contained sub-systems for special applications. Indeed, the CLASS concept is used within SIMULA itself to transform the general-purpose language into a process based simulation language. Another useful concept is that of a VIRTUAL PROCEDURE. Together with the CLASS prefixing concept,it is possible to simulate a complex system using a multi-level structure with many levels of refinement.

After the first attempt at simulation using CSL, an attempt was made to implement a multi-level process simulator using SIMULA. This was most successful. Here we give some demonstrations of the reasons for this. It is the CLASS prefixing concept which is most useful in realising a multi-level structure. The real-time software processes which we are modelling have some identical data structures and must be able to call for the same services of the real-time operating system. In the simulator this can be achieved by defining a simulation process class AP that contains these common data structures and the common interfacing procedures. A SIMULA definition of AP is given in Figure (1.1).

```
PROCESS CLASS AP (PROCESS INDEX);  INTEGER PROCESS INDEX;
COMMENT ** AP FOR APPLICATION PROCESS **;
BEGIN
COMMENT ** NOW DEFINE COMMON DATA STRUCTURES **;
REAL TIMELEFT;
REF(CPU)MYCPU;
REF(PROCESS ALLOCATOR) PA;
REF(HEAD)INPUTQ;
INTEGER ARRAY PROCESS DESCRIPTOR(1:J); TASK INDEX TABLE(1:K,1:L);
.
.
.
COMMENT ** NOW DEFINE INTERFACING PROCEDURES **;
PROCEDURE HAND ;---;
PROCEDURE FETCH(N) ;---;
.
.
.
PROCEDURE BLOCK(N) ;---;
END ** OF AP DEFINITION **;
```

FIGURE (1.1)

These common data structures and interfacing procedures are automatically inherited by the process class which simulates a particular application sub-system, for example the call processing sub-system, merely by prefixing the new simulation process by the identifier AP. We arrived at the choice of SIMULA after a careful review of the available discrete-event simulation approaches and languages. These approaches and languages are thoroughly discussed in Chapter 2,together with the criteria for simulation programming language selection.

The levels in the simulator are shown in Figure (1.2). At the bottom is the SIMULA system,which provides the simulation concepts and the language constructs. The GEC Mark IIBL multiprocessor system model,including a detailed model of the operating system is one level higher. On the next level reside the application software simulations. Each level assumes the services of the level below. The level of application software simulation is open-ended. Models of new application software at different levels of detail may be included to form a library of simulation sub-systems as illustrated in Figure (1.3). A user can easily assemble a model of a particular exchange configuration by initiating process instances of the relevant sub-system simulations from the level below.

Chapter 3 outlines the development of telephone switching systems from the manual exchange to the present stored program controlled switching systems. Since a major impetus of this research is the study of the processor utility sub-system, telecommunications processors are analysed in greater detail including their characteristics, configurations and reliability. SPC software organisation is then introduced. The System X family of SPC exchanges is outlined revealing the complexities of the designs which highlights the need for performance evaluation tools as design aids.

The processor utility sub-system, that is the GEC Mark II BL multiprocessor system, is explained in Chapter 4,both in hardware and software in some detail,to show how the system modules interact and function. The description of the processor utility model then follows in Chapter 5,revealing the 1-to-1 transformation process and,hence, the resemblence between the system's hardware and software modules and the corresponding simulation processes. The processor utility model developed uses the multi-level process approach. The detailed modelling

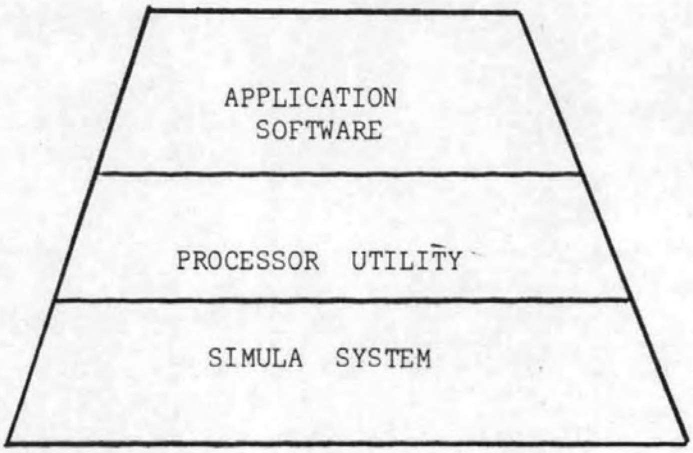


FIGURE (1.2) : HIERARCHICAL STRUCTURE OF THE SIMULATOR

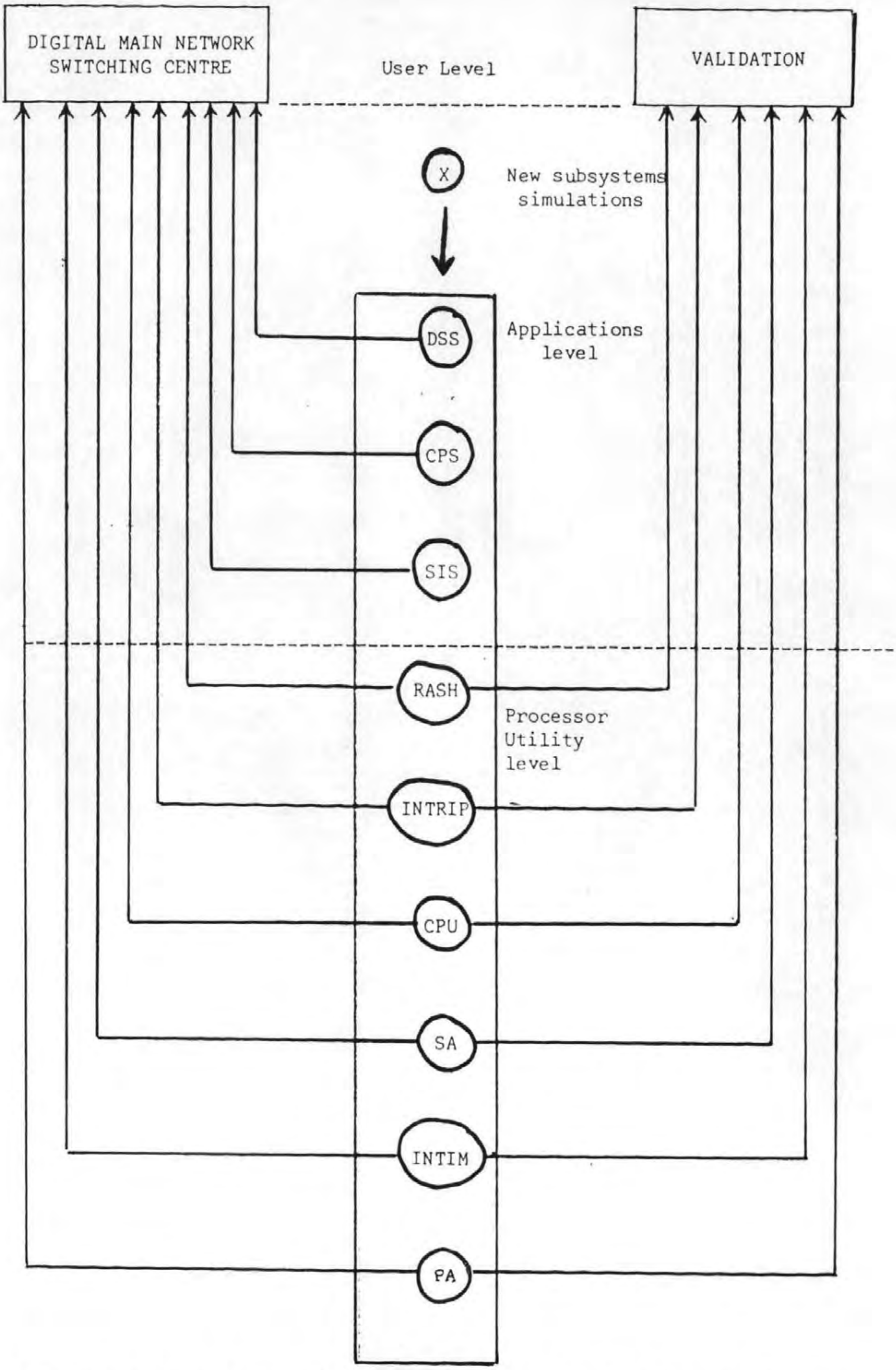


FIGURE (1.3) : THE OPEN-ENDED MULTI-LEVEL PROCESS-BASED SYSTEM X SIMULATOR of the process allocator,which is the central and most important part of the real-time operation system,is explained. The other module models at a coarser level of detail are also outlined.

The processor utility level is first verified and validated. The logic of the simulation is checked using a detailed tracing facility and a model of a hypothetical digital exchange. In order to validate the processor utility model,a series of experiments are conducted in a controlled environment using different configurations of the real processor utility sub-system. Use is made of operating system process instances to represent workload resource demand, that is the system is self driven, and relevant statistics are compiled. The same series of experiments are then duplicated using simulated configurations of the processor utility. The results demonstrate the credibility of the processor utility model and inspire confidence in its findings. These aspects of verification and validation are discussed in detail in Chapter 6.

After validation, the processor utility model is used to investigate several design aspects of the real-time operating system. One example is the interrupt handling mechanism of the multiprocessor system. It is found that a reduction in overhead can be achieved by slight modifications to the original interrupt handling mechanism. Another example is the study of the feasibility of introducing a new service call to the operating system for periodic processes in order to replace two existing calls. The feasibility of such a new call is demonstrated by designing and conducting a number of experiments on the simulator. The models of System X sub-systems which are the basic building blocks of the Digital Main Network Switching Centre (DMNSC) are developed on a level above that of the processor utility model. This application level is open-ended and sub-systems models may be added to form a library of sub-systems models. Using the detailed

trace output, the model of the DMNSC is verified against the message sequence chart of telephone calls through the exchange. This model is then used to obtain delay statistics necessary for the design validation of the call processing sub-system of the exchange.
The DMNSC and its simulation are the subject matter for Chapter 7.

CHAPTER 2

DISCRETE-EVENT SIMULATION APPROACHES AND LANGUAGES

2.1 DISCRETE-EVENT SIMULATION APPROACHES AND TECHNIQUES

2.1.1 Systems

In the context of simulation, by a system we mean a collection of related objects or entities, each characterised by a set of attributes assuming numerical or logical values that may themselves be related (FISH 73). Generally, every system is characterised by three features; it has boundaries, exists in an environment and is made up of sub-systems. The environment constitutes the set of surroundings in which the system is embedded, whereas the boundaries distinguish the entities in a system from those that make up the environment. The system is influenced by the environment through the input it receives from the outside world. This input is transformed by the process operating in the system,resulting in the output of the system.

With regard to the dynamic behaviour of a system, the system progresses through different states characterised by the numerical or logical values of its attributes over time. The system is said to be in a steady-state if the probability of being in some state does not vary over time. The steady-state probabilities are independent of the state in which the system started and are the limiting distribution of the transient states probabilities.

In general, the objectives in studying a system are to learn about how change in system state occur , to predict change and to control change. Particular studies are usually a combination of these objectives of varying emphasis. The ultimate objective always remains to optimise performance in some sense.

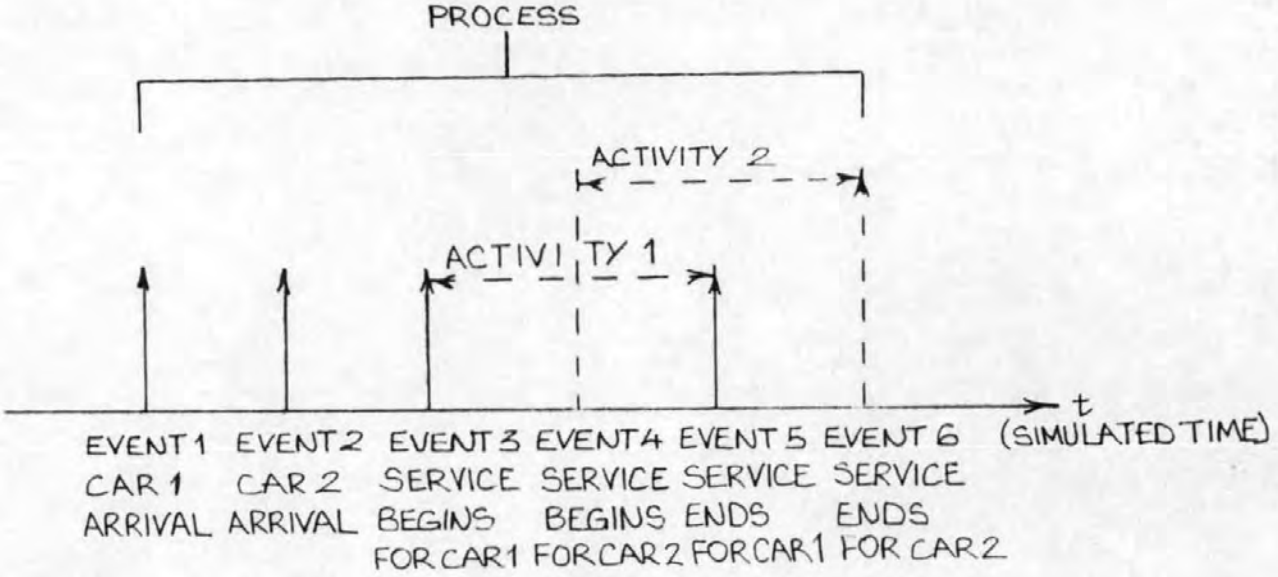
2.1.2 Models

The most general definition of a model is that it is anything that represents something else. This general definition would include such things as statues as models of particular humans, plays representing historic events,etc. A more appropriate definition is that it is a formal representation of theory or a formal account of empirical observation (FISH 73). Kiviat, on the other hand, (KIVI 67) classifies models as iconic (physical), symbolic or analogous model/. Gordon (GORD 78) investigates two general categories of models; physical and mathematical,with subsequent sub-division of the models into static or dynamic, numerical or analytical as shown in Figure (2.1). Furthermore, a model can be either deterministic or stochastic. Here, we are concerned with the class of symbolic, dynamic, numerical models implemented on a digital computer i.e. computer simulation models.

Models, including simulation models are built for a number of reasons,viz:

1) It may be more costly, dangerous, time-consuming or impractical to experiment with the actual system.

2) A real system may not be available e.g. hypothetical system.


PROCESS
ACTIVITY 2
ACTIVITY 1
t
(SIMULATED TIME)
EVENT 1 CAR 1 ARRIVAL
EVENT 2 CAR 2 ARRIVAL
EVENT 3 SERVICE BEGINS FOR CAR1
EVENT 4 SERVICE BEGINS FOR CAR 2
EVENT 5 SERVICE ENDS FOR CAR1
EVENT 6 SERVICE ENDS FOR CAR 2

FIG(2.3): RELATIONSHIP BETWEEN EVENT, ACTIVITY AND PROCESS WITH RESPECT TO CARS ARRIVING AT A FILLING STATION.

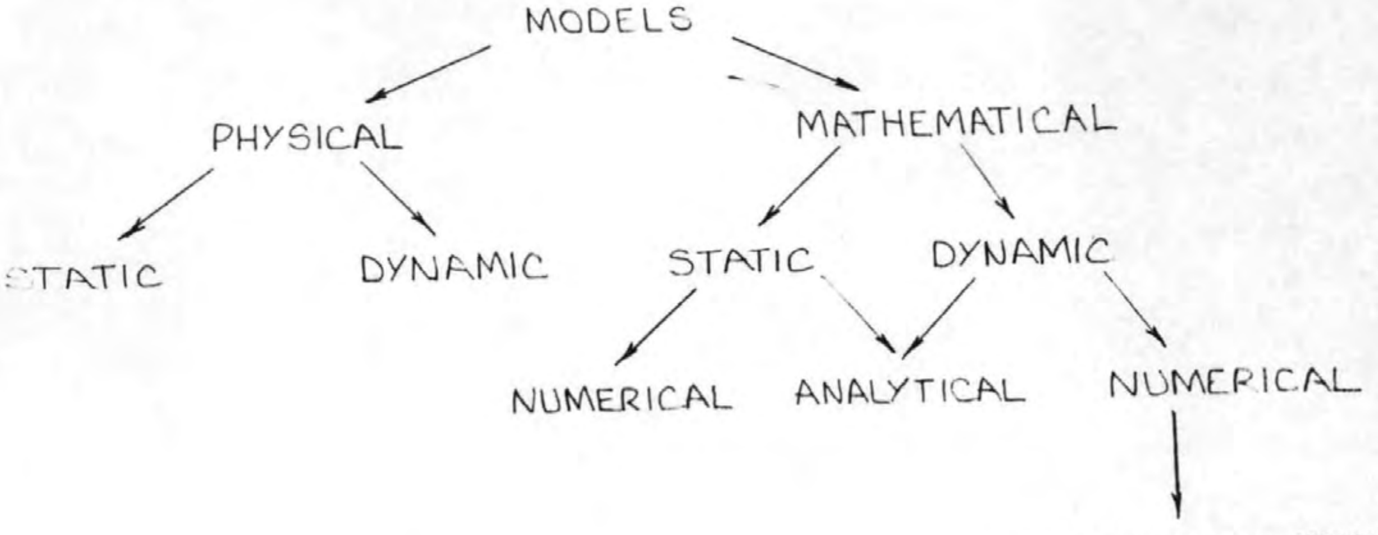


FIG (2.1): MODEL TYPES

3) It is impossible to manipulate or control variables of interest.

4) It Leads to more insight and improved understanding of the system.

5) It provides a convenient framework for testing new modifications and proposals.

The problem at hand determines to a great extent the type of model and solution adopted. Analytical solutions are more attractive than simulations, if they can achieve the objective. This is because analytical models once developed and verified give answers to a variety of input parameters, with very little additonal effort. However, except for the simplest cases, the derivation of an analytical model for a complex system proves intracticable mathematically, except through a series of simplifying assumptions which may affect the credibility of the model itself or limit its use. One such common approximation is that of statistical independence of the inter-arrival and service times, and hence the use of a Poisson input process and exponential service times. The use of this 'memoryless' exponential distribution greatly simplifies the mathematics in analytical models using networks of queues. A further simplifying assumption is that the system is stationary (has reached steady state or statistical equilibrium) thus the time derivatives vanish.

In spite of the simplifying assumptions, queuing models have been used successfully to study computerand communication systems (GRAN 64, FRAN 74, KUCH 75). They are best suited to give a general insight into the system dynamic behaviour, to identify the bottle necks and in the study of sub-systems. Sometimes, it is more advantageous to use them in conjunction with other methods such as system simulation (JOUB 78, UNGU 75).

2.1.3 System Simulation

System Simulation may be broadly defined as the act of representing a system by a symbolic model which can easily be manipulated to produce numerical results. More specifically, Krasnow (KRAS 67) defines system simulation as the activity comprising the description of a system by constructing a model, the description of an experiment to be conducted with the model, the carrying out of the experiment and the analysis of the results.

The range of system simulation models is fairly broad; they range from using a prototype of the system as a model (identity simulation) to a computer model. In this study we are concerned with computer models only. These may be classified as analogue and digital computer models. In analogue simulation models, analogue computers are used to simulate a set of differential equations modelling the behavioural characteristics of a system. Digital simulation models may be implemented in harware or in software (HART 75). Hardware simulators are composed of special-purpose computers (or equipment) and undetailed programming. They are characterised by having high speed and parallel operation. Software simulation models on the other hand, utilise detailed computer programs to model a system in a general-purpose computer. They are characterised by low speed and sequential operation. The simulated time advancement may follow either the epoch-by-epoch (or equal increment) approach or the event-by-event approach for both hardware and software simulators. In the former,the simulated time clock is incremented by a fixed amount, ΔT, every time and the system updated at the epochs where events occur, whereas in the latter,emphasis lies on updating the overall simulation only on the occurrence of an event. In this latter sense, simulation may be viewed as the activity concerned with the generation and cancellation of event notices or records, transformational rule selection and clock maintenance. Time intervals associated with events are normally drawn from appropriate statistical distributions. Due to their relative importance within the simulation methodology, the time advancement mechanisms will be considered in greater details in the next two sections. Thus,software digital simulation models are based on representational descriptions of entity interactions and state variable transformations that must be specified in the computer program (model) which,when executed,traces or mimics the dynamic behaviour of the modelled system. The program is then the realisation of the model. Furthermore, the specification of the program must be made within the confines of the abstractions or concepts supported by the computer language selected to implement the model. Thus the language choice greatly influences the way the modelled system is viewed.

The essential features of all types of computer simulation are the computers, operation rules, mathematical functions and probability distributions and it is specifically suited to systems where the relationships between the key variables cannot be expressed analytically, or where the major attributes of the system are characterised by probability distributions or stochastic processes (REIT 71). Computer simulation offers a scientific approach to system investigation. Although systems differ in their complexities and characteristics, the ingredients of this approach, namely, model building, computer science and statistical techniques are applicable in the study of any system employing this approach.

Many advantages accrue to computer simulation in comparison to other modelling techniques. It can compress time, so that many years of activity may be simulated in minutes or even seconds. This will enable a system analyst to compare long-term behavioural characteristics of alternative designs or operation rules. It can also 'dilute' or expand time. Here, the system is simulated at a finer level of detail,e.g. a time grain of a microsecond say, to enable the components interactions to be closely observed which cannot be done in real time. This research study is one such example,especially the study of the process allocator (Chapter 5). The simulation can also freely identify and control the sources of variation in the model by explicitly specifying the sources of variation and degree of variation due to each,which is not possible when experimenting with a real system. This is particularly important if the statistical analysis of the relationship between input and output factors in an experiment is to be performed.

If programmed appropriately, a computer model may be stopped to investigate the results of a run so far and re-start it again without loss of continuity. Provided that the same seeds for the psuedo-random number generators are used,the dynamic behaviour of a system can be re-produced again and again for purposes of debugging and fault diagnosis. This is difficult to achieve in a real situation. However, a price has to be paid in the form of increased computing cost and human effort. This is particularly true for detailed simulations. Another price is the necessity to apply statistical techniques to the analysis of the simulation output,since simulation is in essence a sampling experiment. In this respect, it should be emphasised that simulation is essentially an experimental technique. The immediate purpose of an experiment is to observe the behaviour of a given model within a given environment. It is also recognised from the outset that digital computers are discrete devices and that a digital simulation model is a discrete approximation to a given system. Thus continuous changes in the real system are represented by a series of discrete changes in the model i.e. events, and such a model is called a 'discrete event model'. In contrast to continuous simulation, where the system as a whole is represented by a set of differential equations, the individual events of a discrete-event model, are often specified in great detail. The apparent realism of the resulting discrete-event model accounts for some of the charm and fascination of the 'art' of discrete event simulation.

The models developed in this research study are of a discrete-event type. The occurrence of an event is represented by a change in a component's attribute value. Since the components or entities states remain constant between events, there is no need to account for this inactivity in the model. Hence, all modern computer simulation programming languages use the next-event approach to time advance; i.e. at a time corresponding to a particular event, all relevant state changes will be made. Simulated time is then advanced to the time of the next event and the above process repeated and so on. This cyclic process is depicted in Figure (2.2). In this way a simulation is able to skip over the inactive time whose passage in the real world people are forced to endure. The efficiency of this event-by-event method over the epoch-by-epoch one is also evident, particularly where the epoch is smaller than the average length of event inter-arrival time, which is usually the case.

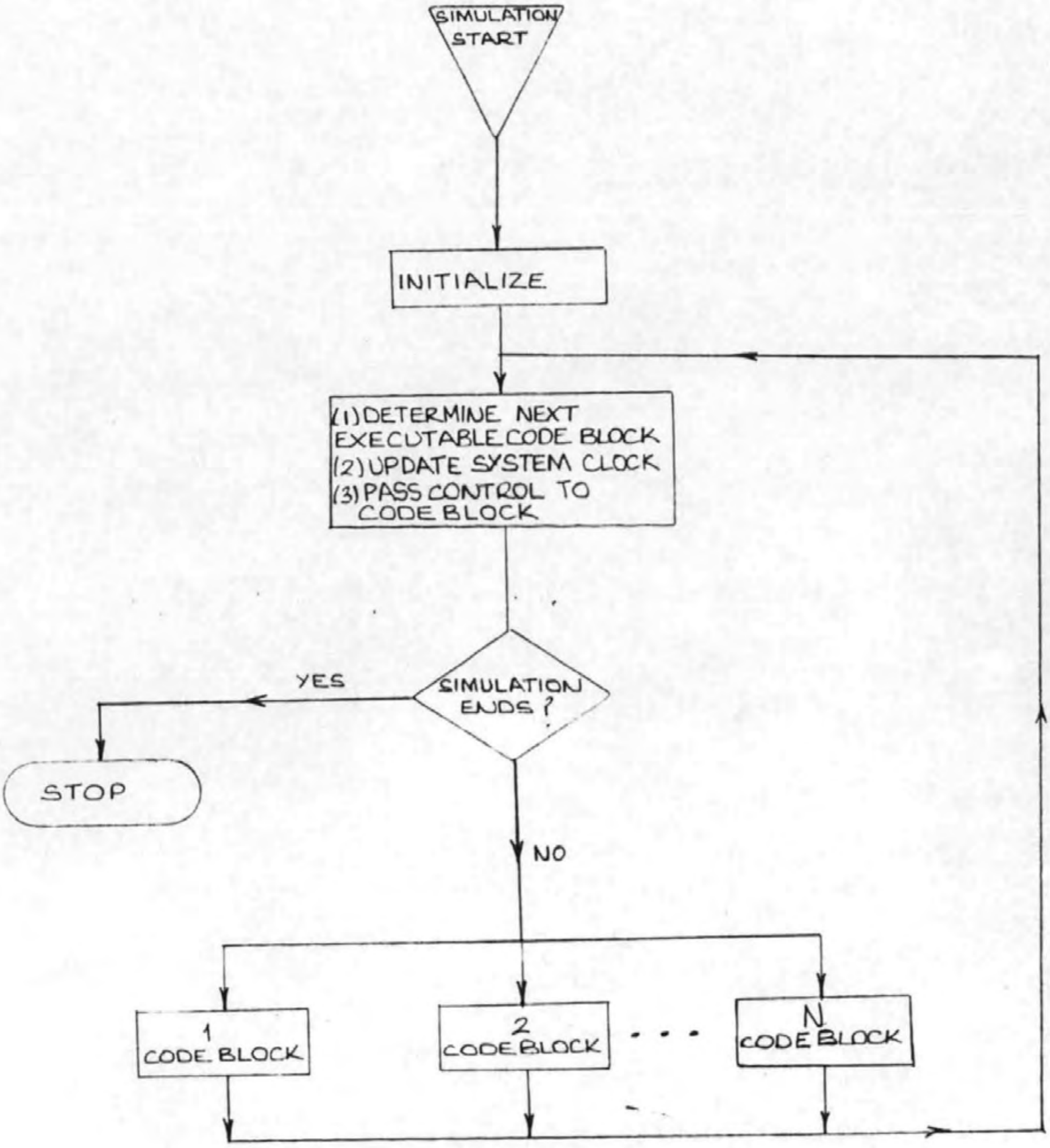


FIG (2.2) BASIC DISCRETE EVENT SIMULATION TECHNIQUE

One can identify two structures that play significant roles in discrete-event modelling. The mathematical relationships between model variables' or entities' attributes comprise one structure.e.g. if L is a queue length, then L becomes equal to L + 1 or L - 1 according to whether a customer arrives or departs. For some systems, the specification of the mathematical relationships for a system serves to describe completely the way in which state changes occur, e.g. a model of a natural economy system (FISH 73). The second structure comprises the logical relationships. Here, logical operating rules are established, logical conditions are checked and actions are taken according to the established operating rules.

The concepts of simulated time passage and relational structures are central to any discrete-event modelling system. Different modelling approaches account for these concepts in varying ways. In the following section, we will try to identify those approaches and the characteristics pertinent to each.

2.1.4 Discrete-Event Simulation Approaches

2.1.4.1 The Approaches

Discrete-event modelling approaches are built around the concepts of event, activity and process. An event, as mentioned before, signals a change in the state of an entity. An activity may be defined as the collection of actions that transform the state of an entity. It starts with an event and ends with an event. Similar concepts are used in other areas of software. One such area is the specification and description languages (KAWA 71, GALE 75).

In particular, with reference to the specification and description language adopted by CCITT (Consultative Committee of International Telephony and Telegraphy) (SDL) for SPC switching systems specification and description (KAWA 71, GALE 75, GERR 74, CCIT 77 etc.), an event is an input and an activity is a transition between two stable states in a finite-state machine. On the other hand, a process may be looked at as a sequence of events ordered in time. Figure (2.3) depicts the relationship between an event, an activity and a process. More specifically, from a simulation point of view, a process may be defined as a dynamic entity, a singularly-occurring instance of execution of a set of logically-related activities (MACD 73A). Processes comprised of like sets of activities are considered to belong to the same class.

Based on the three concepts of event, activity and process, three alternative approaches to discrete-event modelling exist (KIVI 67); namely, the event-scheduling, the activity-scanning and the process-interaction approaches. The evolution of the three approaches is related to the development of computer simulation languages. For example, SIMSCRIPT (KIVI 69A et al) and GASP (PRIT 69) are based on the event-scheduling approach (a recent version of SIMSCRIPT II.5 supports the process approach), CSL (BUXT 66) uses the activity-scanning approach and SIMULA (DAHL 70), GPSS (GORD 78) and ASPOL (MACD 73A) the process-interaction approach.

In the event scheduling approach, the events that may occur in the system are identified first. Some of the events are conditional on the occurrence of other events. Future events records are listed in time-order sequence in a calendar of events. Each record contains the future simulated time for the occurrence of the event and a reference to the code block representing that event. The actions initiated by the occurrence of an event are identified in the event flow chart. This includes scheduling of other events, in the form of filing and/or removal of event records from the calendar. Selection of the next event is cited as the last instruction. When this instruction is encountered, control passes to the simulation control program,which searches the list of filed records to find the record with the earliest scheduled time. Simulated time, kept by a monotonically-increasing global clock, is then advanced to this scheduled time and control is passed again to the code block representing the new event.

One disadvantage of this approach is that the inclusion of an event record in the calendar ordered by future time occurrence and the subsequent search for an appropriate event is a rather time consuming process,especially where the event density is high. To speed up this operation, some simulation systems sub-divide the calendar by the type of event; the actual event list scanned by the timing mechanism is only the 'best-of-show' of sub-calendars of each event. It is claimed that this event approach is more suited to single-resource and logically-simple simulations,where a one-to-one correspondence between events and activities exists (LASK 65). Single-resource activities are those requiring the availability of no more than one entity other than those already present. Here,the event approach is efficient in the sense that each activity is attempted only when it is logically possible for it to be performed.

However, in multi-resource and logically-complex situations,where the relationship between event and activity is many-one or many-many, the programming of such a situation in an event approach is rather complex.

An example of such a situation is the operation of a shipping port. Here, for example, the activity of berthing a ship will involve the presence of a pilot, tug, berth and a ship, each of which is an event in its own right. It is worthwhile noting that, in the event approach, activities are not explicitly recognised. A more convenient method for the above situation is to put into a sub-routine the programme corresponding to transforming the world-state (which constitutes the activity) and stating exactly the conditions of entry to the sub routine.

Such a sub-routine is the essence of the activity scanning approach, where each activity is composed of a test part at the beginning and a body. The emphasis here is on the review of all the activities in a simulation model to determine by performing the tests on the test parts, which can be executed each time an event occurs. In this approach, events are only implicitly recognised. To keep track of various events and advance simulated time to that of the next event, the notion of 'time-cells' is introduced. These are storage locations associated with certain entities and holding the relative time at which the entity will or has become available to participate in activities. Relative time means with reference to 'now' or relative to 'now', e.g. a positive time-cell value indicates the time interval that is to elpase from now for the event to occur, whereas a negative time-cell value indicates how long ago the event associated with the entity has occurred. Here, time advancement is more involved than just advancing a master clock to the next event, a characteristic of the next event approach. Rather, all the time-cells have to be searched to determine which of them stores the least value, then subtracting that value from all the time-cells, a process which is relatively time consuming. However, this disadvantage with respect to the event approach is off-set by the time advantage when causing an event. To cause an event T time units from now, say, will only require setting the appropriate time-cell to the value T and does not involve a search through a calendar to insert an event record appropriately. Thus, from this point of view, there is little to choose between the two approaches, except in cases where events are cancelled or have their times altered. Whereas, in the activity approach, the appropriate time-cell value only requires to be altered, the event approach would require a double search through the calendar.

Thus, one can say that the important advantage of the event approach lies in the execution efficiency it achieves by preserving the knowledge of which entities are involved in an event and using this information to ignore, in the subsequent event - activities phase, those activities whose entry tests do not include the presence of the appropriate entities. In other words, the fact that an event record contains a reference to the event-code block alleviates the need for scanning and testing all event code blocks to determine which ones should be executed, as in the activity approach. The advantage of the activity approach, on the other hand, is that the events-activities inter-relationships need not be explicitly specified, resulting in the simplicity of formulation of a multi-resource or logically-complex systems models.

To improve on the execution efficiency of the activity approach, the Disaggregated Activities List approach is proposed (LASK 65). Instead of having one activity list only, that list is sub-divided into sub-lists according to the entities whose events may result in these activities and according to activities whose execution may result in further activities. Thus, at any entry to the activities phase, the information regarding the entities involved in a particular event from which activities can now result is not lost and is used to determine which sub-lists should be entered. This alleviates the need to scan the <u>whole</u> of the activity list and hence increases the execution efficiency of the simulation system. Thus,the event-scheduling approach contrasts with the activity scanning approach in that a detailed list of scheduled events with their occurrence times is maintained, The number of events grows with the growing number of activities,thus increasing the computer time required to create event records, file them in the event list, select them for execution and destroy them once executed.

The activity-scanning approach, however, substitutes less time-consuming logical checking in the model at each time advance, for the required event scheduling steps. From the foregoing, it is evident that both of the approaches have their advantages and disadvantages. However, it is interesting to note that the event scheduling approach is the most widely use, thanks to languages such as SIMSCRIPT,which is event-based and widely available, in contrast to CSL, say, which is activity-based.

A third approach exists, namely, the process-interaction approach which is believed to combine the sophisticated event scheduling of the event-scheduling approach with the concise modelling power of the activity-scanning approach. From a simulation point of view a process is a collection of events that describes the total history of a system's component through the system. The process interaction approach is felt to be most natural in a variety of modelling situations (BIRT 73). The mode of describing actions is serial here and this is felt most natural when considering a single system component. The process approach encompasses, within a single framework, the features of duration, parallelism and interaction which characterise components of dynamic systems.

Thus, in this approach, the simulation analyst first identifies the system components and then for each component examines the system behaviour from the point of view of that component and hence develops a scenario of behaviour for that component. Hence, a component's life pattern is described by a scenario which includes the transformational rules which that component or entity applies and its possible need to request or react to application of transformational rules which are part of the scenario descriptions of other components (FRAN 77). Therefore, a system simulation program is made up of scenario descriptions of possibly several components and their interactions. This approach to a component within a model permits localisation of statistics gathering, the attributes necessary to define the state of the component at any time and the protocol for interaction with other objects in a single structural entity, thus resulting in a modularly structured simulator. The scenario descriptions are knownas processes and a model,after initialisation,is thus composed of process instances of components'scenario descriptions which co-exist in parallel and progress, in simulated time, in a piece-meal fashion through their scenarios or life patterns independently, yet interacting with other instances.

A more formal definition of a process is due to MacDougal (MACD 73A):

> "A process is a dynamic entity. A singularly occurring instance of execution of a set of logically related activities. Processes composed of like sets of activities are considered to belong to the same class. At any point in time a number of such processes may exist in a system model in varying stages of execution. Each is an instance of object of its class, uniquely identified among other members of that class by certain attributes. The behaviour of processes of the same class may be described by a single set of rules describing the activities of all processes from that class (the action statements)

together with a set of attribute(s) values for each of the existing processes of that class (activation record)."

Thus each process has its own copy of the actions and data structures (attributes) of the class to which it belongs together with a local sequence control or a local pointer which points to the statement in its copy being executed. In the process approach, time may elapse in the model, in contrast to other approaches which provide snapshots only. As processes in a model progress in a quasi-parallel or concurrent fashion, process interaction or synchronization has to be taken care of to resolve any conflicts of overlapping processes.

Processes can delay themselves for a specified period of time by using scheduling constructs such as 'DELAY' or 'HOLD'. The effect of these on the simulation control program is to suspend a process execution and file an event notice indicating the future reactivation time. The process's 'little green finger' (Local Sequence Control or LSC) then points to the reactivation point, i.e. the statement following the scheduling construct where execution will continue after the last abandonment. A process may also suspend its execution for an unspecified period of time, using scheduling constructs such as 'WAIT' where the simulation executive program removes the event notice or record associated with that process from the event list. Here, a process relies on another process to schedule an event notice for its reactivation using scheduling statement such as 'ACTIVATE PROCESS X'. Thus, for a model based on the process-interaction approach, we require processes and synchronization primitives for the manipulation of processes' event notices and hence activation and reactivation times.

Process synchronization implies a constraint on the order of process operations in time. Three types of synchronization mechanisms are required to co-ordinate and manage processes' interactions to simulate a system behaviour. Firstly, the mutual-exclusion mechanism or primitive is needed for competitive process interaction to manage the allocation of single-capacity resources exclusively to a process. An example is the competition of jobs or programs to acquire a CPU or customers to acquire a service station. When the resource is relinquished,it is allocated to one of the waiting customers according to some specified discipline. Secondly, a producer/consumer synchronization primitive is required to manage co-operative processes interactions. This is to avoid a situation where either a producer process is producing units at such a rate that the consumer process can not possibly cope ,or the consumer process is trying to consume non-existent units. Thus, a consumer process must stop and await the arrival of additional units if it runs out of units and the producer process must then re-start the consumer process after producing one or more such units and then suspend itself. Its activation is then the responsibility of the consumer process when it runs out of units. Thirdly, some processes may require the existence of a partial or complete system state before their execution may be continued.. The simulation system must allow such processes to check for the conditions of reactivation at the occurrance of appropriate events. Such a synchronization is referred to as mass retries (FRAN 77). For example, a process whose actions are suspended in a conditional wait using a scheduling construct such as 'WAITUNTIL' must be allowed to check the condition of its reactivation at the occurrence of favourable events. Some simulation systems vest this responsibilty with the simulation control program or executive at the price of increasing run time. This is because the simulation executive has to activate a monitor after the occurrence of each event. The monitor will then activate processes suspended in a 'WAITUNTIL' and filed in a conditional wait list, to test for themselves whether they can proceed or not (HILL 76, VAUC 73). Other simulation systems vest the responsibility wholly with the user *e.g.* SIMULA 67 (DAHL 70) or partly with the user who signals when a possible 'WAITUNTIL' condition occurs *e.g.* DEMOS (BIRT 79). The latter system attempts to strike a balance between run time efficiency and user convenience.

It is worthwhile noting that the concept of a process as expressed here is similar to that employed in operating systems design (HANS 73, HANS 77). Hence,a simulation system based on the process approach was felt a natural choice for simulating SPC system controlled by special-purpose computers using inter-communicating operating systems and application processes. More about [illegible] simulation language selection is given later on in the Chapter.

2.1.4.2 The Simulation Executive Program

A simulation system provides a world view to system modelling. It provides a conceptual framework for precise thinking (DAHL 68). The manner in which the framework accounts for the passage of simulated time is central to the framework since time and its representation is the essence of discrete-event simulation (KIVI 69A). This is the extra dimension to be accounted for in writing discrete-event models programs as opposed to the ordinary type of programming activities.

The heart of every simulation system is a simulation executive program. It is referred to alternatively, as a simulation-control program, a time-control program, a timing mechanism and the like. In all cases, it always performs the same functions: to advance simulated time and to select a code segment that performs a specified simulation activity. Regardless of the simulation approach, the basic structure of the simulation executive program is always the same (Figure (2.2)).

A simulation system may be thought of as a hierarchical three-level structure. At the top level is situated the simulation executive program. The intermediate level is composed of simulation-oriented routines such as events, activities and processes. The bottom level is composed of routines that perform basic housekeeping functions, such as input/output, computation of mathematical functions, generation of random variates and the like.

With reference to Figure (2.2), the number and structure of the code blocks will depend on the sequencing approach adopted i.e. event, activity or process. For the event-scheduling approach, the code blocks are known as event routines. These are the specifications of the transformational rules or activity that define or accompany each event. Each code block contains both a test at the top and action statements which are a description of the situations that can take place whenever an event occurs (KIVI 67) For the activity approach, the code blocks again consist of two parts, a preamble or test part at the top and an action part. The test part may be a complex logical test involving time-cells values and different attributes values. The test is first performed and, if positive, then the activity defined by the action part is executed. For the process approach, the code blocks are the simulation processes as defined before. The scheduling statements within a process body such as WAIT, DELAY and WAITUNTIL, are provided by the simulation system to enable the user to interact with the simulation executive program. DELAY and HOLD type of scheduling statements are referred to sometimes as imperative sequencing statements as they are explicit in stating what should happen and at what time (DAHL 68) e.g.

PROCESS p HOLD(X UNITS OF TIME).

On the other hand, it may not be possible, sometimes, to predict in advance the time at which a given event should take place. This is due to the inter-dependence of system components or processes. In such instances, the scheduling statement WAITUNTIL is used to interact with the simulation executive to take care of the process in such a situation, e.g.

PROCESS p WAINTUNTIL (EVENT S OCCURS).

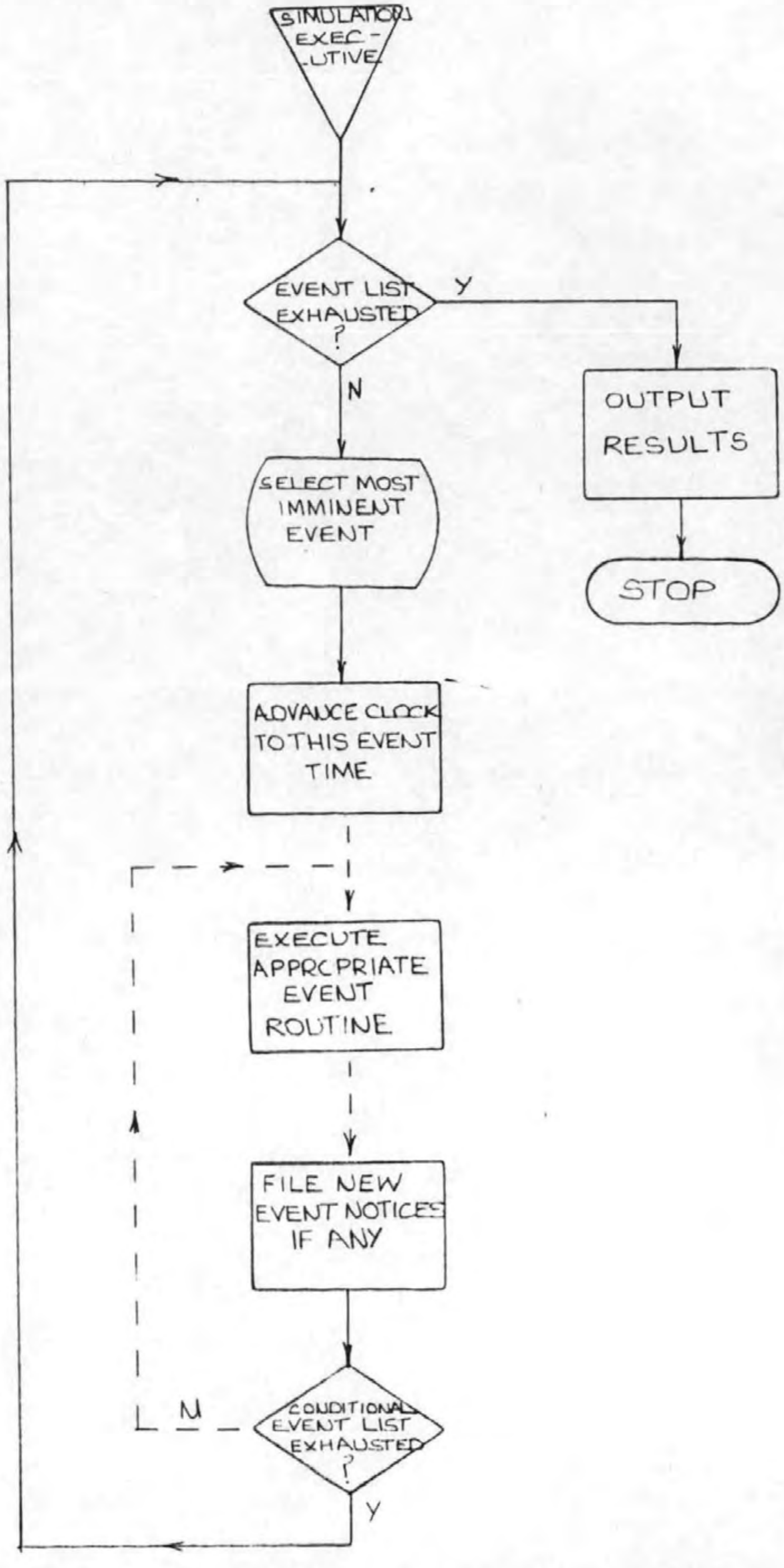
This conditional type of scheduling is referred to as interrogative scheduling.

The simulation executive programs for the event, activity and process approaches are depicted in more detail in Figures (2.4), (2.5) and (2.6) respectively. Note that the dotted lines imply transfer of control between the simulation executive program and the appropriate code block. The flow charts reflect the basic macro-level structure only, omitting any details that might otherwise obscure this basic structure. In Figure (2.4), after the execution of an event routine, its event notice is either destroyed or re-scheduled, possibly together with other new event notices. The conditional event list is then checked to see if any of the pending conditional events could be executed. In Figure (2.5), the RECYCLE flag is set only when certain activities are executed whose execution will result in the execution of some other conditional activities. The conditional event list for processes in a 'WAITUNTIL' in Figure (2.6) is scanned every time after each event. This is an over-simplification of the possible algorithms to deal with such a situation (VAUC 73). The automatic creation and destruction of event notices is implied in each of the above flowcharts of Figure (2.4) and Figure(2.6).

2.1.4.3 An Example and a Summary

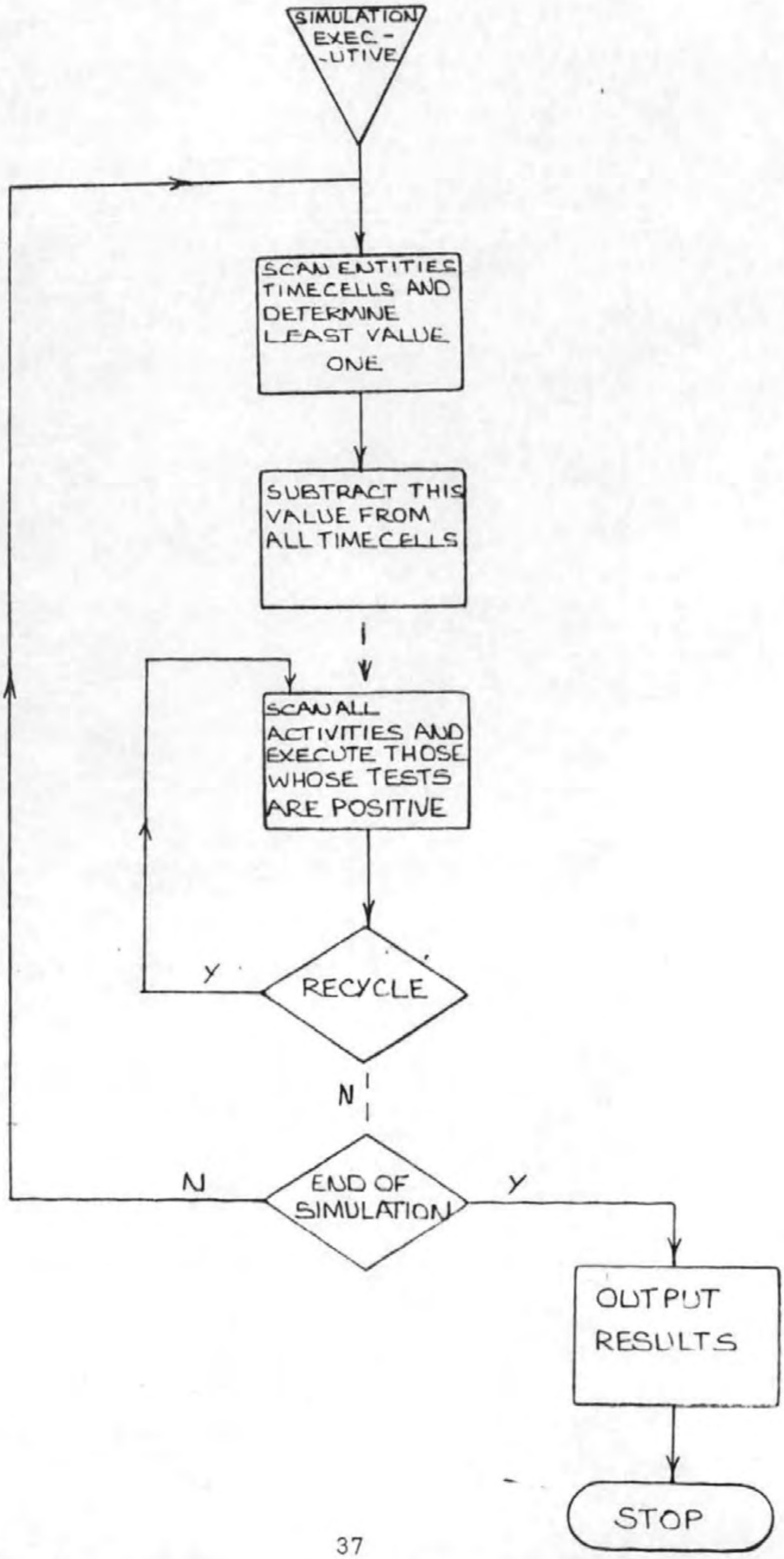
The differences between the nature of the code blocks of Figure (2.2) when expressed in different approaches can best be demonstrated by a simple example. Consider the simple case of simulating a petrol filling station with a fixed number of pumps and attendants. Cars arrive randomly and service is offered when

FIG(2.4): SIMULATION EXECUTIVE ROUTINE - EVENT APPROACH


SIMULATION EXEC-UTIVE
EVENT LIST EXHAUSTED ?
Y
N
OUTPUT RESULTS
STOP
SELECT MOST IMMINENT EVENT
ADVANCE CLOCK TO THIS EVENT TIME
EXECUTE APPROPRIATE EVENT ROUTINE
FILE NEW EVENT NOTICES IF ANY
CONDITIONAL EVENT LIST EXHAUSTED ?
N
Y

)TE: - - -> DOTTED LINE ARROWS IMPLY TRANSFER OF CONTROL BETWEEN EXECUTIVE ROUTINE AND CODE BLOCKS.

FIG(2.5): SIMULATION EXECUTIVE ROUTINE - ACTIVITY APPROACH


SIMULATION EXEC-UTIVE
SCAN ENTITIES TIMECELLS AND DETERMINE LEAST VALUE ONE
SUBTRACT THIS VALUE FROM ALL TIMECELLS
SCAN ALL ACTIVITIES AND EXECUTE THOSE WHOSE TESTS ARE POSITIVE
RECYCLE
Y
N
END OF SIMULATION
N
Y
OUTPUT RESULTS
STOP

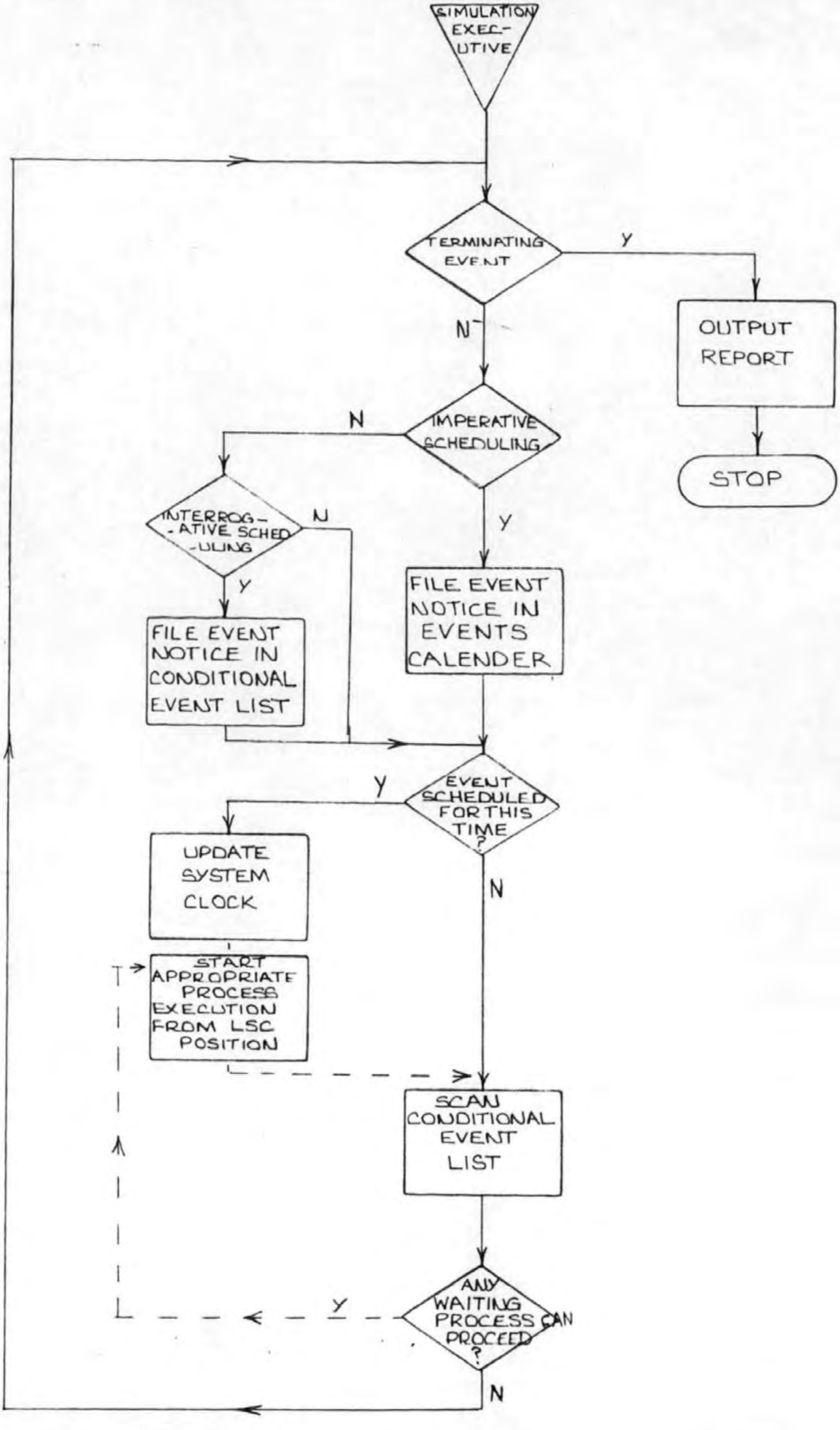

SIMULATION EXECUTIVE
TERMINATING EVENT
Y
N
OUTPUT REPORT
STOP
IMPERATIVE SCHEDULING
N
Y
INTERROG-ATIVE SCHEDULING
N
Y
FILE EVENT NOTICE IN EVENTS CALENDER
FILE EVENT NOTICE IN CONDITIONAL EVENT LIST
EVENT SCHEDULED FOR THIS TIME ?
Y
N
UPDATE SYSTEM CLOCK
START APPROPRIATE PROCESS EXECUTION FROM LSC POSITION
SCAN CONDITIONAL EVENT LIST
ANY WAITING PROCESS CAN PROCEED ?
Y
N

FIG (2.6): SIMULATION EXECUTIVE ROUTINE - PROCESS APPROACH both an attendant and a pump are idle (the station is not self-service). Simulating this situation in the event approach would require writing three code blocks; one for the attendant, one for the pump and one for the car. Each block would contain test and action statements. For example, the code block or the event rountine of the event 'attendant becomes idle' might look like:

```
     Test Part:     If a car is available AND
                    If a pump is idle then do Action 1
                    ELSE do Action 2.

     Action 1:      Engage attendant
                    Engage pump
                    determine time attendant will be engaged
                    determine time pump will be engaged
                    schedule 'attendant becomes idle' event
                    schedule 'pump becomes idle' event
                    return to executive program.

     Action 2:      Put attendant in idle state
                    return to executive program.
```

Expressed in the activity approach, we have to specify the conditions for the state of a service and the actions taken when such conditions are satisfied. One possible description of this activity is as follows:

```
     Test Part:     If a car is available AND
                    If a pump is idle AND
                    If an attendant is idle THEN DO Activity
                    Body ELSE go to next activity or if last
                    one to executive program.

 Activity Body:     Engage attendant
                    Engage pump
                    determine time attendant will be engaged
                    determine time pump will be engaged
                    Set attendant-clock to time attendant
                    will be idle
                    Set pump-clock to time pump will be idle
                    Go to next activity or if last activity
                    to simulation executive program.
```

Using the process interaction approach, the above system dynamics behaviour may be expressed by a scenario from the point of view of either the car, the attendant or the pump. For example, from the point of view of a car, the service process might look something like this:

```
Process Car:   Generate a new car after a random
               inter-arrival time.
               Acquire an attendant resource
               Acquire a pump resource
               Hold the attendant resource for appropriate period
               Hold pump resource for appropriate period
               Release attendant resource
               Release pump resource
               Quit filling station.
```

Modelling the situation in the event approach requires three event routines; one for each of the three events, pump available, attendant available and a car arrival. Using the activity approach, on the other hand, only one activity is required. However, while an event routine is only executed when its event occurs, the activity test preamble is checked with every time advance. Thus, the event approach has a higher execution efficiency and a lower storage efficiency,while the activity approach has a lower execution efficiency and a higher storage efficiency. On the other hand, the process approach combines both execution and storage efficiencies, by making use of both imperative and interrogative scheduling characteristics of event and activity approaches respectively.

However, each of the approaches has distinct advantages in certain modelling situation for a particular class of problemsand each of the three approaches can certainly do whatever the other two can do. From what has been said, it is clear that a simulation system based on the event approach gives the simulation analyst precise control over the execution of the programs. That based on the activity approach simplifies the modelling of multi-resource systems by allowing conditional statements of resource availability to be specified in one place. The process approach helps to reduce the number of 'overhead' statements a programmer has to write,since he or she can combine many event routines in one process description. Moreover, the overall flow of a system is clear, as all the logic of a system component is contained in one module rather than several.

To sum up we list characteristics of the three approaches as summarised by Hills (HILL 73A):

EVENT:

Not easy to specify in complex situations

Easy to explain once specified

Hard to modify since control statements are scattered.

Efficient in execution but consumes more storage than activity model.

Suitable for models which are:

a) Well-defined in real life

b) Event/activity relationship is 1:1 e.g. single resource

c) Where the real situation is defined in terms of events or break points.

Examples: Air-traffic control
military models and
communication systems.

ACTIVITY:

Easy to specify

Easy to explain

Control statements grouped together

Lower execution efficiency than event models.

Suitable for models which are:

a) Complex or ill-defined

b) Where objective is to study changes in control

c) Where the natural plant is thought of in terms of jobs to be done.

Examples: Workshops
Warehouses and ports.

PROCESS:

Hard to specify in some cases

Efficient and compact

Suitable when:

a) Efficiency is important

b) A well-defined simulation or previously specified model (in activity or event) exists.

c) The real system is naturally defined in terms of processes or easily identifiable objects.

Examples: Production plants
Refineries
Communication and computer systems.

So much so for the different simulation approaches.We will now turn our attention to the requirements demanded by simulation users from special-purpose simulation languages that implement those approaches and concepts and how those languages go about providing them.

2.2 DISCRETE-EVENT SIMULATION PROGRAMMING LANGUAGES

2.2.1 Introduction

Discrete-event simulation models are particularly difficult to program and debug in a high-level language such as FORTRAN or ALGOL. This is because they entail creation, filing and destruction of records, searching of lists, generation of random variates, data collection, analysis and display and model initialisation. The commonality of these and other features has prompted the development of special-purpose simulation languages to reduce the modelling and programming effort.

The first suggestion for a special programming language to aid in simulation model building is due to TOCKER and dates back to 1960 (TOCK 60). Since then, a proliferation of simulation programming languages (SPLs for short) has resulted. Recently, it has been reported that the number of discrete-event SPLs alone is as high as seventy (SIMU 79). This proliferation of SPLs stems partly from a genuine need to provide SPLs oriented towards particular classes of problems or application areas. However, for the majority of SPLs it was that the 'pride of authorship kept many going when common sense should indicate that other available languages would fulfil the needs' (KAY 72).

In any case, SPLs help to provide a conceptual framework for system components identification,together with the required notation for dynammic behaviour description and model development aids. In the following discussion, we will confine ourselves mainly to a small sub-set of SPLs from the whole universe of SPLs. This sub-set will include four of the most widely known SPLs; two originating in the USA (GPSS, SIMSCRIPT) and two in Europe (CSL,SIMULA). We will use them as a platform for our discussion of SPLs features.

Within the SPLs, some are at a higher level than others. For example, SIMULA and SIMSCRIPT are very powerful general-purpose languages that can define complex data structures, handle lists and allocate memory dynamically. Yet they contain only a relatively small number of special features directed towards simulation, *e.g.* simulated system time and imperative scheduling. On the other hand, languages such as GPSS (GORD 78) and SOL (KNUT 64) are at a higher level, because they have predefined objects (*e.g.* facilities and storages), interrogative sequencing and automatic reporting. However, SIMULA and SIMSCRIPT generate more efficient code and are more general and flexible in their application. Moreover, experience has shown that these extra 'niceties' can still be developed in the language source code to augment the language simulation facilities already existing and, hence, reduce model development costs further *e.g.* SIMON (HILL 76), DEMOS (BIRT 79).

Some SPL experts distinguish between a simulation language and a simulation system (KIVI 67). While a language provides certain concepts and a framework for model building, a simulation system is the implementation of that language on a particular computer or class of computers together with the model development aids. Others propose classifying SPLs differently *e.g.* Tocker (TOCK 65) suggests classifying SPLs according both to the approach adopted and the dominant type of entity in the language *i.e.* whether active or passive *e.g.* machine or material. According to this classification, SPL development in USA was essentially materially-based extending FORTRAN *e.g.* SIMSCRIPT, GPSS, GASP *etc.* and Europe machine-based extending ALGOL *e.g.* SIMULA.
In our discussion, however, we will stick to the classification of SPLs by the approach to modelling. This classification is depicted for various SPLs in Table (2.1).

Event Oriented Languages	Activity Oriented Languages	Process Oriented Languages
GASP	CSL	SIMULA
SIMSCRIPT	ECSL	GPSS
TELESIM	SILLY	SOL
SIMPAC	AS	ASPOL

TABLE (2.1) : Categorisation of Some of the Well-known SPLs According to Approach

Type of Object Class	SIMULA	CSL	SIMSCRIPT	GPSS
Predefined Permanent	no	no	no	yes
User defined Permanent	yes	yes	yes	no
Predefined Transient	(no)	(no)	no	yes
User defined Transient	yes	no	yes	no

TABLE (2.2): Types of Object Classes Available in Some SPLs

REF: (DAHL 68)

The functions that SPLs are expected to provide are as follows:

a) Static Representation
b) Dynamic Representation
c) Simulation Support

In the following, we will turn our attention to each of these functions in greater detail.

2.2.2 Static Representation

Static representation is concerned with the status of a system at a given point in time. A system is thought of in terms of the objects it contains. The system status is determined by the status of each object and their inter-relationships. An SPL has to provide the tools for the definition of the classes of objects within a system. The class concept simplifies to a great extent the description of a system, because it limits the great number of objects possible in a system to a handful of classes of objects. The definition of a class serves as a template description for all objects belonging to that class.

Objects are represented as data structures of different classes. These are records of the objects attributes. A simulation operates on these records as simulated time progresses. An SPL has to enable the definition of attributes that can both describe and differentiate between objects of the same class. Some of these attributes are system defined, *i.e.* defined by the simulation system at the time of object creation. The system-defined attributes are either hidden (not accessible to the user) or protected (only their values can be accessed). The remainder of object attributes are user-defined, and these can assume numerical or logical values, *eg* by assignment. The collective values of an object attributes determine the status of that object. For example, in a port simulation, we may identify the classes of objects as SHIP (with attributes SIZE, TIME OF ARRIVAL, CARGO *etc.*), CRANE (with attributes CAPACITY, POSITION, IDLE), BERTH (with attribute VACANT) *etc.* All ships belong to the class SHIP and individual ships are identified by different values of attributes.

Objects can either be temporary or permanent and an SPL is expected to provide for the generation of both types. In terms of job-shop models, which are essentially networks of inter-connected elements of M/G/1 queues (M/G/1 indicates Markovian or stochastic arrival, General service time distribution and one server), transient objects are the units of flow *e.g.* a customer in a supermarket or a car in a filling station. They are generated, pass through the system and fade away. When a transient object 'dies' or fades away; the storage allocated to it is reclaimed and allocated dynamically for the generation of a new transient object. Permanent objects, on the other hand, are fixed in their number throughout a simulation run. Mostly they represent the resources in a model which give service to or are controlled by transient objects. Such resources are either time-shared, *e.g.* a CPU, a machine *etc.* or space-shared *e.g.* a store. They are acquired by statements such as COOPT SEIZE and ENTER and relinquished by statements such as RETURN and LEAVE e.g. GPSS (GORD 78), SIMON (HILL 76).

In GPSS: a transient object known as a transaction, is generated by the statement:

GENERATE <List of Arguments>

with specified priority and user-defined attributes.

In SIMSCRIPT: CREATE < class >'CALLED < variable >
an object of 'class'is created and referenced by the reference variable < variable > whose value is the object being referenced
The statement

DESTROY < class > CALLED < variable >

destroys the transient object referenced by < variable >.

In SIMULA: PROCESS CLASS < class > (< formal parameter list >)
NEW < class > (< actual parameter list >)

generates a process instance of class name < class > and the value of the expression is a reference to the process instance generated. This reference value can be stored in a number of ways *viz:*

a) by assignment to a reference variable *e.g.*

```
REF(CAR)A;          A:- NEW CAR
```

b) by entering a process instance into a queue *e.g.*

```
NEW CAR. INTO(WAITQ)
```

c) by scheduling an event for the process instance *e.g.*

```
ACTIVATE NEW CAR
```

NO delete or destroy statement is used in SIMULA. A process remains in the system as long as it is referenced. A'reference count' which is a hidden attribute of a process is updated each time a reference to a process is stored or deleted. A process instance is automatically deleted when its reference count becomes zero. If a referenced process instance terminates, *i.e.* exhausts its actions and passes through the last END statement, it will still remain in the system, but only as a data structure whose attributes are accessible.

In CSL: Only a fixed number of objects is allowed to be generated and remain throughout the simulation run, *i.e.* only static memory allocation *e.g.*

```
CLASS  CAR.100
```

will define and create 100 objects of class CAR. Table (2.2) shows the types of object classes available in some SPLs and Table (2.3) summarises object generation and related concepts.

Aside from object generation, an SPL must enable relating objects to one another and to their common environment. This is achieved through relational mechanisms, such as sets, queues and lists. For example, in CSL we may define a set called READY to contaim all ships ready to depart from the port. Selection of a particular ship for departure (*e.g.* of particular tonnage) is achieved by searching through the set. As a matter of fact, list processing is a dominant feature in SPLs due to the need for flexibility and efficiency in data manipulation and search.

CONCEPT	SIMULA	CSL.	SIMSCRIPT	GPSS
Relational Concept	HEAD	SET	SET	User Chain Group
Example	REF(HEAD)Q MAN.INTO(Q) (Man Queued in (Q)	SET OCEAN SHIP 1 INTO OCEAN (Ship 1 moved to Set Ocean)	FILE MAN FIRST IN SET(I) (man inserted into set(I) as first mem.)	LINK I, FIFO (Current Transaction Put first in chain I)
Object Generation	Generate a new process when required (Dynamic)	No Dynamic Object Generation (Static)	Generate a new entity whenever required (Dynamic)	Generate a new transaction when ever required (Dynamic)
Example	QE2:- New Ship	-	CREATE A DOG CALLED SNOOPY	GENERATE 6,2

TABLE (2.3) : Relational Concept and Object Generation Method

Scheduling Type	SIMULA	CSL	SIMSCRIPT	GPSS
Imperative	YES	NO	YES	YES
Interrogative	NO	YES	NO	YES

TABLE(2.4) : Scheduling Properties of SIMULA, CSL, SIMSCRIPT & GPSS

2.2.3 Dynamic Representation

This is concerned with the system changes of state as a function of simulated time. The concept of simulated time and its passage with the consequent changes in system state is implemented through a simulation executive program, provided by the simulation system. The structures of the executive programs for the three simulation approaches have been discussed in (2.1.4.2). It is interesting to note here, that as far as the simulation activity is concerned, three different time concepts may be identified. These are:

a) Real time, in which the actual system operates.
b) System time or simulated time which is the representation of real time within the simulation model.
c) Computer time, the time taken by the computer to run the model.

It is also interesting to note the following relationships between the above mentioned time concepts:

1) An activity in the model that results in a change of state consumes computer time while taking place instantaneously in simulated time.

2) An activity that consumes simulated time *e.g.* the scheduling statement WAIT(X UNITS OF TIME),is instantaneous in computer time.

In CSL and SIMSCRIPT, the dynamic structure is given entirely in terms of events, the events being implicit in CSL. The events are associated with objects by operating on or making reference to the attributes of the objects. In GPSS and SIMULA, the concept of scheduling statements, *e.g.* WAIT, WAITUNTIL *etc.*, representing the passage of simulated time within a transaction or process respectively, permits the stringing together of events that take place at different points in simulated time. GPSS allows both imperative and interrogative scheduling *e.g.* ADVANCE < arguments > . Here, the transaction executing the statement is scheduled ΔT time units later, say, ΔT being determined by the arguments values and the transaction is suspended for ΔT units of time. The interrogative scheduling is implied in statements such as SEIZE and ENTER for acquiring system resources. When the resources are not available, the transactions are queued. GPSS statements are executed interpretively and correspond to powerful sub-routines.

In SIMSCRIPT,where the three building blocks are entities, attributes and sets, only imperative scheduling is allowed, using statements such as CAUSE, SCHEDULE and RESCHEDULE *e.g.*

SCHEDULE AN ARRIVAL IN 5 MINUTES.

The statement CANCEL P CALLED X

removes the event notice of object X of class P from the event list.

SIMULA offers imperative scheduling only, though the range of scheduling statements is comprehensive.*e.g.*

$$\left\{\begin{array}{c}\text{ACTIVATE}\\ \text{REACTIVATE}\end{array}\right\} \quad \text{X} \quad \left\{\begin{array}{c}\text{AT}\\ \text{DELAY}\end{array}\right\} \quad \text{T} \quad \left[\text{PRIOR}\right]$$

is for direct activation of process X either at simulated time, T (AT) or at TIME + T, where TIME is the current simulated time (DELAY). If the option PRIOR is used; the event notice is inserted in the event list in front of any other event notices of processes scheduled for the same system time. The statement

$$\left\{\begin{array}{c}\text{ACTIVATE}\\ \text{REACTIVATE}\end{array}\right\} \quad \text{X} \quad \left\{\begin{array}{c}\text{BEFORE}\\ \text{AFTER}\end{array}\right\} \quad \text{Y}$$

is for relative activation of process X with respect to process Y. The statement HOLD(T) suspends the running process for T units of simulated time and CANCEL(X) removes the event notice of process X from the event list if it is scheduled, otherwise the statement has no effect. Every event notice has the following attributes: SUCC and PREDE to reference the successor and predecessor of the event notice respectively, EVTIME, the time of next activation of a process and PROC, a reference to the process to whom the event notice belongs.

In CSL,sequencing may be considered as interrogative only,since time-cells values may enter into complex logical tests before the execution of an entity. The user interacts with the simulation executive program through assigning numerical values to the time cells. The scheduling characteristics of the four SPLs are summarised in Table (2.4).

2.2.4 Simulation Support

A simulation exercise requires a number of supporting facilities during the development and production runs of a model. One such facility is the generation of stochastic variates from different statistical distributions. Data collection, analysis and display is another important aspect. GPSS automatically performs data collection and estimation and then prints summary statistics at the end of a run. In SIMULA and SIMSCRIPT, the user has to program the data collection analysis and display, though SIMSCRIPT provides a convenient way of computing sums, means, variances, standard deviations etc. SIMULA provides the system procedures HISTO and ACCUM for histograms and weighted queue lengths. DEMOS (BIRT 79 and SIMON (HILL 76), both written in SIMULA source code, provide automatic data collection, analysis and display. CSL provides histogram compilation and printing.

SPLs are expected to assist in the monitoring and debugging of a simulation model. This includes trapping of syntatic and semantic erros at compile time rather than run -time and this reflects the level of security in an SPL. In SIMULA this level of security is high. For example, referencing is checked at compile time and so is the fact that reference expressions evaluate reference values. Remote accessing is also checked at compile time and a user is forced to qualify his references. On the other hand, the security level of SIMSCRIPT is low (DAHL 68); attribute references are not checked. For example, if a reference X to an object is in error, then the effect of assigning to an attribute A(X) is unpredictable, as is the use of DESTROY when X does not reference an existing object. In CSL, attribute references are easily checked, since object references are ordinary numbers in a known list or set. Monitoring and debugging are also assisted by the availability of a tracing facility that echoes the system dynamics as it evolves through time and displays the contents of tables, counters, queues etc. Some SPLs possess tracing facilities of varying strength, while others leave it for the users to provide their own e.g. SIMULA.

2.3 CRITERIA FOR SIMULATION PROGRAMMING LANGUAGE (SPL) SELECTION

Some 140 simulation languages and packages have been written (SIMU 79) since Tocker et al's pioneering paper (TOCK 60). Out of those, 70 are discrete-event languages, 40 are continuous and 30 are combined (discrete-continuous). Continuous simulation languages are targetted at representing a model by a set of differential equations and solving these equations either by using the circuitry of an analogue computer or by numerical computation. Combined discrete/continuous simulation languages are a recent development and are aimed at modelling systems (such as a chemical plant) where, for example, differential equations govern the reaction rates, while the events of switching the reactors on and off are provided discretely, (CUNN 76).

Selection of a particular simulation language from this ocean of languages proved to be a frustrating and formidable job. The factors that influence the selection process are many and diversified. Unfortunately, there exists no definite set of rules for this selection process.

There are few simulation languages reviews in the literature. They do help in giving a flavour of some of the most commonly-known languages through contrasting their approaches, facilities, capabilities etc. e.g. (DAHL 68, TOCK 64, KIVI 69A,KAY 72, TEIK 67). Some authors did provide a quantitative comparison of a few of the well-known languages, though the factors taken into account were not comprehensive.e.g. Tognetti et al (TOGN 72) reported a quantitative comparison of SIMSCRIPT II and SIMULA 67 using a single-server queuing program as an example and reporting on aspects such as documentation, capabilities and efficiency. The conclusion was that SIMSCRIPT II, is better documented and easier to learn while SIMULA 67 is more capable and efficient. Virjo (VIRJ 72) has reported a similar comparison for GPSS, SIMULA and SIMSCRIPT, concluding that SIMULA is the most capable of the three. Hills (HILL 73A)compared the activity, event and process approaches to simulation,using an example of a steel mill written in SIMULA,and concluded that the process approach is the most compact and efficient. But the most comprehensive survey of users' views on simulation languages was conducted by Kleine (KLEI 70, SHAN 73, KLEI 71). A questionnaire was distributed among simulation users in the USA and 103 responded. Four questions were posed in the questionnaire. These were familiarity and experience with the language, preference, evaluation of ease-of-use and evaluation of capability. The languages considered were GPSS, SIMULA, SIMSCRIPT 1.5, SIMCRIPT II, GASP, PROGRAMMING BY QUESTIONNAIRE, FORTRAN, PL/1 and APL. The survey was by no means comprehensive. The sample size was too small to warrant valid statistical inferences, Kliene himself commented that "One can only conclude that one should try to conclude very little about opinion surveys" (KLEI 71).

However, the general consensus was that simulation languages were preferred to general-purpose languages as far as capability and ease of use are concerned expecially SIMSCRIPT, GPSS and SIMULA. As for experience and preference, the balance was very much in favour of SIMSCRIPT, GPSS and FORTRAN. One might wonder, as did Palme (PALM 71), as to the outcome of an identical survey had it been conducted in a different country, say Sweden, where the dominant language is SIMULA. A more comprehensive study is now underway at the centre in Simulation, University of Lancaster, U.K. Theobjective is to provide a comprehensive easy-to-use document on simulation languages, their properties and criteria of choice (ELLI 78). If a survey is conducted, one would hope that the sample would include simulation users from USA, GB and Europe where the three approaches event, activity and process (and hence the languages based upon them) each have a strong hold.

As stated before, many factors effect the decision as to which language to adopt for a particular application. - Judging by our own experience, the factors most relevant are the following:

(1) The characteristics of the system to be simulated
(2) The specifications of the resulting model
(3) The portability of the model programs
(4) Simulation Programming Languages available on site.
(5) Cost of installation, learning and maintenance of a new simulation language.

The first factor determines the approach best suited for the problem at hand. The second factor determines the extent of the capability and flexibility required of the language adopting that approach in order to fulfil the stated goals of the simulation study. Here is included the modelling power, monitoring and debugging facilities, automatic reporting,etc. The other three factors are self-explanatory.

The specification, of the simulation package to be developed for the telecommunication-oriented, multi-processor system (Mark II BL) is summarised in Section (5.2). The portability of the model programs between Plymouth Polytechnic where the research was conducted, and GEC (Telecommunications) Ltd., Coventry, where the system is being designed and built was of prime importance. So was the cost incurred if a new simulation language was to be installed at Plymouth Polytechnic, Computer Centre. It was felt that a base language (i.e. FORTRAN) was available at both sites and hence a simulation language based on FORTRAN would resolve the portability problem. CSL is available at the Polytechnic Computer Centre. It is FORTRAN-based and uses a translator program to translate CSL statements into FORTRAN. It is capable of handling complex logical decision making. It was developed in this country and is hence widely used together with its successor ECSL. So it was decided to use CSL to build the simulation model. The decision was thus heavily influenced by the availability, portability and cost factors.

The telecommunication-oriented multiprocessor system and its telephony environment was modelled by being abstracted in a number of activities sub-programs and sub-routines in CSL. The model was developed and ran successfully and a number of results were obtained. The model program with its flow charts are included in Appendix (A) A number of comments on the CSL model are appropriate here:

1) Mark II BL system is highly modular in nature-both in hardware and software. The software development engineers developing the system use a block-structured language, CORAL 66, in developing the operating system and application software.

2) CSL proved to be restrictive in its modelling power for this kind of application where dynamic memory allocation *i.e.* generation of transient objects, parallelism and the ability to construct hierarchically - structured models are of prime importance.

3) It was rather difficult to explain the model to the software engineers who are the ultimate users as an aid in their software development. This is because the system dynamics were fragmented into a number of activities with no obvious correspondence between the activities and the system software and hardware modules.

Thus if the simulation package is to be easily usable by software design engineers with little or no knowledge of simulation techniques, the first two factors in SPL selection criteria (i.e. characteristics of system and model) should have the over-riding consideration over the other remaining three. The search for an alternative SPL in the light of these new considerations continued. At one time, the use of a program generator package, DRAFT (MACH 74) was proposed. This was rejected as being unsuitable for a problem of this size and complexity in a discussion with its author. Opinions of other experts in the field were sought and the general view was that SIMULA is mostsuitable for this particular application. Some featuresof SIMULA have already been discussed. We will now divert our attention to the characteristics of the language that influenced its choice.

2.4 SIMULA 67

SIMULA 67 is the successor to SIMULA (DAHL 66). It has ALGOL as a sub-set because its basic structure lends itself to extension (NAUR 63); i.e. it is ALGOL +. It extends the ALGOL block concept to enable the generation and naming of blocks that can exist as coroutines. These represent objects in a model generated from a template of class or process declaration. The language was designed by Dahl, Myhrhang and Nyguard at the Norwegian Computing Centre, Oslo, Norway. The language was developed as an all-purpose kernel where application packages for specific problem areas can be built. This was thought an appropriate solution to the problem of proliferation of SPLs. However, the realisation of model programs using the process approach was the major impetus (FRAN 77), first used by Knuth in the design of SOL language (KNUT 64).

The lanaguage is available on most computers e.g. IBM, ICL, CDC, BUROUGHS, DEC10, UNIVAC etc., and portability of SIMULA programs is maintained by the implementers of SIMULA systems on different computers adhering to a common standard set by the SIMULA standards group in the SIMULA Common Base Language (DAHL 70).

The main features of the language are modularity, parallelism and hierarchical model structure. The language, thus, provides mechanisms to:

a) Decompose a system into natural, easily conceived components and specify process descriptions for the components consisting of complex data structures and action parts.

b) Generate and name a variable number of instances of those processes which co-exist, interact and progress in parallel in simulated time. Processes interact through interaction mechanismssuch as object references, remote accessing, quasi-parallel operation and block and class concatenation.

c) Concatinate class declarations to other classes and blocks to realise a hierarchical model structure.

d) Provide a powerful list processing and sequencing capabilities to relate and operate a collection of classes and processes.

e) Reduce debugging effort by providing 'reference security' where invalid data referencing is spotted at compile time.

The features (a) to (e) are the very ones required to build a flexible hierarchical model structure where a hardware or software module in the real system is mapped in a one-to-one correspondence to its model module. The superior capability of the language compared to other SPLs is beyond doubt (HILL 73A,TOGN 72, KLEI 71). The language itself is more difficult to learn and use than other SPLs e.g. SIMSCRIPT, GPSS, but the effort put in that direction is worthwhile.

SIMULA 67, itself contains sufficient primitivesto simulate any model. However, it does not contain all the facilities or 'niceties' one expects from a whole-hearted SPL such as automatic reporting, process tracing etc. But being an application-oriented language, a few simulation wizards can develop these features and others in packages intended for different application areas (HILL 76, BIRT 79, LOKE 74, CUNN 76, VAUC 71 etc.). A number of packages now exist in areas as varied as simulation, computer graphics, data base management, communication networks, etc. In simulation, one such package, SIMON, (HILL 76) is being used extensively in industrial simulation and teaching of simulation courses and operational research.

DEMOS, which appeared recently (BIRT 79) includes new pre-defined resource entities and a rather efficient 'WAITUNTIL' algorithm. These packages are very powerful, easy to learn and use, thanks to the concatenation feature of SIMULA.

From the point of view of simulating the Mark II BL multiprocessor system and other System X sub-systems, SIMULA provided those characteristics which solve many of the problems experienced in the earlier experiments with CSL, namely:

1) The inability of CSL to produce a modularly-structured simulator, where hardware components and software processes can be modelled in a one-to-one transformation process.

2) CSL does not lend itself to the modelling of transient objects, nor does it facilitate the modelling of parallelism in a system with complex interactions between the parallel processes.

3) The difficulty experience by the software engineers in understanding and using the simulator. This was mainly due to the lack of resemblance between the simulator and the real system.

CHAPTER 3

3.1 THE TELECOMMUNICATION NETWORK

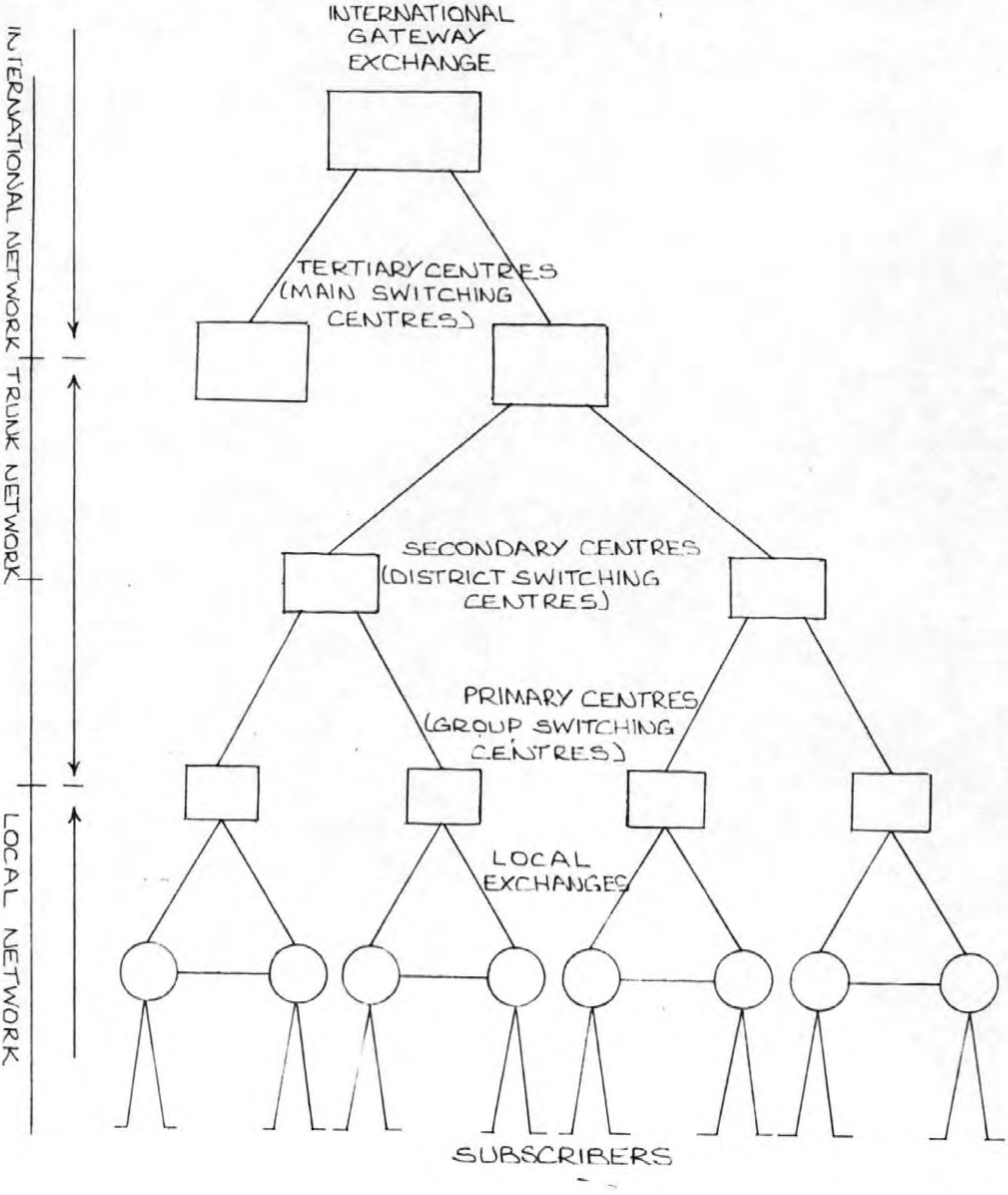
The Telecommunication network is by far the biggest man-made system in the world, interconnecting some 400 million telephones, sustaining millions of independent conversations simultaneously and in an ever-varying pattern. The network is growing at an average rate of 7% and providing more and more varied services e.g. telephone, telex, data transmission, viewdata etc.and the annual expenditure in the 'Western Block' alone amounts to 2.5 billion pounds (TIPP 77).
This monstrous man-made system was born just over a hundred years ago with the invention of the telephone by Alexander Graham Bell in 1876. The word 'telephone' itself was first used in 1796 for a purely acoustic method of communication and later for telegraph systems where the received electrical signal generates a sound heard and interpreted by the operator (FLOO 76). As the number of subscribers grew, it became clear that it was impractical and uneconomical for each subscriber to have links to every other subscriber. The solution adopted was to develop routing or switching equipment at one central location to which all the subscribers were star-connected. Thus in 1878, the first public telephone exchange came into being interconnecting [illegible] subscribers.

However, the telephone service continued to grow, and the number of subscribers became inconveniently large to be handled by one local exchange and thus a number of local exchanges were built to deal with each community. Moreover, subscribers of one local exchange required to establish conversations with subscribers of another local exchange and so the local exchanges were either mesh connected or star-connected through a 'trunk' exchange, thus a network hierarchy of exchanges resulted.

The U.K. network hierarchy is shown in Figure (3.1). The routes or circuits between local exchanges are called junctions while circuits to higher level exchanges are called trunks. Due to economical considerations, the network is not entirely star-connected and hence interconnections between exchanges in the same level do exist. In the U.K. communication network, there exists about 6200 local exchanges, 370 primary trunk exchanges (Group Switching Centres), 27 secondary trunk exchanges and 9 tertiary trunk exchanges interconnecting some 25 million telephones (HARR 79). This network is interconnected to the global network via international exchanges.

In the international network, transmission is via submarine cables, microwave radio and communication satellites. High-frequency radio transmission (HF) is still used in certain areas where the traffic is low. The transmission quality is poorer as it is affected by fading due to the changing nature of the ionosphere.

The total number of telephones in a network is a fair indication of the size of the network, given that the quality of service is comparable. Table (3.1) summarizes the telephone statistics of the 12 countries with the highest number of telephones. One noticable thing is the number of telephones in the United States which roughly equals those in the rest of the world.


INTERNATIONAL GATEWAY EXCHANGE
TERTIARY CENTRES (MAIN SWITCHING CENTRES)
SECONDARY CENTRES (DISTRICT SWITCHING CENTRES)
PRIMARY CENTRES (GROUP SWITCHING CENTRES)
LOCAL EXCHANGES
SUBSCRIBERS
INTERNATIONAL NETWORK
TRUNK NETWORK
LOCAL NETWORK

FIG(3.1) UK NETWORK HIERARCHY

COUNTRY	NUMBER OF TELEPHONES $X10^6$	PERCENTAGE INCREASE FROM 1967	TELEPHONES PER 100 OF POPULATION
USA	155	57	72
SWEDEN	6	51	69
CANADA	14.8	75	60
JAPAN	48	203	43
AUSTRALIA	6	85	40
UK	22	94	39
NETHERLANDS	5	115	39
WEST GERMANY	21	122	34
FRANCE	16	137	29
ITALY	15	136	27
SPAIN	9	180	24
USSR	18	114	7

TABLE (3.1) : Telephone Statistics of 12 Countries with Highest Telephones/100 of Population

(WORLD TELEPHONE STATISTICS, 1977)

The highest number of telephones per 100 of population are found in the USA and Sweden whereas the fastest growing network is that of Japan.

The public telephone network can conveniently be considered to comprise of three basic building blocks (LEAK 77); terminal equipment to match the user's or subscriber's mode of communication with that of the network,transmission equipment to convey the information from the sender to the recipient and routing or switching equipment to enable the sender to select the recepient of his choice.

A public network has also some important parameters whose consideration are paramount in the planning of the network. These parameters may be identified as the numbering plan, the routing plan, the congestion standards, the transmission standards, the charging plan and the signalling plan (FLOO 75).

From the foregoing, it is evident that the international telecommunication network is a vast and complex entity interconnecting some 400 million telephones, and the number is doubling approximately every ten years. With the introduction of high capacity submarine cables and satellite systems, international links are by far the greatest growth area. Although the international network is used for other services such as telex, fascimile, television *etc.*, the telephone service still constitutes the highest proportion of traffic carried.

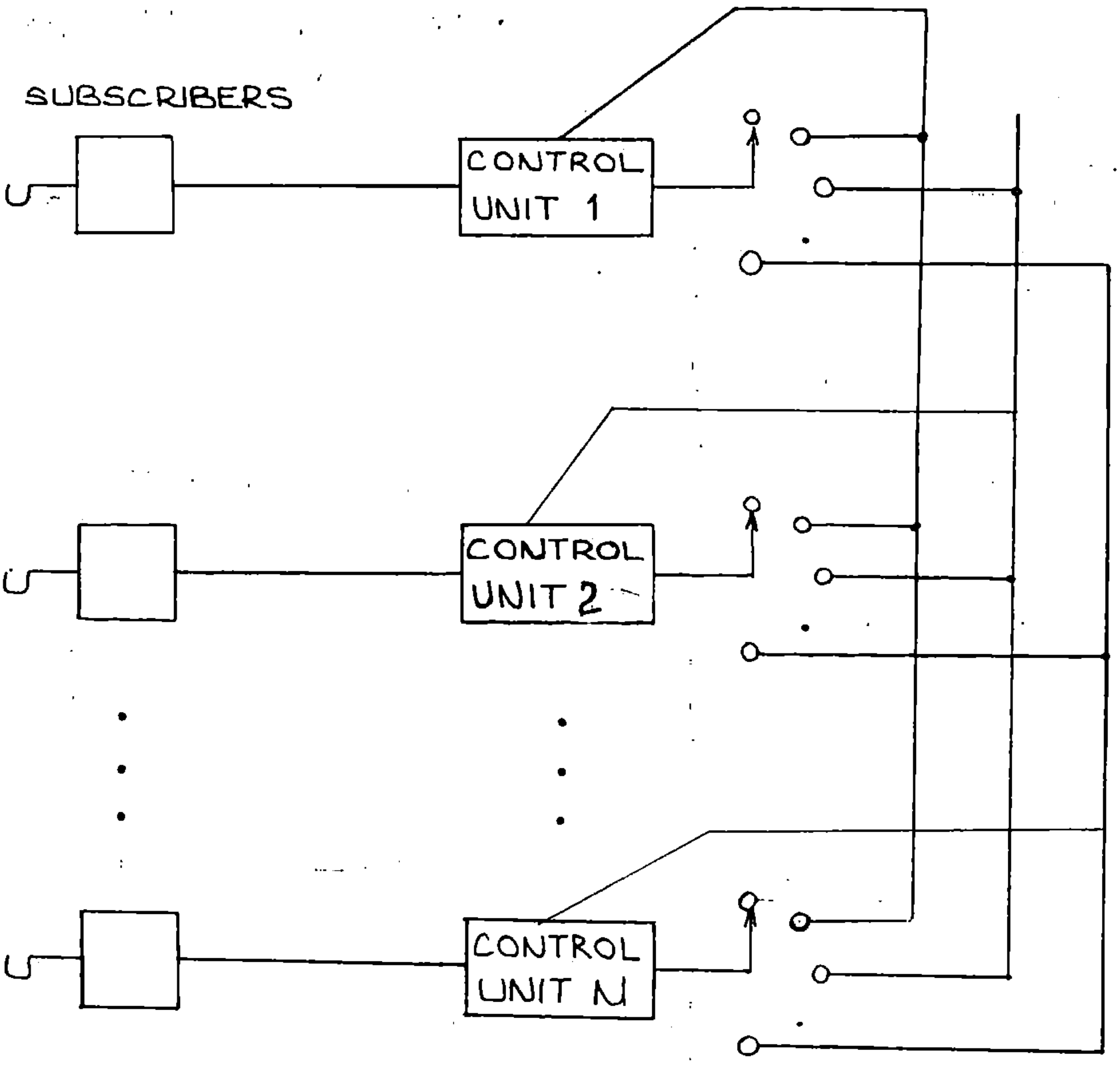
3.2. TYPES OF TELEPHONE EXCHANGES

3.2.1 Introduction

A varied number of telephone exchange systems do exist in the world today, though this variation is fairly transparent to a user of the basic telephone service. Basically, these variations have arisen due to the need for capital and maintenance economy in exchange and external plants, resulting in a number of different basic systems being introduced by different telecommunications manufacturing companies.

Until the 50's the starting point of an exchange organisation and design was usually a switch mechanism proposed to solve a particular problem at the time. For example, the Strowger switch function was to dispense with mannual operators for economical reasons and give customers remote control of their connections by sending electrical signals (dialling a number). Likewise, the crossbar switch solved the reliability problem in remote small exchanges before its potential was realised in larger exchanges. However, in newer systems, the designers emphasis has shifted to other areas such as the concentration and enhancement of the processing power of the exchange.

Basically, an exchange consists of a control unit and switches multiplied together and provided on a per-line basis as shown in Figure (3.2). The control sub-system has a number of important functions. It has to receive the routing instructions from the subscriber in a form of a directory number, to translate that number, establish, supervise and terminate a call. Other subsidiary functions include call charging, equipment monitoring and fault avoidance.



FIG(3.2): A BASIC SWITCHING MACHINE

The system depicted in Figure (3.2) will be prohibitively expensive for a large number of customers. To make such a system economically viable, extra stages in the switch network have to be introduced to reduce the number of crosspoints, together with resource sharing of the control sub-system. Thus, practical systems employ resource sharing techniques such as the use of a common switch network and control, functional sub-division and time-sharing decision making (HILL 76A).

The different fundamental exchange organisations can be classified under two categories:

a) the manual exchanges featuring manual control

and

b) automatice exchanges - direct control, register control and common control.

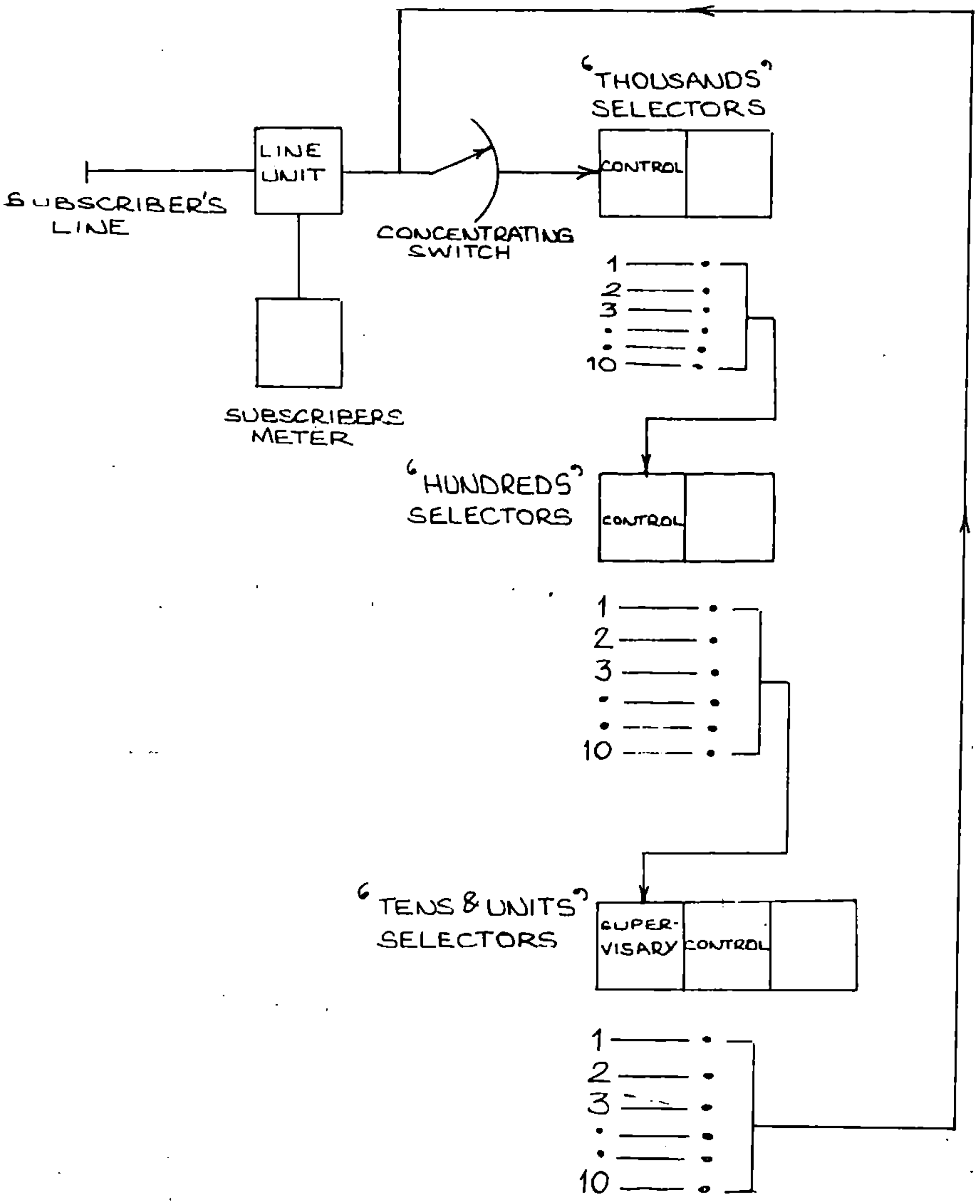
3.2.2 Manual Exchanges

The function of a telephone exchange is to interconnect on demand two or more of the exchange terminations for a period, of time, usually to permit speech signals to pass. In a simple conventional manual exchange, the terminations appear on multiplied jacks interconnected by human operators using double plug-ended cords. Each operator is provided with a headphone, keys and lamps to carry the control functions, and normally deals with up to 17 cord circuits. A striking feature of the manual exchange system is its flexibility and power.

To start with, signalling between the subscriber and the exchange is by the spoken word, including names and numbers. The use of human operators here made possible facilities which stored program controlled systems are trying to achieve, *e.g.* automatic call transfer based on a customer's visiting habits, ring back when free, *etc.* Human intelligence also made fault detection, fault and internal blocking avoidance more efficient. The attending, interconnecting, supervisory and operator functions of manual exchanges correspond to the individual line circuits, switching block, supervisory unit and the control unit of a modern exchange respectively. As the communication network grew, it became evident that the number of operators to be provided became prohibitive and call set-up times unacceptable. As a result, an automatic system had to be introduced.

3.2.3 Direct Control Systems

The direct Control System is referred to as the step-by-step or strowger system. Here the subscriber is given the ability to remotely control the set-up of his call. In the Strowger switch, the cord of the manual system was miniaturised and the plug modified into 'wipers'. It is a two-orthogonal-motion switch control. A subscriber uses a decadic rotary dial to set-up the switches. Each digit is arranged to operate a switch in each stage. A ratchet mechanism moves the wiper vertically in accordance with the number of pulses in a dialled digit. The wiper then hunts in a rotary manner for a free outlet to the next rank of selectors. All the switching stages except the last are operated by single digits, the last stage being operated by the last two digits, to select one of a hundred subscribers. The only common equipment provided here are those for functions such as routing, tone generation *etc.*, thus in a basic strowger system, the control is fully dispersed (Figure. (3.3)).



FIG(3.3): STROWGER STEP-BY-STEP SYSTEM

Since the subscriber does not have knowledge of the overall state of the exchange, the exchange capability is limited. Alternative routing is not possible as the trunking is rigidly fixed by the digit code. In addition, calls between local exchanges are characterised by a multiplicity of dialling codes with variable lengths depending on the number of switching stages, thus there is no unique code identifying a subscriber in the network. Despite these disadvantages, the Strowger switch is simple to understand, cheap and allows flexible modular growth of an exchange. As a matter of fact a large proportion of exchange equipment in the global telecommunication networks is still of Strowger type.

3.2.4 Register Control Systems

The disadvantages of the step-by-step or strowger control in the rigidity of routing and in non-uniform numbering were solved by the introduction of a register-translater (or director) - Figure (3.4). This enables a standard number to be dialled, only dependent on the location of the called subscriber. The translation function is made to vary from exchange to exchange to translate a directory number into an equivalent number (equipment number) which is actually used to route the call. The director is located immediately after the first concentration stage and through a further concentration stage (the register-access switch). The first few digits of a subscriber number represents the unique code digit of his exchange in his area (local exchange). The director translates these into a corresponding set of digits chosen to meet the requirements of switching and routing economy at the particular originating exchange. The digits after the exchange code are passed without translation. Thus, the prime aim is to eliminate the mutual

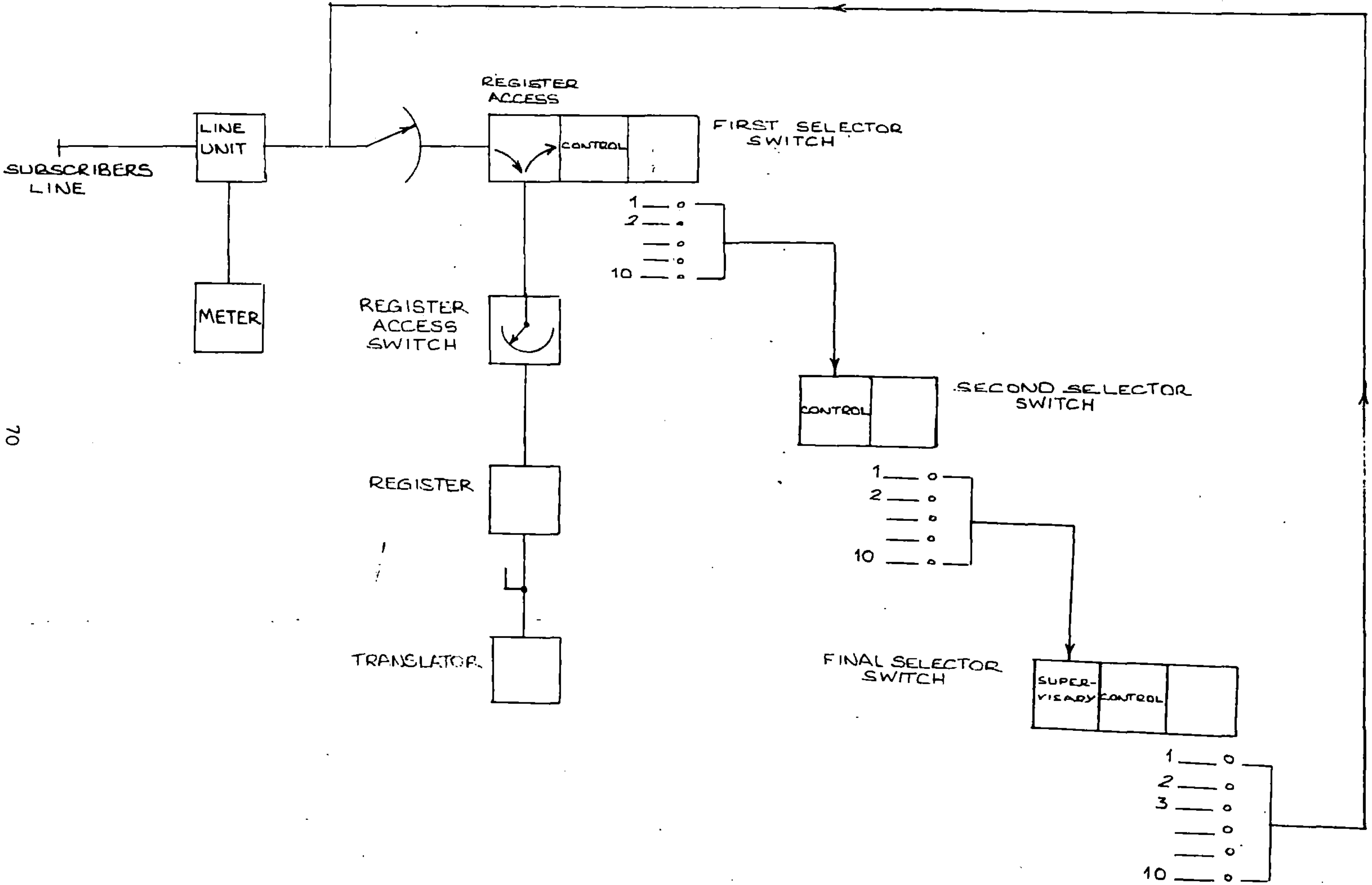
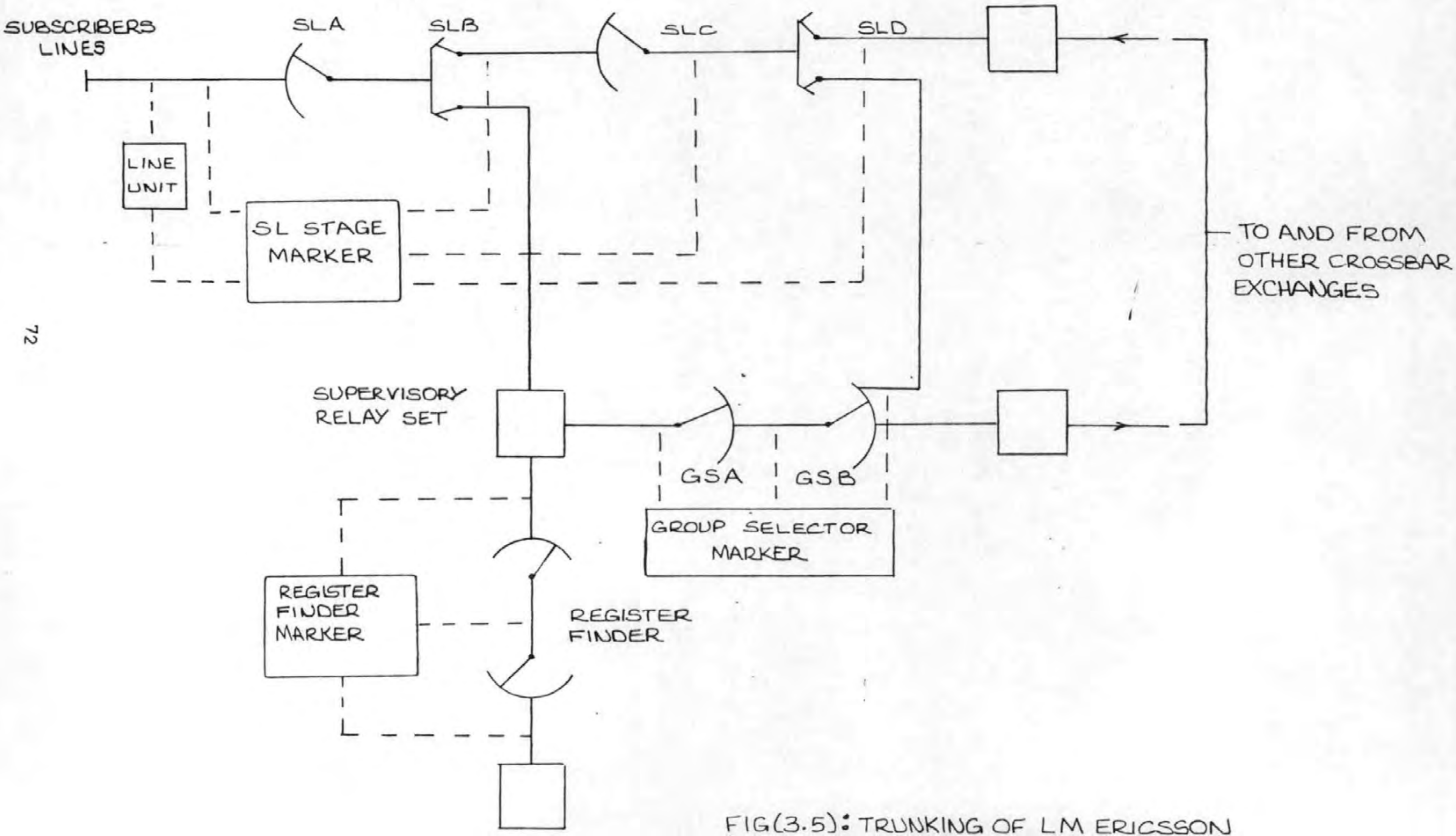


FIG (3.4): A STROWGER EXCHANGE WITH A REGISTER TRANSLATOR restrictions between telephone numbers and the switching machine.

A second reason for using a register is in the situation when the switching mechanism is unsuitable for the reception of dialled digits directly. Such situations include mechanical unsuitability of the switch in some motor-driven switches, or when the switch is not decadic or it is inefficient to be so as in a cross-bar switch. Hence registers are introduced in crossbar exchanges to store and translate the dialled digits in conjunction with a translator. Figure (3.5) shows the trunking arrangements of one cross-bar exchange - the Erricson type ARF. Three principal stages of the cross-bar switches are involved: Line-stage switches (SLA, SLB, SLC, SLD), register-access switches and group selector switches. At each stage, calls must pass through more than one switch and the operation of the switch is controlled by marker equipment common to many of the switches at each stage.

For an originating exchange call, SLA and SLB initially connect the subscriber to a free supervisory relay set by the SL stage marker. The supervisory unit is then connected to a free register by the register finder marker *via* the register connecting switches. The signalling information from the subscriber is then transferred to the register. The register performs the digit translation and signals the group selector marker at high speed to mark the appropriate connections.

For an outgoing call, a junction circuit is allotted, while for an own exchange call a path is connected back to the line-stage switches. Advantages of cross-bar systems include independence between directory and equipment numbers, high reliability, low post dialling delay in multi-exchange calls due to the use of multi-frequency signalling between exchanges and backward signalling which



FIG(3.5): TRUNKING OF LM ERICSSON ARF CROSSBAR EXCHANGE enables an originating exchange to drop a connection and make another attempt to set it up over the same or an alternative route. The use of common control necessitates duplication of equipment and fault detection and isolation equipment.

To economise on cross-points link-coupled trunking is used, where a common control interrogates the paths through two or more stages virtually simultaneously and establishes the connection as a result of this overall appreciation of the network. From about 1950 cross-bar systems were being installed in large quantitites. The reliability of the cross-bar switch was extremely useful, particularly when subscriber trunk dialling was introduced (STD) This is because the probability of encountering undetected faulty equipment with Strowger type of exchanges increases owing to the additional equipment necessary to complete a connection.

3.2.5 Electronic Systems

Electromechanical common-control systems still suffered from certain operational disadvantages. For instance, the number translating capability may have to be provided separately in every register and registers have a restricted choice of switch paths. Electronic techniques, both in the switching block and central control were introduced to alleviate these problems. The high speed of operation of electronic components coupled with powerful processors enabled more centralised control, more switching stages, better appreciation of the overall state of the network and flexibility of operation (Figure (3.6)).

Electronic systems are either hard-wired or stored-program-controlled, where the control functions are implemented by software in a central processor. Electronic exchanges are more and more approaching

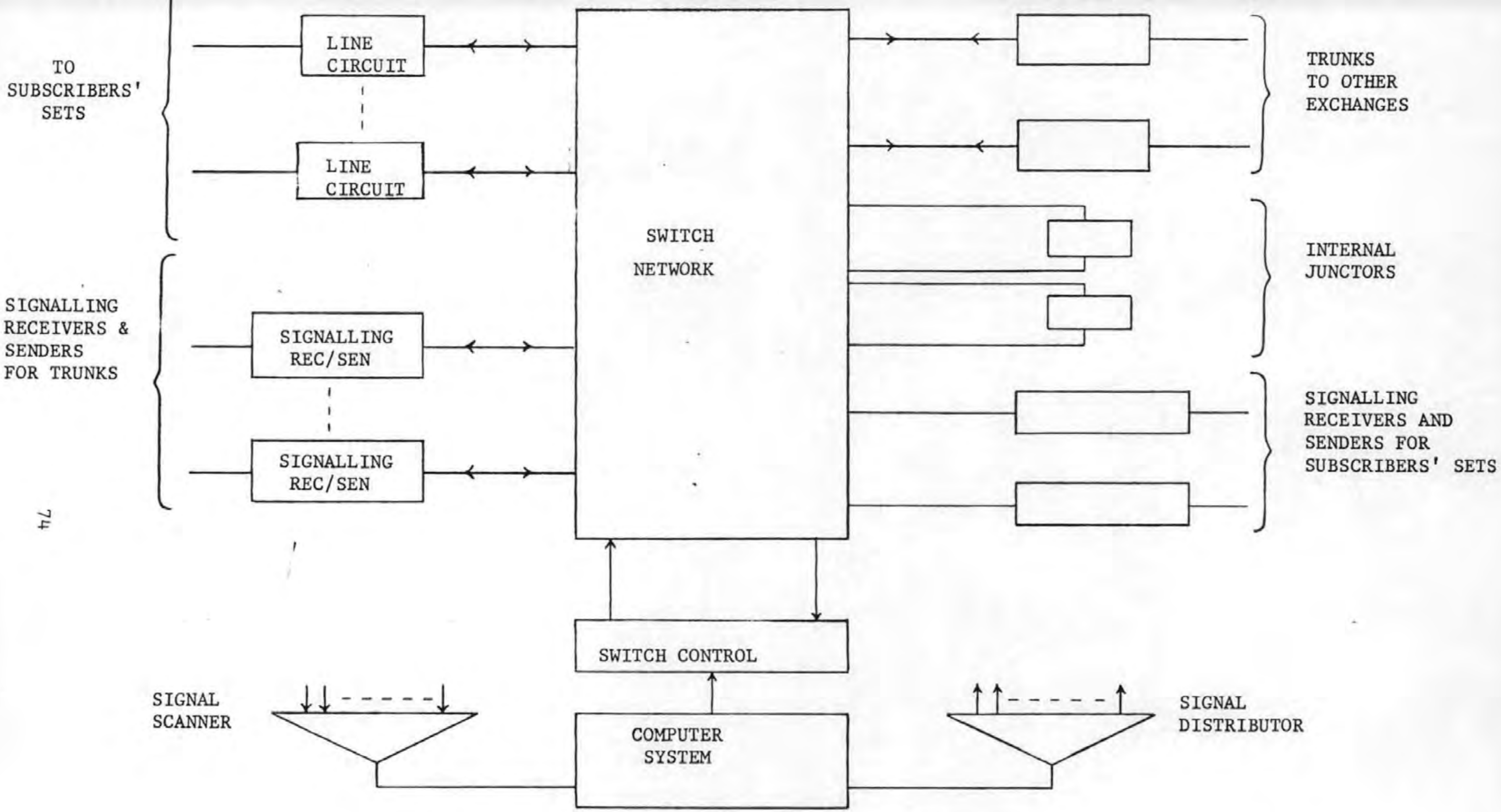


FIGURE (3.6) : A GENERAL MODEL OF A COMPUTER-CONTROLLED SWITCHING SYSTEM the flexibility and power of manual systems. In the switch block, the availability of cheap crosspoints in integrated circuit form and the use of digital switching (Pulse code modulation and time-division multiplexing) enabled the realisation of economic multi-stage switch blocks with very low probability of blocking. In the following sections, we consider in greater detail the characteristics of stored program controlled (SPC) systems.

3.3 STORED PROGRAM CONTROLLED (SPC) SYSTEMS

The control of telecommunications switching systems has evolved during telephony's first century from manual through electro-mechanical (in various forms) to electronic, both wired-logic and stored program control. The driving force behind this evolution is primarily economics (reduction of equipment and labour costs) as well as the need for enhanced capabilities. In this respect stored-program-controlled systems possess an overall capability not known to conventional systems. This overall capability is revealed by the following features:

(i) The system's ability to detect and isolate faults and reconfigure itself so as to provide a reasonably good service;that is system security.

(ii) The ability of the system to interwork with the existing network with its limited signalling capability, low speed and noisy environment; that is introducability.

(iii) SPC systems have powerful in-built maintainability features that provide rapid diagnosis, reporting and isolation of faults and reporting of defective parts or their restoration to full service after recovery. This is an important feature in the light of higher maintenance cost and scarcity of expertise. This ability is being utlised to centralise the maintenance in a few maintenance and administration centres.

(iv) SPC systems offer new facilities for both the administration and the customer. For the customer, it offers the possibility of new services such as conference calls, abbreviated dialling and call transfer. For the administration it offers improve network management, maintenance control and charging flexibility, all at an economic price.

(v) SPC systems have the evolutionary potential in both hardware and software. This is an important feature allowing the incorporation of changing design concepts and technologies over the life of the system.

(vi) SPC systems result in space saving, power saving and higher traffic capacity as compared to conventional systems.

(vii) SPC systems may cash in on the vast developing technology of digital computers resulting in further cost reduction.

All of these features have been brought about by the employment of software using powerful telecommunication-oriented processors. It is appropriate to consider in greater detail the different characteristics of these telecommunication processors.

3.4 TELECOMMUNICATION PROCESSORS

3.4.1 Their Characteristics

Telecommunication processors differ from commercial computers in that they have to:

1) Provide continuous service even in the presence of faults.

2) Operate normally in the exchange environment without special measures to control closely the ambient temperature, humidity, dust and electrical noise.

3) Be easily extendable in processing power without interruption to service.

4) Operate from the standard exchange power supplies and battery.

5) Use equipment practice compatible with the rest of the exchange.

6) Be maintainable by the exchange maintenance staff.

Although they are similar to commercial computers, they possess features unique either in nature or degree of application.

The real-time functions demanded by telecommunications processing may be classified under the following four classes:

(i) Scanning: the monitoring of the status of lines, trunks and service circuits at a frequency which is a function of the urgency required of a given class of entity.

(ii) Translation: the derivation from the directory number, the equipment number, class of service, routing information *etc.*

(iii) Call processing proper: hunting network paths, setting up paths, call supervision, charging, path clear down *etc.*

(iv) Maintenance: Checking, diagnosing, isolating and reconfiguring to various degrees.

The need to perform these functions efficiently has its impact on the processor structure and design. The need to perform scanning at rates independent of the processor load calls for a sophisticated interrupt mechanism to handle time-driven activities efficiently. Beside input/output data processing involving digit reception, digit sending, scanning and peripherals communication functions, the interrupt mechanism is implemented to service software and hardware traps and interrupts as well as a priority-based process structure. This interrupt facility is explained in more detail in Chapter 4.

The need for translation, call processing, input/output control and maintenance, calls for a processor with an extensive instruction repetoire: bit and data field manipulation, masking, rotating and shifting for translation; Boolean logic operations and address manipulation for the call processing, and special instructions for input/output, maintenance and diagnostics.

Telephone calls processing programs tend to be highly decision-oriented. Extensive testing and branching instruction are provided for this purpose. Input/output operations include transfer of peripheral equipment status (such as lines and trunks) and transfer of control, and addressing information. Highly-efficient macro-instructions for repetitive functions are constructed using micro-programming techniques, such as the calls to the process allocator in Mark II BL System (Chapter 4). Micro-programming also allows a machine to emulate another machine and could be used to replace a processor by a more versatile and updated one such as in the updating of No. 2 processor by No. 3 processor in the ESS family of SPC systems (MAND 76). For higher programming efficiency, an unusally high number of registers is provided and accessible to the programmers.

Earlier systems used read-only memories (ROM) for programs, translation tables and exchange parameters. More recently, cheap semiconductor RAM memories are used extensively, with disc, drum or tape back-up. With increased use of micro-processors, however, ROM and EPROM are much in use again.

Serial, parallel and serial/parallel bus structures are used for input/output control. In these systems with such a large number of peripherals, parallel bus with input/output blocks and address decode capability are found to be necessary.

The reliability and security levels required in SPC systems call for processor configurations with duplication or triplication of some equipment. In Chapter 4, these aspects are explained with reference to the GEC Mark II BL system. Arithmetic operations required for telephony functions are addition, subtraction and logical operations, though some telecommunication processors provide arithmetical functions similar to those of commercial computers. Table (3.2) summarises the telecommunication processors types in the world.

3.4.2 Configurations of the Processor Utility

To achieve the high level of hardware reliability required from the processor control sub-system of an SPC telephone exchange (typically 2 hours down-time in 40 years,) redundancy is used and implemented in a number of configurations. These may be classified as follows:

(i) Dual Worker, Stand-by

(ii) Dual Synchronous

(iii) Dual load-sharing

(iv) Multiprocessor

(v) Distributed Control

Figures (3.7.1) - (3.7.3) are simplified schematic diagrams of the first three configurations, whereas Figure (4.1) exemplifies a multi-processor structure.

In dual systems, the word processor implies a processor-store combination. In a worker -stand by system, one processor is taking the whole load. The stand-by processor is only switched on to take the load when the working processor develops a fault. The stand-by processor may either be 'hot'or 'cold' *i.e.* the power one or off. Provided the switch over time is less than the failure defined time (of the order of milliseconds), the reliability of such a system is the same as that of a dual system.

TABLE (3.2) : TELECOMMUNICATION PROCESSORS TYPES AND CHARACTERISTICS

(Ref: BRUC 77)

COUNTRY	Introduction Date	No. of Processors	Word Length	Modes	Micro-program	Gen. Registers	Memory
United States							
Bell System:							
No. 1 ESS	1965	1 pair	37/23	SYNC	N	0	Ferrite Sheet
No. 2 ESS	1970	1 pair	10,21/16	SYNC	N	0	Twister
No. 3 ESS	1976	1 pair	16,32/16	STANDBY	Y	16	IGFET
No. 1A ESS	1976	1 pair	24,48/24	SYNC	N	8	CORE-IC
Gen. Tel & Elec:							
C1 EAX	1967	1 pair	20	STANDBY	N	0	Diamond Ring
No. 2 EAX(2A)	1977	1 pair	32	SYNC	N	7	MOS(Dynamic)
No. 3 EAX(2B)	1978	≤ 4 pairs	32	SYNC/MULTI PROC	N	7	Semicond.
North Electric:							
ETS-4(APZ-130)	1975	≤ 7 pairs	16	SYNC/SHARE	N	4	CORE-IC
NX-1E(OMN14)	1971	≤ 4 pairs	16	SYNC/SHARE	N	4	CORE
Canada							
Northern Telecom:							
SP1	1971	1 pair	24	SYNC	N	0	Ferrite Sheet
DMS 100/200	1978/79	1 pair		SYNC	Y		PB-Twister
United Kingdom							
GEC:							
Mark II		≤ 12	16	MULTIPRO.	Y	16	Semicon
Plessey:							
S250		≤ 12	24	MULTIPRO.	Y	8	Plated wire core

continued.......

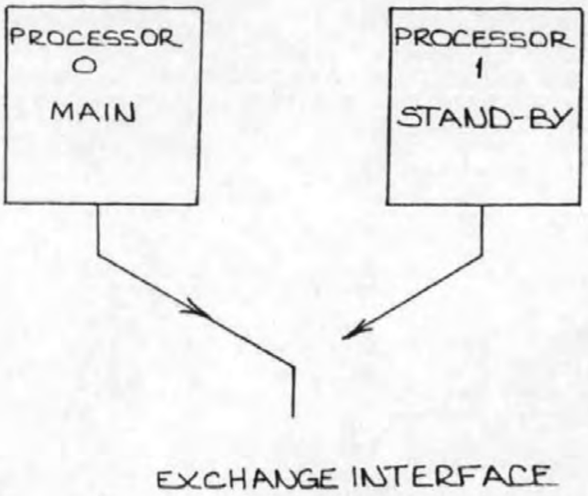
TABLE (3.2) TELECOMMUNICATION PROCESSORS TYPES AND CHARACTERISTICS continued....

COUNTRY	Introduction Date	No. of Processors	Word Length	Modes	Micro-program	Gen. Registers	Memory
France							
E10(CS40)	1970	5	32	FUNCTIONAL	Y		Core
E11(ITT3200)	1976	2	32	LOADSHARE	N		Core
E12(CS40)	1973	2	32	LOADSHARE	Y		Core
W. Germany							
ESK10,000(ELST 801)	1966	≤35	12		Y		Diamond Ring
EDS	1975	1 pair		SYNC	N		Core
EWS01	1973	1 pair		SYNC	N		Core
EWSF1	1978	1 pair		SYNC	N		Core
Sweden							
Ericson : ARE	1973	2-3		LOADSHARE	N		Core
AXE 10,11(APZ 210)	1975	1 pair +	16	SYNC	N		Core
AKE11(APZ110)	1968	1 pair	16	SYNC	N		Core
12(APZ120)	1968	1 pair	16	SYNC	Y	4	Core
13(APZ150)	1971	≤8 pairs	18	SHARE/FUNCT	N		Core
Netherlands							
PHILIPS:							
PRX205(TCP18)	1972	≤ 4 pairs	16	SYNC/SHARE			
(TCP36)		≤ 8 pairs	32	SYNC/SHARE	Y	8	Core

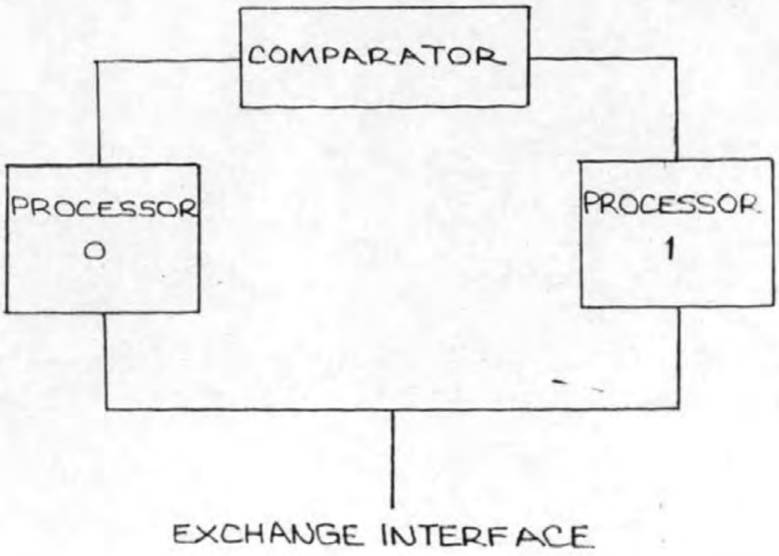
continued.......

TABLE (3.2) TELECOMMUNICATION PROCESSORS TYPES AND CHARACTERISTICS continued........

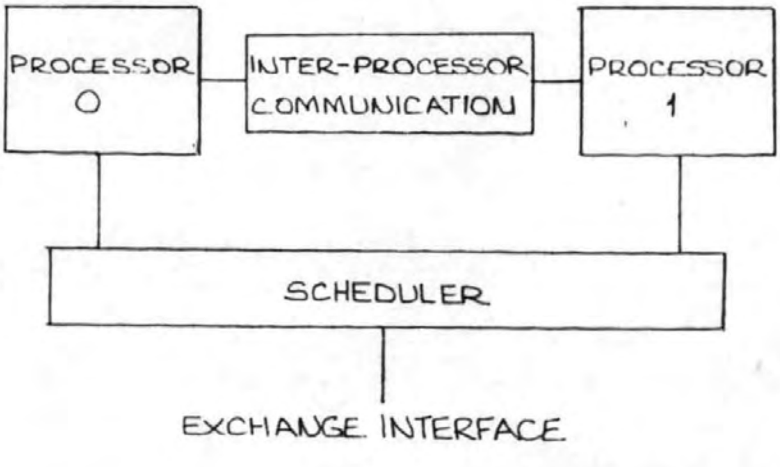
COUNTRY	Introduction Date	No. of Processors	Word Length	Modes	Micro-program	Gen. Registers	Memory
Japan							
D-10	1971	≤ 2 pairs	32	SYNC/SEP	N	16	Core-plated wire
D-20	1973	1 pair	16	SYNC/SEP	N	4	MOS, DRUM
ITT Metaconta							
10C(ITT1600/3700 3202)	1973	≤ 8	16/32	LOADSHARE	N	16	Core
11AC(ITT1600/3200)	1968/72	2	16/32	LOADSHARE	N	16	Core
TCSS(ITT1050/1652)	1974	2	16	LOADSHARE	N	16	Core



FIG(3.7.1): MAIN AND STAND-BY SYSTEM



FIG(3.7.2): DUAL SYNCHRONOUS SYSTEM



FIG(3.7.3): DUAL LOAD SHARING SYSTEM

The stand-by processor might be updated by the worker and given background work to perform. Disadvantages of such a configuration are that detection of errors is difficult, switch-over circuitry is costly, and calls in the set-up stage are lost when one machine fails.

In dual synchronous systems, both processors are locked together at the clock frequency to perform in synchronism the same function, but only one processor provides the output. The result of each operation is compared by the comparator circuitry and if disagreement occurs, diagnostic routines are run to determine which of the machines is faulty. This is the most widely-used configuration in terms of the number of systems designed and those in service. Advantages include easier fault location due to instantaneous comparison of data at all stages of processing and absence of contention between machines when handling the same call. On the other hand, the configuration poses some problems, namely the correct determination of the faulty machine, the reliability of the comparator, the 'deadly embrace' problem where the faulty machine takes out of service the good machine and the difficulty of connecting backing store units such as drum, disc or magnetic tape units to the synchronous machines. Examples using this configuration are No. 1 ESS (USA) and SP-1 (Canada).

In dual load-sharing systems, the work load is divided between the two processors by a scheduler. In classical load-sharing, the two processors are switched on and off say 10 msec on and 10 msec off. During its on period, the processor handles the traffic that originates during that period. Each machine updates the other by using messages. Advantages of load-sharing include lower probability of simultaneous program faults and better overload characteristics compared to dual synchronous systems. It is also more flexible in that the load sharing percentage may be reduced from 50-50 to 100-0 to allow for on-line updates and for the introduction of new facilities. Its problems include the requirement for comprehensive diagnostic software and self checking circuitry and the possibility of 'deadly embrace'. An example of a load-sharing system is the Metaconta L (KOBU 72). Some dual systems such as the E 11 (France) and the Metaconta 10 and 11 (ITT) can be switched dynamically from load-sharing during busy periods to a synchronous pair during light traffic periods (BRIL 77). Such a system has better overall performance characteristics.

A common disadvantage of all dual systems is the high cost of incremental growth, since the processors are added in pairs with increased problems of inter-processor communication and fault location. A multiprocessor configuration alleviates this problem and allows the m + n redundancy principle to be used, e.g. in the Mark II BL System: m processors take the load and n are redundant. In this configuration, the processors and storage modules are separate and connected to a common highway and each processor can access any of the storage modules. Jobs may be run on any available processors. Examples of this configuration are the GEC Mark II BL system and the Plessey S250 system. The Mark II BL is described in greater detail in Chapter 4. A major problem with multiprocessor systems is fault detection, containment and system recovery due to faulty processor corrupting areas of the common store. Sophisticated techniques have been developed to deal with this problem (OWEN 73, EDGE 72). Other examples of multiprocessor systems are the E12 (France) and the PRX (Netherlands).

An [illegible] example of the relative reliability of dual and multiprocessor systems is given below, based on the procedure used in the Post Office Requirements Document 1075 (HALT 77).

$$F_{100} = \frac{8.76 \times 10^5}{D} \quad mC_{n+1} \left(\frac{D}{u}\right)^{n+1} (n+1)$$

where F_{100} : number of failures per 100 years

D : Mean time to repair

u : Mean time between failure of module

m : Total number of identical modules

n : Number of spare modules

8.76×10^5 : Number of hours in 100 years

D is normally taken to be 5 hours. U is calculated from the reliability figures of components in a module and is typically 8920 hours or approximately one year.

Consider a dual processor system where the maximum load can be taken by one processor, then

$U = 8920$ hours

$m = 2$

$n = 1$

$$mC_{n+1} = \frac{m!}{(n+1)!(m-n-1)!} = 2C_1 = \frac{2.1}{1.2} = 1$$

$\therefore\ F_{100} = 175 \times 10^3 \times 1 \times 31.3 \times 10^{-8} \times 2 = 0.10955$

$\therefore$ MTBF $\simeq$ 912 years. MTBF is the mean time between failures.

On the other hand, considering a multiprocessor system where the load is taken by 3 processors with only one processor spare *i.e.* $m = 4$ and $n = 1$, we get

$F_{100} = 0.6573$

and

MTBF = 152 years.

The advantage of the multiprocessor system is that it provides a reasonably high degree of reliability at an economic price and allows for a smooth economic growth. If dual processors are used for the above multiprocessor systems, a total of 6 processors would have been required.

With the advent of microprocessors, the trend is again shifting towards distributed control using loosely - coupled microprocessors in a multi-microprocessor configuration (NISS 79, CULL 79). These systems are still under development.

3.5 SPC SOFTWARE ORGANISATION

There is no standard classification of SPC software types, however. One possible such classification is:

1) Call Processing Proper
2) Real-time Operating System and Support
3) Maintenance and diagnostic
4) Administrative.

The total amount of software in an SPC system is huge and costly. Formalised methods of specifying and producing modular, efficient and manageable software have been proposed and used. Such methods include structure-oriented modelling (BRAE 79), specification and description languages based on state transition diagrams (CCIT 77, KAWA 71, GALE 75, GERR 74,) structured programming (DIJK 72, BAKE 75), generic program production (KAWA 79) and verification-oriented software specification (CUNN 81).

1) Call Processing Proper:

These are the programs that implement the telephony functions and facilities in a particular exchange, for example, call detection, digit reception, route translation, path search, and supervision. They also implement new facilities, such as conference call, camp-on, and abbreviated dialling.

Three approaches in implementing the suite of programs are generally followed, namely, function division, time division and call division or state-of-call (SOC) (HILL 76A, LAWR 72). A combination of these approaches is normally employed, for example function and time division in System X software. Program modules that have stringent real-time requirements or carry important functions are activated periodically and have a higher priority than other programs.

2) Real-Time Operating System and Support

The real-time operating system manages the resources of the processing utility, including the CPUs and store blocks, and provides facilities such as timing, interrupt handling, I/O and communications between the software modules. Its structure depends to some extent on the architecture of the machine for which it is written. It runs on-line.

Support software includes compilers, linkers, loaders and debugging aids and these are off-line programs to effect modifications and/or extensions of existing programs or additions of new ones. Debugging aids include module logic testing using emulators, system testing using an on-line break-point program or data tracer and simulation for design checking, performance evaluation and system tuning.

3) Maintenance and Diagnostics:

The overall maintenance process is composed of fault detection, fault recovery, fault diagnosis and fault repair in this sequence. Great importance is attached to these maintenance programs in view of the high reliability required of SPC systems (1 hour in 20 years downtime) in spite of the multiplicity of hardware and software faults that may arise. Half of the total exchange software is likely to be for maintenance and diagnostics(Table (3.3)).

Fault detection is aided by hardware parity checking, comparison of outputs from duplicated quipment, validity checking, routing and time outs. Fault isolation and recovery is carried out before fault diagnosis because of the necessity for guaranteed continuity of service. Diagnostic software is then invoked which will examine the faulty entity if it is a hardware fault or re-initialise it if it is a software process. Sophisticated techniques to implement this function have been developed for various systems (OWEN 73, EDGE 72, ARGO 79).

4) Administrative

Ease of management of SPC systems is important to operating administrations. Software in this area is used for operations 'soft data' or semi-permanent data operations relating to routing, junctions , exchange configuration control and extension, traffic measurement and subscriber and system monitoring.

SYSTEM	MAINTENANCE CODE (WORDS, APPROX.)	APPROX. PROPORTION OF OPERATIONAL SOFTWARE
ESS1 (USA)	100K	53%
D10 (JAPAN)	67K	54%
CNET (FRANCE)	50K	50%

TABLE (3.3) : Percentage of Maintenance Software for some of the Well Known SPC Systems

	Termination Capacity	Switch Capacity (Switched erlangs)	Processing Capacity (busy hour call attempts)
Multiplexer	24 or 30	4 or 5	
Concentrator	2000	160	8000
Small local Exchange	2000	160	8000
Medium local Exchange	10000	2000	80000
Large local Exchange	60000	10000	500000
Medium Trunk Exchange	8000	2000	80000
Large Trunk Exchange	85000	20000	500000
Medium International Transit Exchange	8000	2000	50000
Large International Transit Exchange	85000	20000	400000
Combined local and Trunk Exchanges	10000 subscribers of 5000 trunks in Combination	2000	80000

TABLE (3.4) : SYSTEM X FAMILY OF EXCHANGES (TIPP 79)

The administration communicates with the exchange through I/O devices, such as a teletype using a man/machine language, in addition to a high speed paper tape punch, a magnetic tape unit and a test panel or console. All the above functions are in operation while the exchange is carrying live traffic. In addition, centralised maintenance and administration centres for remote exchanges are possible using SPC systems.

Beside the operating system and applications software, the SPC software also contains the exchange-dependent data base relating to customer data, translation tables, routing and charging information and call records.

3.6 SYSTEM X

3.6.1 System X Characteristics

The present telecommunication network in Britain is dominated by electromechanical switching and signalling systems. The ability to exploit these systems on a network basis is constrained by a variety of limitations inherent in these systems. These constraints and limitations of the national network can be summarised as:

1) Domination by 2-wire switching and channel associated signalling.

2) Multiplicity of limited-capacity signalling systems.

3) Relative slow set-up time for multi-link calls.

4) Transmission loss varies with call routing.

5) Prone to noise and distortion.

6) Limited capacity for further evolution and provisioning of new facilities.

7) Mechanical switches prone to wear.

8) Manufacture and maintenance of equipment highly labour intensive.

9) Out of tune with modern technology.

Thus, in the late sixty's, the need was realised for a system which will allow the present network to evolve into one with much greater capabilities was identified by the Advisory Group on Systems Definitions (AGSD) (MART 79).

This fact, coupled with the advances in device technology, led to the adoption of a total approach to the present network problems in a joint venture between the British Telecommunications and the three major telecommunication equipment manufacturers, GEC, STC and Plessey. As a result, System X was born. System X is based on a family of new switching and associated systems characterised by the use of micro-electronics technology, integrated digital transmission and switching, common-channel signalling and stored-program-control. System X is characterised by:

a) 2-wire or 4-wire subscriber switching.

b) 4-wire junction and trunk switching.

c) Extensive signalling capability using common-channel signalling.

d) Fast set-up time for multi-link calls.

e) Transmission loss independent of circuit length because of the use of PCM transmission.

f) Low noise and distortion by the use of digital switching as opposed to electromechanical switches.

g) Extensive capability for evolution and provisioning of new facilities.

h) Facilitates interworking with international networks.

i) Digital switch reliability.

j) Potential automation of manufacture and maintenance of equipment.

k) Exploitation of modern electronic techniques.

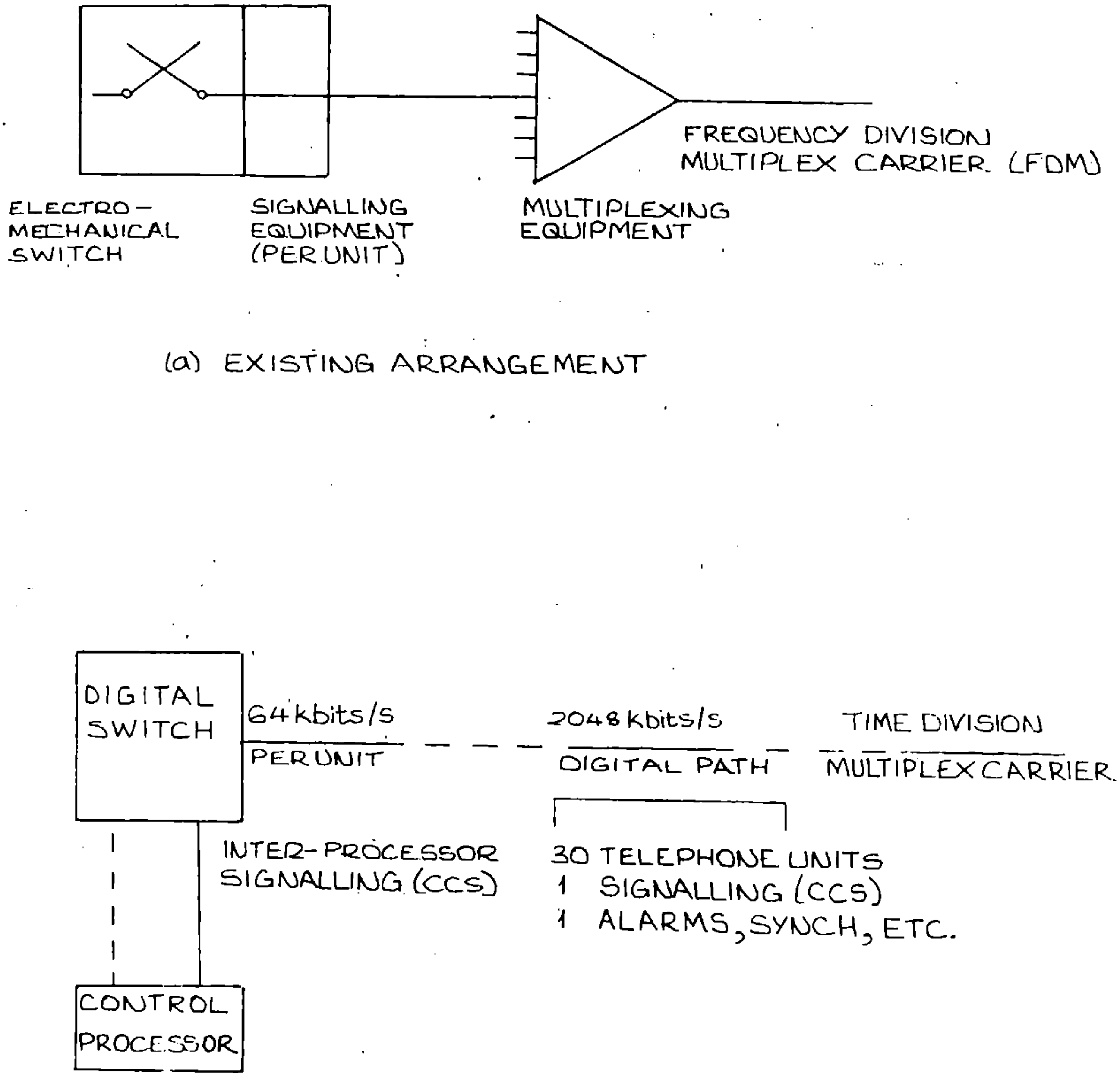
l) Integrated transmission and switching using digital techniques minimises the equipment required at the interface between the transmission and exchange equipment. The findings of the UK Trunk Task Force (UKTTF) in the late 60's indicated that by opting for an integrated digital network (IDN), reduction in the total main network cost, in terms of annual charges could be as much as 50% (HARR 79). This is mainly due to the elimination of the intermediate primary multiplexing equipment, pre-circuit signalling equipment, cheaper electronic components and lower manufacturing, accommodation and maintenance cost (See Fig. (3.8)).

m) Use of common channel signalling, because signalling equipment is no longer associated with individual channels there is a significant reduction in signalling equipment required in the system. The signalling sub-system is used to send both control and network management information in the form of flexible, open-ended data messages. Other concepts of System X are remote control of exchange functions and concentrator working. (JONE, 79).

3.6.2 System X Architecture

Among the concepts imbedded in System X design is adaptability and in-service flexibility. This concept is realised through a modular architectural approach both in hardware and software. An individual System X exchange is thus made up of an appropriate number of software and hardware modules or building blocks drawn from a large set of modules.

The sub-division of System X into a number of modules or sub-systems is done on a functional basis. A sub-system interworks with other sub-systems across well-defined functional interfaces which form the boundaries for the sub-systems. They also provide convenient points for growth and adaptability.



(a) EXISTING ARRANGEMENT

(b) INTEGRATED SWITCHING AND TRANSMISSION WITH COMMON CHANNEL SIGNALLING

FIG (3.8): CHARACTERISTIC FEATURES OF EXISTING AND NEW NETWORKS

This approach is attractive in many ways. Changes in user service requirements in traffic or operation strategy can easily be effected by additions or modifications of individual modules. With the rapid advances in microelectronics, new improved devices are easier to incorporate into the hardware modules. The generic nature of System X software provides a central software facility for the generation and assembly of the software sub-systems necessary for particular exchanges. The modular structure with clearly defined interfaces reduces the probability of hardware and software errors propagation. Finally, and most importantly, the modular structure enables the development of the sub-systems to be undertaken by a number of development teams simultaneously and at different physical locations.

3.6.3 System X Sub-systems

(a) HARDWARE SUB-SYSTEMS

1) Processor Utility Sub-system (PUS)

The PUS provides data processing facilities for traffic handling and control of its own and other exchanges. As mentioned before, two versions exist; a smaller version (Mark IP) used in a worker -stand-by configuration for small and medium-size exchanges, as a preprocessor and in SPC assisted electronichal systems The second version is the Mark II BL multiprocessor system for large local and trunk exchanges. Both types are managed by a sophisticated real-time operating system. This sub-system is developed by GEC and is fully explained in Chapter 4.

2) Digital Switching Sub-system (DSS)

A DSS has a time-space-time switch configuration as is used to interconnect time-division multiplexed, pulse-code-modulated channels. Each time-switch provides for the connection of 32 line systems with 32 time-slots in each system, and in a large exchange up to 96 such time switches may be provided (TIPP 79, JONE 79) to interconnect both transmit and receive channels with very low blocking probability. This sub-system has a software handler.

3) Signal Interworking Sub-system (SIS)

SIS facilitates interworking with existing exchanges that use a multiplicity of signalling systems. It also provides tones and recorded announcements. It has an SIS software handler.

4) Message Transmission Sub-system (MTS)

The MTS performs common channel signalling functions over digital bearers at 64 kbits/s. The MTS is based on the British Post Office Signalling System Common-Channel No. 1 (SSCC No. 1) (Jone 79) which may be made compatible with the CCITT signalling system No. 7 for national and international applications.

5) Analogue Line Terminating Sub-system (ALTS)

An ALTS is a system for processing signals before enetering and after leaving a System X switching node, combining an analogue to digital converter and a digital to analogue convertor.

6) Network Synchronisation Sub-system (NSS)

The NSS ensures that all System X exchanges in the network are operating at the same average bit rate by synchronising the exchanges clocks to a master network clock.

7) Subscriber Switching Sub-system (SSS)

An SSS is used for traffic conectration into heavily used common circuits at a local exchange.

(b) SOFTWARE SUB-SYSTEMS

1) Call Processing Sub-system (CPS)

A CPS interfaces with other sub-systems to control calls progress through the system. This sub-system is dealt with in greater detail in Chapter 7.

2) Call Accounting Sub-system (CAS)

A CAS is located at the local exchange and is responsible for collecting data for call-charging purposes.

3) Maintenance Control Sub-system (MCS)

The MCS is used for system diagnosis under faulty conditions and provides help for maintenance staff.

4) Management Statistics Sub-system (MSS)

The MSS is used for the collection of telephone statistical data for short and long term planning.

5) Overload Control Sub-system (OCS)

The OCS monitors the load on the processor sub-system and when the load on a particular sub-system or the total system load exceeds a pre-determined level, this sub-system interferes to modify the mode of operation of one or all of the other sub-systems.

6) Multiparty Connection Sub-system (MPCS)

The MPCS provides for conference calls where three or more parties participate in a conversation.

7) Automatic Announcement Sub-system (AAS)

The ASS synthesises announcements from digitally recorded segments of speech for customer exchange interactions. This is used particularly for supplementary services.

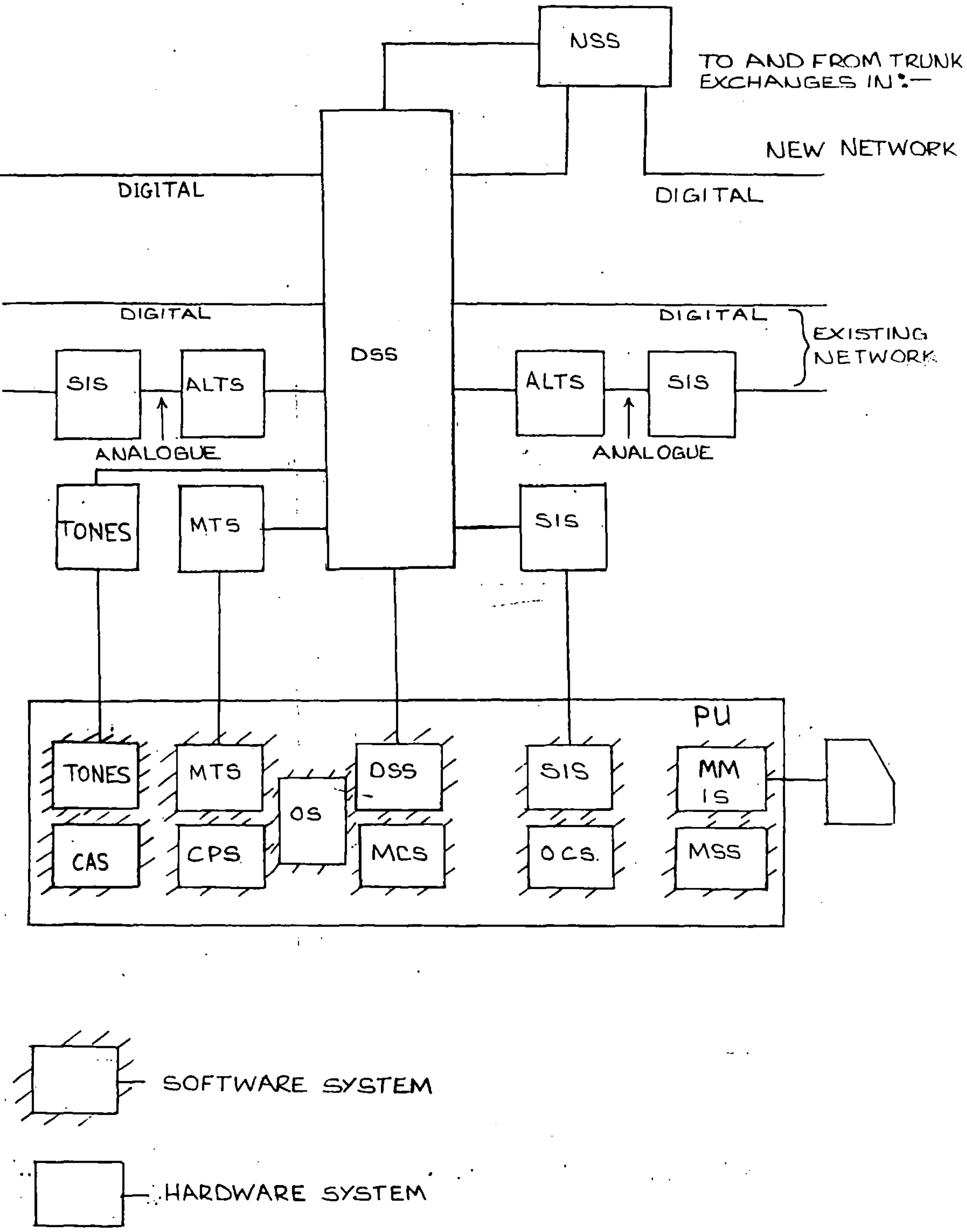
8) Man/Machine Interface Sub-system (MMIS)

The MMIS provides communication facilities between operation and maintenance staff and the processor sub-system for monitoring, controlling and maintaining an exchange.

3.6.4 System X Family of Exchanges

The range of System X family of exchanges is shown in Table (3.4). It is characterised by a wide range of capacities. Its prime role is in the large and medium systems to meet the needs of the telecommunications administrations both here and abroad (TIPP 79). At the lower end of the range (concentrator and small local exchange) with very low traffic, the system designers have to adapt the sub-systems and hardware layout in order to provide economically viable common control, digitally switched exchanges.

At the higher end, the problem is the achievement of the high level of throughput shown (500,000 busy hour call attempts). For a central control using a multiprocessor configuration, this calls for a very efficient operating system and application software. This efficiency also depends on how much software is processed in the central control and how much is devolved to microprocessors located with the hardware of specific systems as well as on the processor speed. Such a high throughput is difficult to achieve and test. One of the objectives of this research is the provision of a software tool to enable the testing of this high throughput (c.f. the DMNSC in Chapter 7). A block diagram of a typical System X Trunk exchange is hown in Figure (3.9).



FIG(3.9): SYSTEM X TRUNK EXCHANGE (DMNSC)

The System X family of exchanges includes,also, local administration centres (LAC) (HARR 79) for exchange and network administration. Each individual local administration centre caters for the administration of a group of exchanges. A local administration centre provides a control centre for maintenance, a concentration point for data being transferred between exchanges and data processing centres. It also provides a concentration and control point for man/machine communication with exchanges.

The penetration of System X in the existing network is expected to be gradual and over a number of years. As more System X and digital transmission equipment is introduced the network will evolve into an integrated services digital network providing 64 kbits/s transmission capability initially, and higher rates subsequently to provide for a multiplicity of services.

3.7 PERFORMANCE EVALUATION OF COMPUTER AND SPC SWITCHING SYSTEMS

3.7.1 The Objectives

The objectives of evaluating the hardware and software performance of computer systems used in commercial and scientific applications and those used in SPC switching systems are similar. The evaluation of computer performance is of vital importance in the selection of computer systems, the design of applications and equipment and the analysis of existing systems.

Historically, only computer hardware performance was evaluated. This included parameters such as the organisation of the machine: the word size, data path and number of addresses per instruction, I/O channels and secondary storage. With the evolution of the third-generation computers, the programming system has become an integral part of modern computers. Hence-the evaluation process must consider the software as well as the hardware in performance evaluation.

The capabilities of the operating system are central to the performance of a computer system and in particular its multi-programming and multi-processing features. Application programs and software utilities are also a part of the computer system and their performance contribute to the total system performance. Thus, more had to be learned about the internal system operations such as scheduling algorithms and resource allocation policies. Time-sharing, interactive processing and real-time processing emphasized the need for performance evaluation and analysis. This is quite understandable in the light of the high cost of software development and hardware designs, large investments in installed computer equipment and the high cost of running a computer centre. Hence, means and ways of increasing system efficiency through hardware and/or software changes are very desirable.

To put the objectives of computer systems evluation into perspective, a number of classification schemes have been suggested by practitioners in the field. Lucas (LUCA 71) cited three general purposes of performance evaluation: selection evaluation, performance projection and performance monitoring.

Selection evaluation is concered with selecting a computer system from among a number of systems on the basis of certain performance criteria. Performance projection is oriented towards designing a new system, a hardware component or a software package.

Performance monitoring provides data on the actual performance of an existing system, by the use of software and hardware probes. This information is used to forecast the impact of changes in the system such as a reconfiguration of the hardware or an improvement in the frequently executed software modules. It is also used to obtain a profile of the use of the system to aid in making strategic decisions, such as the allocation of priorities to process instances in Mark II BL System.

On the other hand, Svobodova (SVOB 76) differentiates between two types of performance evaluation: comparative and analytic. The former includes:

a) Lease or purchase of new hardware and software.

b) Selection of a supplier of computing services.

c) Classification of existing systems.

d) Comparison of the performance before and after a modification. (DAVI 74, KASP 74)

In analytical evaluation, the performance of the computer system is assessed against variations in the system parameters and work load. This is usually done in an attempt to:

a) Improve the performance of an existing system (system tuning).

b) Maintain the performance of an operating system within specified limits (Performance Control).

c) Design and implement new systems (FAGA 74).

A third categorisation is, due to Bell (BELL 72). He identifies five categories of objectives for performance evaluation and analysis of computer systems. These are feasibility analysis

(BAU 74), procurement decision making, design support, determining capacity (for existing or projected systems) and improving system performance (tuning). He further proposed a methodology whereby the issues in a simulation are associated with the objectives in a matrix form with the solutions as entries. He also compacted the five objectives into three to make his matrix methodology easier to handle (Table (3.5)). The three alternative categories of objectives suggested were:

i) Absolute Projection: Here the objective is to make basically dis-similar comparisons. This is the case for example, where a system's processing capacity for a particular load is tested against the maximum allowable time or the response times compared with the stated requirements. A characteristic of these simulations is the necessity of evaluating an objective function in absolute terms with a high degree of absolute accuracy. An example of this is the evaluation of the delays in the DMNSC model, (COHE 68), (STAN 68).

ii) Sensitivity Analysis: This category includes those performance evaluation studies where the emphasis is on similar comparisons (KUCH 75, SHER 72, UNGE 77, KATZ 66, FRAN 73, FAGA 74). Here high accuracy is only required in the areas in which the two cases are not identical and the areas that significantly interact with the non-identical areas. A basic characteristic of sensitivity analysis is the comparison of slightly different alternatives, for example the effects of changing a hardware component, a software module or ; the scheduling algorithm. A relevant study here is that of experimenting with an alternative

ISSUE	OBJECTIVE		
	Absolute Projection	Sensitivity Analysis	Diagnostic Investigation
Resources	Critical	Important	Desirable
Changes	Desirable	Critical	Important
Boundaries	Desirable	Important	Critical
Costs	Desirable	Important	Irrelevant
Experimental Design	Important	Critical	Desirable
Detail	Macro	Moderate	Micro
Accuracy	Critical Overall	Critical in places	Reasonableness only
Validation	Value Comparison	Derivative Comparison	Sequence Checking

TABLE (3.5) : Issues vs. Objectives

scheduling algorithm for Mark II BL system where a process running is only interrupted if it is a background process (Chapter 6).

iii) Diagnostic Investigation: The main objective here is to gain insight into the detailed manner in which the system's components interact and behave or tracing the progress of a transaction or an object to determine whether it goes through the system as expected. The verification experiment reported in Chapter 6 falls into this category. Other cited references include LEHM 68, BARK 69, UNGE 77, REI 68, BLUN 74. The issues or problems particular to the simulation technique of performance evaluation are associated with these three objectives in matrix form as shown in Table (3.5), and will be dealt with in greater detail in (3.7.3).

3.7.2 Performance Evaluation Techniques

3.7.2.1 Introduction

Performance of a computer system can be looked at from two different angles. It may either be defined as the effectiveness with which the system handles a specific application or be defined as a measure of internal efficiency (SVOB 76).

Effectiveness is what is seen by the user, that is "How well does the system enable me to do what I want to do?" Efficiency is how the system uses its resources in order to process a particular work load; "How well does the system do what it is intended to do?" Moreover, system performance can be evaluated only with respect to a particular workload. By workload we mean the total of resource demands from the users community.

But workload characterisation and modelling still remains one of the formidable problems in the field of computer performance evaluation and much attention has been paid to it. The following section looks more deeply into this problem area.

3.7.2.2 Workload Characterisation and Modelling

In most computer systems, the instantaneous workload changes quite unpredictably. This is especially true for interactive systems where the speed of the user's response plays an important role in shaping the load generated at individual system entry points. It is this uncontrollable fluctuation of the system workload that makes the characterisation of the workload and hence the system performance valuation so difficult.

Many parameters are used to describe the workoad in computer performance evaluation studies. For example job CPU time , I/O requests, inter-arrival time and priority. A complete list of parameters used in workload characterisation is found in (SVOB 76).

Although the workload of a particular system normally has statistical properties that remain constant over reasonably long time periods, these characteristics change as the users community changes, and as new facilities are introduced. This has made workload characterisation difficult to undertake and has lead to the new beatitude: "Blessed is he who found his computer's workload. Let him ask no other blessedness"

Fortunately, in SPC switching systems, the workload is represented by the arrival of telephone calls, characteristically a Poisson arrival process. Moreover, the resource demands generated by the arrival of a telephone call are known beforehand and deterministic in nature. In other words, the processing of a call is identical to that of all other calls of the same type and hence the computer resource demands are identical. This is unlike commercial and scientific computer systems where every individual job arriving has different resource demands. In such cases, workload models have to be constructed as workload drivers for an actual system or a model of it. A number of such workload models have been suggested and used.

One such model is the instruction mix. This specifies the relative frequencies of usage of different instructions (*e.g.* ADD, MULTIPLY, JUMP *etc.*) in a particular application, and it is different from application to application whethar scientific or commercial, but one particular mix, the Gibson mix, (GIBS 70) is believed to be applicable to a wide class of applications.

Another workload model is a benchmark. This can be an instruction, a special program or a sequence of calls to selected software components. In most cases, however, the term benchmark is used to mean a set of jobs (selected by random sampling of the job stream) that represents a typical workload of the evaluated system. A good benchmark is meant to represent the classes of jobs present and to exercise all the system functions such as scheduling, file management and I/O (LUCA 71).

Another workload model is a synthetic benchmark or a synthetic program. As the name implies, it is a program or set of programs constructed either from the resource demands or service demands but does not necessarily exist before hand (BUCH 69). It includes I/O considerations, files and environment provided by the operating system (LUCA 71).

Another workload model is the trace. This is a record of selected events that preserves the exact sequence in which these events occurred in the system. It is obtained either from the system natural workload or from a representative benchmark mix. It is used to drive simulation models where the pattern of sequence of events is important (CHEN 69, SHER 72, NOE 74).

Probabilistic workload models, where the workload is characterised by a probability distribution such as the negative exponential distribution, are also reported to have been used both in analytical models and simulations (SVOB 76). These are, of course, approximate models of workload characteristics of computer systems because they are difficult to represent by simple mathematical distributions (ANDE 72).

Models used in conjunction with interactive systems are known as interactive workload drivers. These models take care of user characteristics (such as think time, type time and user-generated interrupts) as well as modelling of resource demands by synthetic benchmarks or ones constructed from the real system commands and input data.

In SPC systems, as telephone calls arrival is characteristically a Poisson process and resource demands of calls of a particular type are identical, the workload characterisation and modelling is simplified to a great extent. A central call generator is used to inject calls into the model at intervals drawn from a negative exponential distribution whose mean is determined by the telephone traffic. For each call generated, the time taken from the start of ringing to the called party answering and the call duration are drawn from negative exponential distributions with means of 6 seconds and 3 minutes respectively. Resource demands per call are represented by modelling the events occurring at the line circuit. SPC workload characterisation and modelling are discussed in greater detail in relation to the DMNSC in section (7.2.5)

3.7.2.3 Performance Evaluation Models

Computer system performance is a function of a number of parameters the most important of which are the following:

1) System configuration

2) Resource management policies of the operating system.

3) Efficiency of system and application programs

4) Effectiveness of the processor instruction set

5) The speed of the hardware components.

The performance characteristics are shaped through modelling and measurement during the system design, system implementation and when matching the system to a given workload. These characteristics are shaped through the adjustment of system control parameters, change or modification to resource management policies, load balancing through system reconfiguration and replacement or modification of system components. With the system software, the efficiency of a program is determined by the efficiency of the used algorithm, the programming style and the implementation language.

The general classification of models was outlined in Chapter 2. For the purpose of evaluation of computer and SPC systems, these include empirical models, such as regression models (TSAO 72, SALT 74), analytical models and simulation models.

An analytical model is a mathematical representation of a computing system derived by analysis of the behaviour of the system. A number of such models have been developed, those based on queuing theory are particularly numerous in the literature. They are frequently employed to provide performance data on one particular system component such as CPU scheduling, though whole time-sharing multiprogramming and multiprocessing systems have been approximated by queuing models (FRAN 74, KUCH 75). Unfortunately, representativeness and mathematical tractability in analytical models are conflicting requirements. The class of problems that is solvable with existing mathematical methods is limited and many simplifying assumptions have to be made for other than simple systems. In spite of these simplifications, analytical models play an important role in performance analysis: they provide insight and a quick first-order approximation of system performance. An added advantage is that once a model has been developed, then the cost of obtaining results for that particular class of system is less compared to simulation, say.

On the other hand, the simplifying assumptions necessary to develop analytical models of complicated systems have got to be checked. One way of doing that is through the use of a detailed simulator. This is the technique adopted by the analytical modelling group at GEC Hirst Research Centre, where their simplifying assumption of nodal independence in their development of Mark II BL analytical model, is being checked by the use of the DMNSC simulation model (Chapter 7) developed as part of this research work.

Analytical models do not generally include a comprehensive set of operating system functions nor do they consider the quality of software performance. It is also difficult to include the random effects of multiprogramming and multiprocessing, and for some models, it is difficult to change the parameters for testing different aspects of the system. The development and particularly the revision of models is tedious and time consuming (HUES 67). In many cases, the entire system may be too complex for analytical modelling, giving the interactions between hardware, software, applications programs and a sophisticated interrupt handling structure. Simulation is then used in the detailed study of such complex systems.

3.7.3 Performance Evaluation through Simulation Modelling

The most potentially powerful and flexible of the computer-systems evaluation techniques is simulation, which provides a testing ground for and insight into the functioning of the system (CALI 67). It can be viewed as a combination of modelling and measurement. The process of simulating a computer system consists of building of a model of the system, a model of the workload and a simulation system. The simulation system organises the activities of the model as it evolves in simulated time. This aspect of simulation modelling is fully explored in Chapter 2. Here we are more concerned with the application of the technique in this particular problem area.

Since more details may be incorporated in a simulation, it is used as an extension to analytical modelling where a closed form expression cannot be obtained (LAVE 75) or to validate analytical models, as mentioned before. The level of detail that is difficult to incorporate in an analytical model includesfeatures such as dynamic memory allocation, interrupts in a multiprocessing environment and various system overheads. Moreover a simulation model does not have to use workload models described by stationary probability distributions only. Thus, studies of operating systems' storage allocation and scheduling strategies often require simulations. It is also the only method of estimating the performance of hypothetical systems and new designs before actually implementing them (KOSY 73).

In short, it has been used to fulfil all the objectives in (3.7.1). The ideal paramaterised simulator, capable of simulating any proposed computer system is not feasible because they differ so much in their organisation. However attempts have been made to develop models for a particular class of computer systems (UNG 72, NEIL 67).

Simulation as a technique has its pitfalls and problems that must be watched carefully so as not to render a costly simulation effort fruitless. An essential ingredient for a successful simulation is a clearly defined and agreed set of realisable objectives. These usually depend on answers to questions such as "What is to be learned about the system under study?" or "What decisions will be based on the simulation results?" Moreover, these objectives cannot be defined without the active participation of the end user. Defining the goals is the first step in any simulation project and perhaps the one most commonly bypassed.

For a simulation project to fulfil its objectives with the minimum time and effort, the simulation team must possess knowledge and experience in at least four areas: firstly project leadership to motivate, lead and manage. Secondly modelling skill to design a conceptual model that mimics the system under study at the appropriate level of detail, thirdly programming ability to transform the conceptual model into a readable, modifiable working program and lastly, sufficient understanding of the modelled system to guide the modelling effort and judge the validity of the simulation results. The model-building team must work with the user organisation from start to finish if both are to have the confidence and understanding necessary to use the completed work effectively.

A model is a simplified representation of a system, and it should incorporate only those features of the system that are important to attain the objectives. Too few details, render the model unreliable; too much detail and it will be costly both in development and use. (TEOR 73)

Selection of the appropriate simulation programming language is an important factor in the success or failure of a simulation project. The language has to be English-like, self documenting and readable by the user. It must support the proper concepts and be powerful enough to cope with the modelling requirements. This problem area has been thoroughly investigated in Chapter 2.

Verification and validation of the simulation model are essential to gain confidence in the model and its results for all parties concerned. This particular problem area is more thoroughly investigated in Chapter (6).

Failure to use modern software engineering tools and techniques (such as structured and modular programming) to manage the development of a large, complex computer program will result in long development time to the extent that it might be too late for the model to be of any use.

Other factors to bear in mind are the resources (man power and machine) required and available, ease of change of a model, total cost and experimental design (BELL 72).

In spite of all these problems and pitfalls, simulation of computer systems will remain the most general, most flexible and most powerful technique for studying and predicting system performance (SVOB 76).

3.7.4 Simulation in the Field of SPC Switching Systems

Before the era of SPC switching systems, the techniques used in the performance evaluation of the switching systems and communication networks in general were, the application of teletraffic engineering to the dimensioning and capacity studies of the switching systems, field trials, artifical traffic generators and traffic simulation systems (KOST 70, COLE 64, POVE 65). With the merging of computer and communication technologies, the techniques of computer systems are spilling over into the communication engineering field. Examples of this are the specification and description languages (BREA 79, KWA 79) and the simulation of computer systems applied to telecommunications processors.

Real-time environment simulators have been used recently to provide accurate models of SPC switching systems. The simulator is executed in real-time and simulates the whole environment of the system processors, with exact repetition of events as often as required. An environment simulator uses a separate processor to substitute all the elements of the SPC system except the system processors and their software. This includes the cross-points of the switching network, the junctors, peripheral devices, subscribers and trunks, all of which are represented by single bits in the simulator processor memory. This simulated environment is transparent to the system processors and the system state is changed by changes to those bits by the system processors and external events such as call arrivals.Figure (3.10) shows the schematics of a real SPC system and when the network and periphery are substituted by a simulator. The external events such as calls on subscribers or incoming lines, dialing, answer, release, *etc.* are simulated in the system by a call – pattern program (FONT 71).

Real-time environment simulators have been pioneered and implemented by BTM, a Belgian company of ITT, since 1966 for checking the software of SPC systems such as their METACONTA family of exchanges.

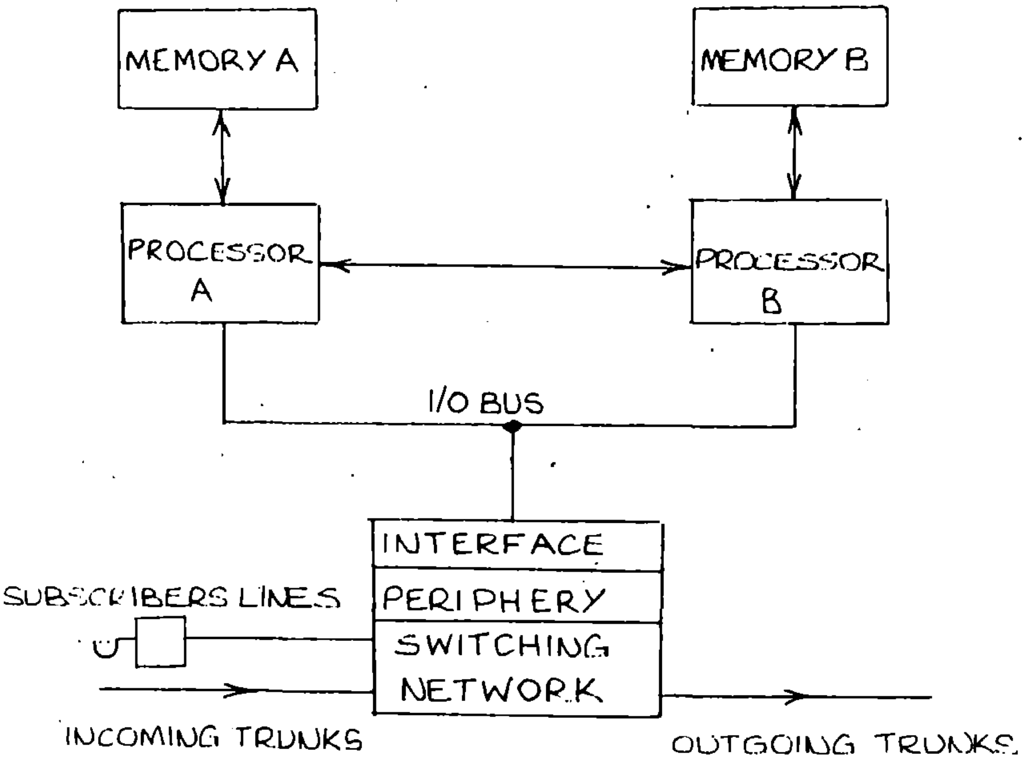


FIG (3.10.1) : SPC SWITCHING SYSTEM IN REAL STATE

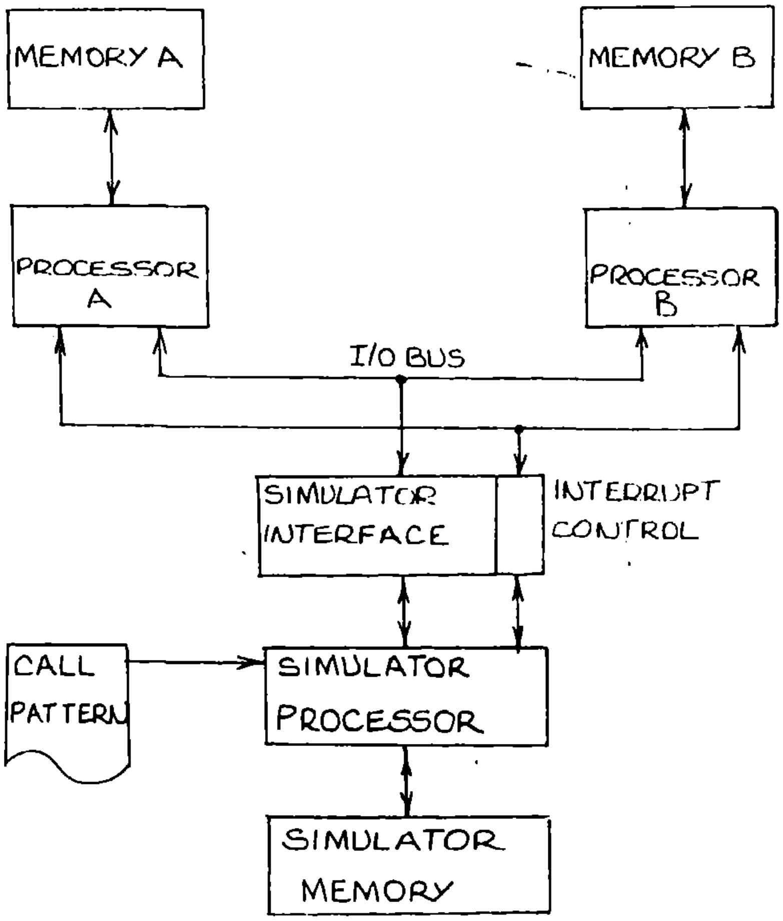


FIG (3.10.2) : SPC SWITCHING SYSTEM WHERE NETWORK AND PERIPHERY SUBSTITUTED BY ENVIRONMENTAL SIMULATOR

With relatively small additional programming and engineering effort, they could also be used for traffic studies of call handling capacity, testing of overload control strategies and the study of software efficiency under load conditions. The measurement programs provided in the system processors are written to suit the user administration requirements for statistics for the correct management of the system in the field, for traffic forecasting and for the measurement of the grade of service.

The calls are generated automatically by repetition of the basic call patterns determined beforehand for each type of call. Each time a call is generated, its parameters are modified. The mean inter-arrival time of each type of call is chosen separately in accordance with the traffic of each type and the call mix. Such types of simulators are reported to have simulated time : real-time ratio of the order 10:1 to 30:1 depending on the traffic (GRUS 76), (LAMP 79).

In an environment simulator, all processors (including the simulator processor) run under the control of one clock located in the simulator interface. By having only one clock in the system, all processors work in complete synchronization. Thus, if the initial conditions are the same, exact repetition of a given simulation is possible. The system allows the system processors to think that they run in real-time. This is achieved by stopping them whenever they receive information from or output information to the interface. When this occurs, the simulator receives an interrupt from the interface. During the time that the system processors are stopped, the real-time counter in the interface is also stopped, so that as far (as the control programs are concerned,) real-time is preserved.

This real-time counter is also stopped when required by the simulation program, and each time an interrupt is generated in the simulator by the interrupt control system located in the interface (every 1 ms), this interrupt initiates a program that updates the environment.

The main objective of an environment simulator remains the debugging and correction of program errors which takes more than half of the total effort spent in the production of real-time software. It is most useful when the hardware is not ready and the software engineers need to check their software, or when it is essential to repeat the environment conditions that generated a software fault. Other uses of environment simulation include evaluation and calibration of a processor and simulation of hardware faults for checkout of test and diagnostic programs. A reduction of up to 30% in debugging and correction time has been claimed by the use of environment simulators (GRUS 76, FONT 71).

Interestingly enough, environment simulators have spilled over to the field of computer engineering and computer science for example the environment simulator for the IBM System 360 (CHAR 78). This is yet another instance of the convergence and interaction of the two technologies of computers and communications, the initiative this time being taken in the communication engineering field.

One can trace the history of simulation of telecommunications traffic problems back to 1908. The manual methods and special purpose simulating machines (KOST 70) developed then were used until the early 1950's when digital computers and later simulation languages.were adopted. However, there has been little cross-pollination between the telecommunication traffic field and other areas hitherto.

That is to say, the development in simulation languages and techniques was not influenced by the nature of problems in the teletraffic engineering area, but rather by the nature of problems in other fields of Operational Research. That was so, because electromechanical switching systems were simple and the objective of a simulation study such as a grade of service or an average waiting time could be obtained conveniently with a roulette-type simulation which is simpler to construct and faster in operation.

But with the introduction of special purpose computers in the control of switching systems, the objectives of the simulation studies changed to testing of configurations, alternative scheduling strategies, waiting-time distributions and so on. This then called for simulations which included the time dimension (unlike roulette) and which vary in complexity and size according to the objectives of the study. These simulation models were built for particular SPC switching system as well as communication networks (DIET 75, BUSS 68, BERN 68, THOM 79, SCHM 79, YAN 78, FRAS 75, ANDE 72 *etc.*).

3.7.5 Mark II BL Simulator Package in Relation to Other SPC Systems Simulator Packages

As we have noted, before the era of SPC switching systems, traffic studies were undertaken using teletraffic theory or straight-forward and uncomplicated time-oriented or roulette simulations. Hence, there was no cross-pollination between this simulation application area and other areas which might have resulted in new simulation methodologies or approaches (KOST 70).

With the convergence of computer and communications as exemplified by SPC switching systems, the designers and analysts of such systems started to make use of performance evaluation and monitoring tools already being used in the computer technology field. (GERR 79, SEDG 70).

The simulation package reported in this thesis is an attempt to provide a CAD tool for the performance evaluation of a class of microprogrammed telecommunications - oriented multiprocessor system, with a difference. The difference is that it is meant to be an integrated package that simulates both the system processors and the exchange environment in a flexible level of detail and covers a whole range of System X exchanges.

This is in contrast with the previous simulation work in SPC field where the simulators were either environment simulators or concentrated on one aspect or sub-system only, such as traffic studies, networks and control-sub-system studies.

To provide such an integrated package, a hierarchical modular structure is adopted. Both of these features were exploited to the extreme by the choice of the powerful process-oriented simulation language SIMULA, with its CLASS and concatenation features (Chapter 2). These features enabled a one to one transformation of the system modules into their corresponding simulation modules in a multi-level fashion.

Multi-level simulation has been suggested before (ZURC 68). However, Zurcher et al.used the term multi-level modelling in a different context and meaning. What they suggested was an iterative method with the concurrent existence within a single model of several representations of the system being modelled at different levels of detail using an activity based simulation approach. The methodology and philosophy we suggested in this research for SPC systems simulation is primarily concerned with the inexperienced user who can build up his model of a particular SPC system from a library of models, where no more than one representation of a module exists. There are other fundamental differences between the two methodologies and these have been elaborated on in the Introduction Chapter (Chapter 1).

CHAPTER 4

THE GEC MARK II BL MULTI-PROCESSOR COMPUTER SYSTEM

4.1 INTRODUCTION

GEC Mk II BL communications processor is a powerful multiprocessor system which can perform all the functions required in the control of telecommunications switching networks. It has been designed to meet the operational requirements expected of normal switching systems with life expectancy well over 30 years. During its life span, a total switching system failure is expected with a very low probability indeed, and only limited periods of service degradation due to faults or errors is tolerated.

To meet such stringent requirements, the system designers have adopted the approach of using redundancy and dividing the hardware and software into modules with well defined interfaces, thus preventing the propagation of hardware and software faults.

In spite of the cost inherent in this approach, SPC systems must be cost-effective compared to conventional switching systems. To this end, the system is designed to be capable of controlling different types of switching functions within the network. The software organisation should allow both a realistic concentration of programming resources as well as the updating of existing software modules and the addition of new ones on-line while the system is carrying traffic. The hardware organisation should allow an initial reduced system to be used, which can grow in computing 'power' as the work load increases. The hardware must allow the benefits of technological advances to be exploited such as semiconductor technology both to enhance machine power and to counter the effects of component obsolescence. (WARD 72A)

In the GEC Mk II BL multiprocessor system, the store blocks and peripheral devices are shared between the CPUs. The computing load is shared equally between the CPUs, without assigning any specific function to any of the CPUs. Thus,if a program is interrupted in one CPU, it may be resumed later on a different CPU without affecting the program results. That is why the system is both a multiprocessor and a multiprogrammed system.

4.2 Mark II BL Hardware

The prime considerations in the system hardware design were security, reliability of operation and the ability of the system to grow in a modular fashion. Thus the system is divided into basic security modules with well-defined boundaries and interfaces. These basic modules are the CPUs, main random-access memory store blocks, input/output blocks, backing store, interrupt units and DMA multiplexers (Figure (4.1.))

If any fault is detected in a security unit, the entire unit is generally taken out of service. The number of redundant security units allows for the loss of one or even two units of any type without much service degradation.

Virtual addressing is used in a paged-store system. A two-stage translation is used to get the page physical address using firstly a table unique to the process, the process page table, and secondly the system page table. The status of the page is checked at the same time. For protection purposes, each software process is only allowed a limited range of virtual addresses and is restricted in the type of access to a given page within that range (read only, read/write or execute only). The effective address accessed by an instruction is the contents of the base register, plus the value

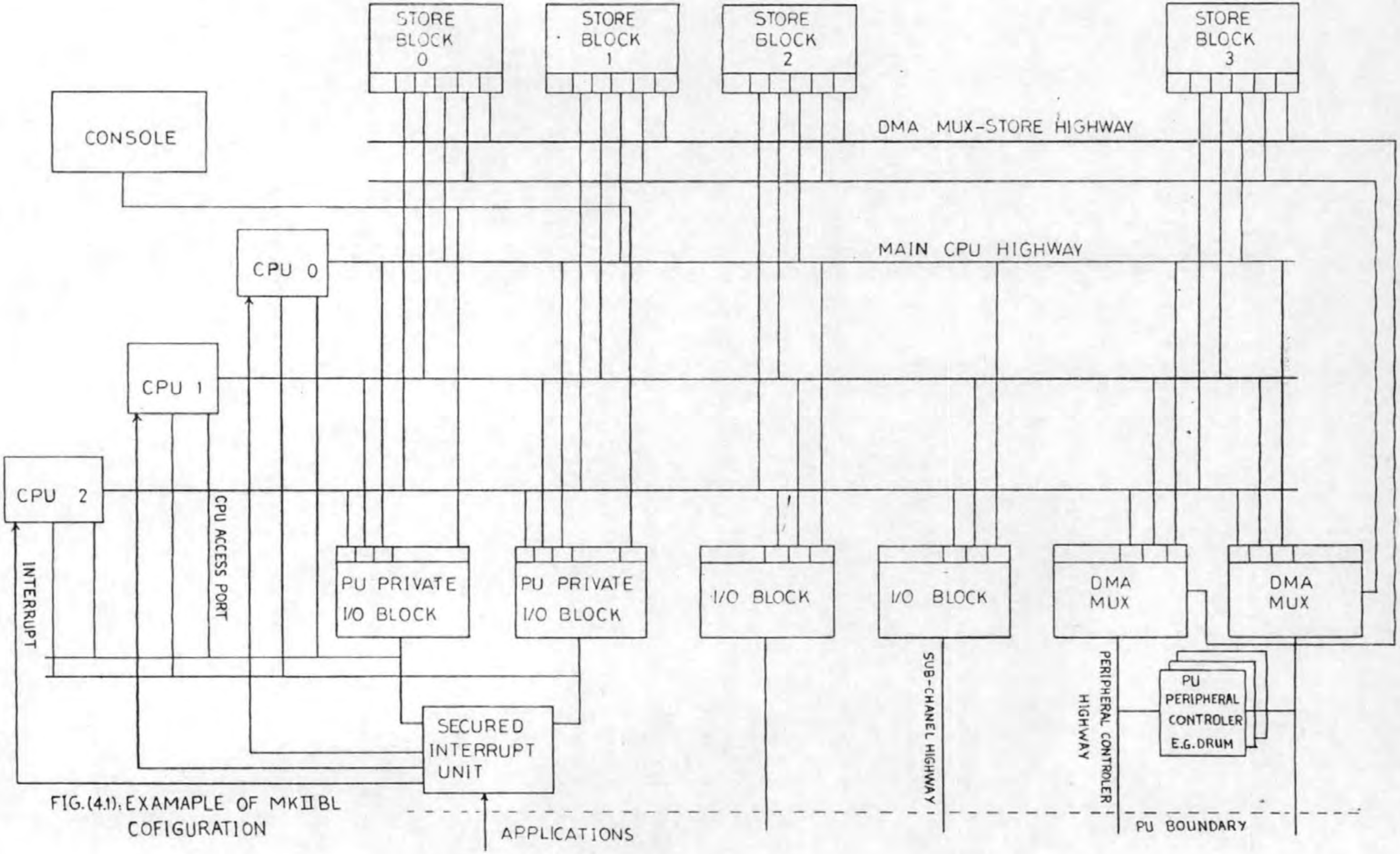


FIG.(4.1): EXAMAPLE OF MKIIBL COFIGURATION used in the index register (if used), plus the offset within the instruction. I/O channels are addressed in the same way as pages and hence the page protection mechanism applies.

As seen from Figure (4.1), the basic modules are the CPUs, main random-access store blocks, input/output blocks, backing store, the interrupt triplicates and DMA multiplexers.

Peripherals are of two types: those requiring direct CPU control are associated with an I/O block which can address up to 4K peripherals, they are connected by sub-channels. There are 256 sub-channels to a channel and sixteen channels to a block. Peripherals requiring DMA are associated with peripheral controllers which are connected to DMA multiplexers. Typical peripherals are the backing store units such as drums which require DMA, teletype writers, VDUs and the different exchange equipment components.

Mk II BL is characterised by a powerful instruction set compatible with higher-level languages. The instructions allow manipulation of individual bits and groups of bits, a desirable feature for efficient store usage and a feature not found in most computers. This is coupled with a large number of registers and register instructions which result in fewer store accesses, thus reduing the store contention and increasing the useful run time. There is a total of 16 'scratch pad' registers that can be manipulated by a programmer using register instructions. The store blocks have 16 access ports each. Thus, there is a limit of 16 on the number of CPUs and DMA multiplexers connected to the store block. The present Mk II BL version can have a maximum of 11 CPUs.

For security reasons, to make sure that faulty CPUs do not corrupt large areas of memory and that a fault in a store block is limited to that block, each store block has its own power supplies and sequencing control. This asynchronous mode of operation of the store blocks allows store blocks of different speeds to work side by side without having to force the faster blocks to work at the speed of the slower ones as in some systems. Each word consists of two 8-bit bytes with a parity bit per byte. A store block performs parity checks on addresses as well as data.

If a CPU or DMA multiplexer wishes to access a store block, the address (and data) is sent together with a request signal to the appropriate store block. The selection logic of the store block chooses which request to honour in the case of simultaneous accesses on a "round-robin" basis. It blocks the other requests and activates the microprogram to service the request.

The store blocks are either core-stores or semiconductor stores. Due to the semiconductor stores having the advantages of lower cost, higher speed and packing density they are becoming the more attractive form of main memory. The volatility problem of semiconductor memories is overcome by using ROM or PROM to store the essential parts of the operating system that deal with re-start and re-load from backing store. An added advantage in this case is the protection of the vital programs and data against corruption by noise transients and software bugs. This 4K memory module is provided on a per-CPU basis and treated as part of the CPU security unit. However, the addressing and protocol are the same as for the main memory blocks.

The protocol between CPUs and I/O blocks is the same as that between CPUs and store blocks. I/O blocks are used for communication between CPUs and peripherals. There are three types of I/O blocks in the system:-

(i) Application I/O block
(ii) PU private I/O block
(iii) DMA multiplexer I/O block

An application I/O block allows addressing up to 4K peripheral units. The peripherals are divided into 16 application groups (channels) each group having up to 256 peripherals (sub-channel). Thus to a CPU a channel is equivalent to a page and a sub-channel to a word.

The private I/O block gives CPU access to CPU access ports, the interrupt unit, teletypes and paper tape units.

A DMA multiplexer I/O block is the same as an application one except that some buffer boards not required are removed because of the proximity of the DMA multiplexer with its own I/O block.

The DMA multiplexers are capable of addressing and selecting up to 240 peripheral controllers. The peripherals can be connected to both I/O channels for direct CPU control, and to peripheral controllers for DMA. To save on channel equipment, the direct CPU control can be performed via the DMA multiplexer. The multiplexer will have its own I/O address within this block, so that the CPU can address the DMA multiplexer itself. Neglecting parity bits, the address highways from CPUs to the store and I/O blocks are 20 bits wide and the DMA multiplexer address highways are 12 bits wide. Separate in and out data highways each transmit two bytes with a parity bit per byte.

The implementation of interrupt facilities for a multiprocessor system has its special problems (WARD 72A). The interrupt system must be independent of the CPUs and be capable of deciding which CPU to interrupt. The interrupt unit is triplicated for security reasons. The interrupt system is used to stop a CPU from running a process and to cause it to access external devices when they need service. Devices send their interrupts to the interrupt unit where they are detected and stored as levels in appropriate groups. There is a maximum of 8 groups and 16 levels to a group.

Interrupt signals to the interrupt system are classified as immediate or non-immediate. Immediate interrupts are processed immediately while the non-immediate ones have to wait for the arrival of an immediate interrupt. The real-time clock is an example of the immediate interrupt. The longest time that a non-immediate interrupt has to wait is a clock period (10 msec).

The interrupt unit contains a register that holds the identity of the CPU running the lowest priority process (LCPU). This register is updated by the process allocators when scheduling processes to run in their respective CPUs. When an immediate interrupt arrives,the interrupt system sends an interrupt to the LCPU and starts a time-out. If a reset signal is not received before the time-out expires, the interrupt system notes this fact and sends an interrupt to another CPU. Attempts are made on CPUs cyclically until a CPU services both the immediate and non-immediate triggered levels.

All highways and registers in a CPU are 18 bits wide so that they contain a parity-bit per byte for security. The CPU contains an exceptionally large number of registers not found in comparable commercial computers. Registers can be divided into two groups; scratch pad registers and control registers. Scratch pad registers are those that can be addressed by a programmer and there are 16 of them. The arithmetic unit performs arithmetic and logical operations. These include addition, subtraction and multiplication with division as an optional extra, inversion, AND, OR, exclusive OR and 'INVERT and AND'. It also performs bit manipulation,shifting and searching.

The heart of the CPU is the microprogram. It interprets the instructions and provides the control stimuli to the highways, registers and the arithmatic units. It also performs virtual— to physical address translation and contains the process allocator, the central part of the operating system to be described later.

The functions of the DMA multiplexer can be summarised as,firstly, allowing peripherals to access main memory through their peripheral controllers without the need for having separate ports on each store for each peripheral controller. In this sense,the multiplexer acts as a concentrator for the DMA traffic. Secondly, the controller allows individual CPUs to input directly to peripheral controllers, without the need for the controllers to have separate ports for each CPU. In this respect,it is acting as an extension unit on the store highway. Thirdly, it passes on interrupt requests from the peripherals through their controllers to the interrupt unit. Thus,the multiplexer has interfaces to the main store blocks, the peripheral controllers, highway, CPUs and the interrupt unit. It is microprogrammed and at least two multiplexers are provided in an installation.

The peripheral controllers are application-dependent devices, acting as an interface between the application requirements and the DMA multiplexer protocols-as for example in the drum controller.

For backing store on the Mark II BL system, fixed-head-per-track magnetic drums or discs are provided. Each drum is divided into 256 tracks with 8 sectors per track. A sector holds a page of information. The size of the backing store normally depends on the application.. For example, two small 1.5 Mbit drums were sufficient for the CCITT No. 6 signalling system. All the fetches and dumps are done via the peripheral controller and DMA multiplexer.

For security purposes extensive fault diagnosis and recovery facilities are provided. A console is provided with each CPU to monitor the various register contents (including microprogram registers) and it has extensive facilities to aid in fault diagnosis. A fault channel is also provided for automatic recovery. The trap process in the faulty CPU enables this channel and sends an interrupt to the system. The CPU servicing the interrupt then runs extensive test routines and either takes the CPU out of service or restarts it.

Preprocessors may be added to the Mark II BL system to increase the computing capacity, increase the efficiency by reducing store contention and increase the mean time between system failures where a more distributed system is preferable in this respect. Whether an installation has no preprocessors, Mark II BL preprocessors or Mark IP preprocessors, depends on the size and nature of the particular application. Preprocessors are incorporated as modules into the Mark II BL system by connecting them directly or indirectly to the DMA multiplexers. The ability to add preprocessors

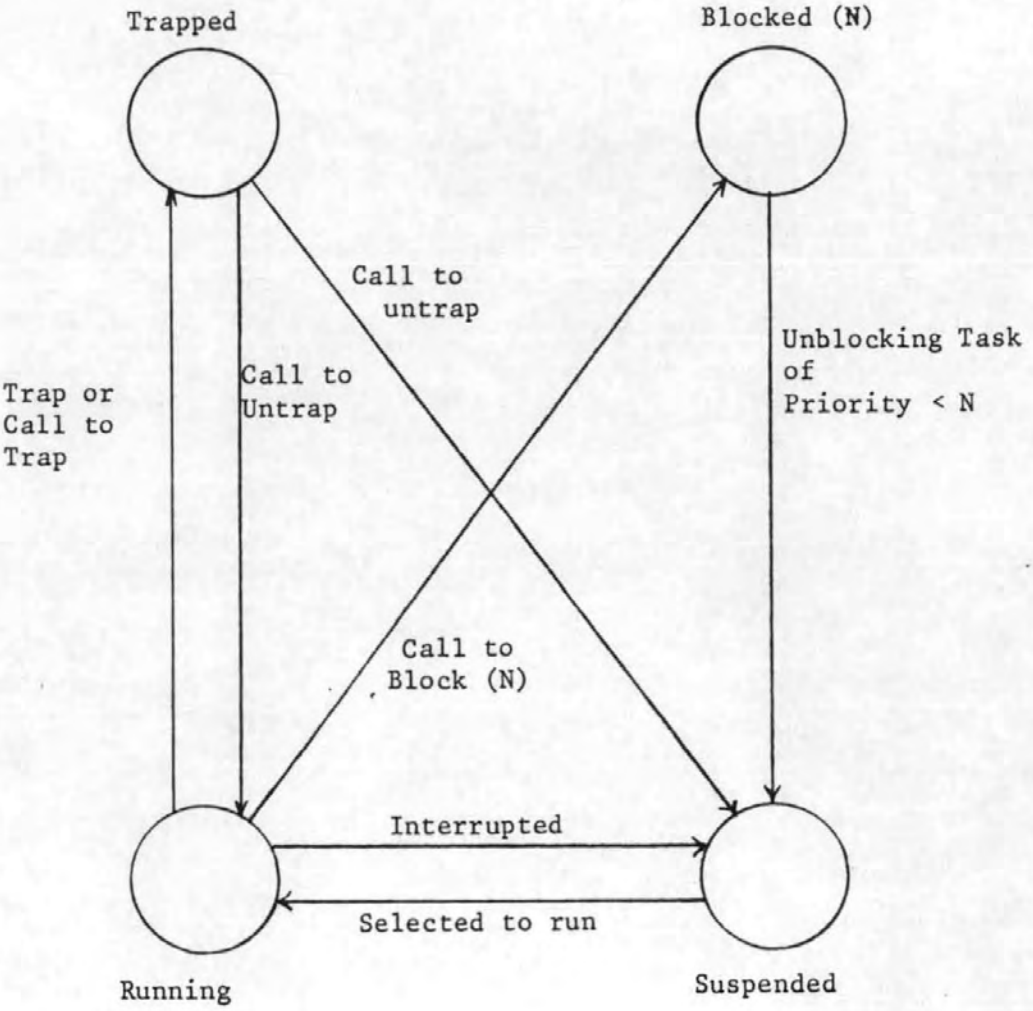


FIGURE (4.2) : PROCESS STATES

to the system gives the necessary flexibility in the range of computing power required by the diversity of applications for Mark II BL system intended by the British Post Office (GEC 75A).

4.3 Mark II BL Software (Figure (4.7))

4.3.1 Introduction

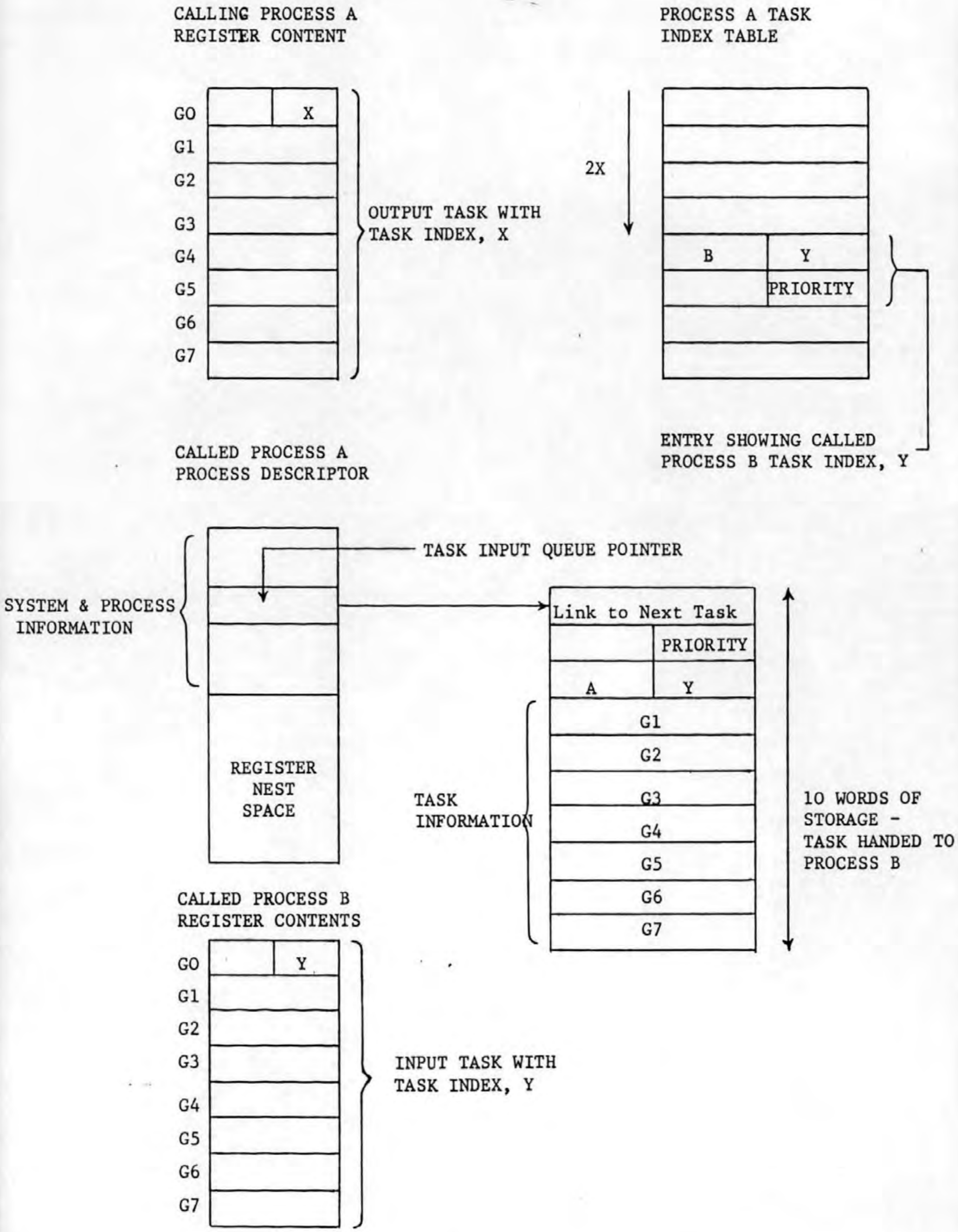
A modular approach is adopted in the design of Mark II BL Software as well as its hardware. Modules are processes which can be defined as entities with a unique priority and protection status. Processes are non-re-entrant, that is to say a process cannot be run in more than one CPU at the same time, and will normally have dedicated program and data which it can only use, though it may share program and fixed data with other processes. Generally, for security reasons, sharing of working storage between processes is not desirable. Processes are futher divided into either operating system or application processes.

The operating system is of the inter communicating-processes type. A clear boundary is defined and maintained between the operating system and the application software. By so doing, the operating system can be used throughout the range of applications without modification. This, also, helps to define a clear line of responsibility between those who design the operating system and those who design the application software.

Processes are function-oriented such that each process has a function or a group of functions to carry on a range of data.

In a telephony application, information about individual telephone calls is passed in tasks (Figure (4.3)) from one process to another to enable each process to perform its functions.

FIGURE (4.3) : MK II BL TASKING SYSTEM



4.3.2 The Real Time Operating System

The operating system can generally be divided into two parts: the real-time operating system which runs on-line with the application software and the 'support software' used for software development. The 'support software'(which includes items such as a compiler/assembler, a linker, a lister, loaders and packages for debugging)may not be required permanently on site and is not subject to the same timing constraints as the real-time operating system.

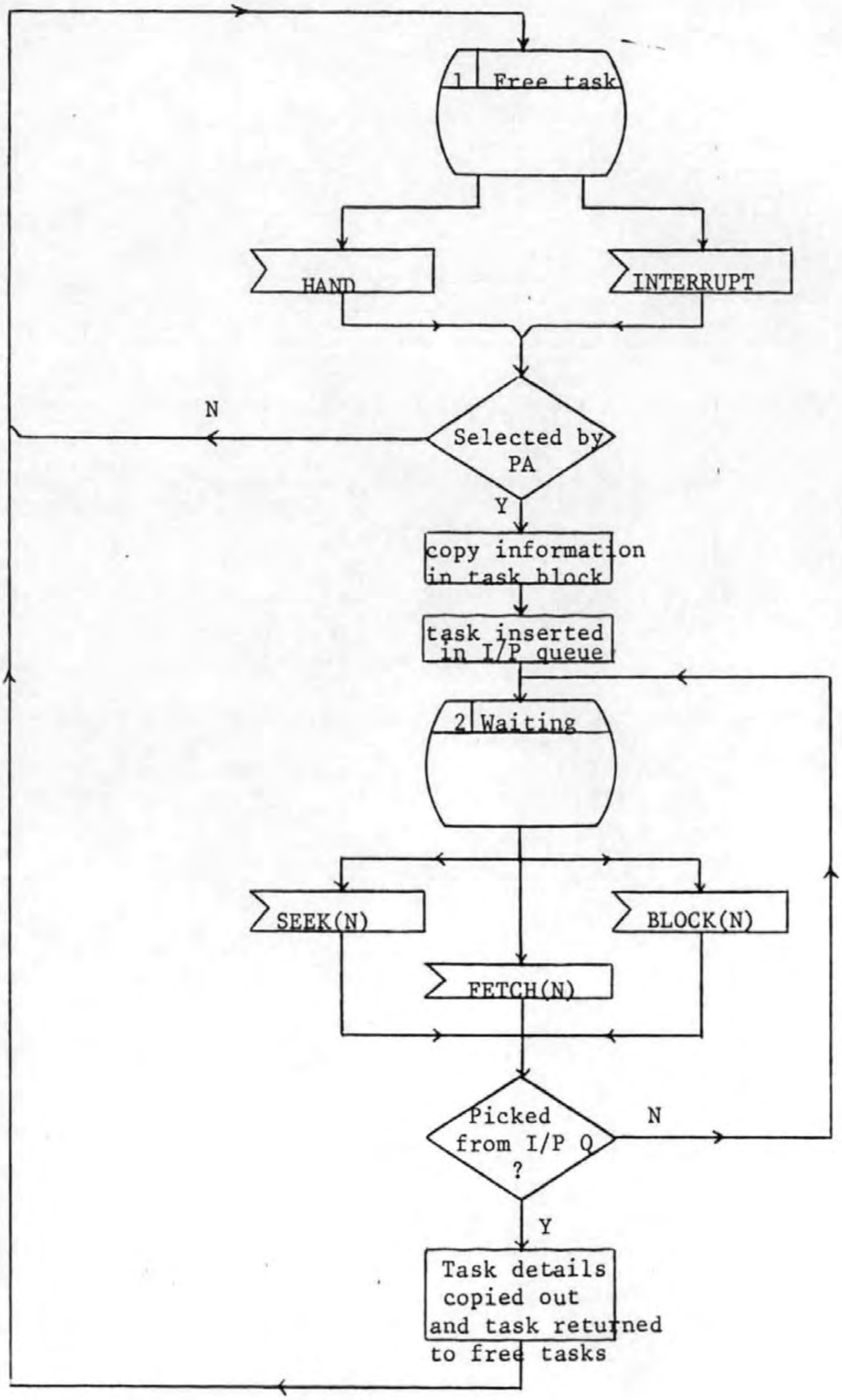
The real-time operating system provides functions such as timing, store allocation, CPU and system scheduling, input - output buffering, message printing, diagnosis of process utility faults, as well as permitting a modular organisation of the software.

In its widest sense, the operating system includes the microprogram, and hence includes the process allocator which is implemented in microprogram. The rest of the operating system is organised into processes whose programs are stored in the main memory, backing store or private ROM per CPU (or a combination of these). In what follows, the major components and functioning of the real-time operating system is explained in more detail.

4.3.2.1 The Process Allocator

The process allocator is at the heart of the real-time operating system. It is concerned with the scheduling of the software processes to run on different CPUs, the communication between those processes and the servicing of interrupts in the complex multi-processor system environment (GEC 76A).

FIGURE (4.4) : A TASK BLOCK STATE TRANSITION DIAGRAM


1 Free task
HAND
INTERRUPT
N
Selected by PA
Y
copy information in task block
task inserted in I/P queue
2 Waiting
SEEK(N)
BLOCK(N)
FETCH(N)
Picked from I/P Q ?
N
Y
Task details copied out and task returned to free tasks

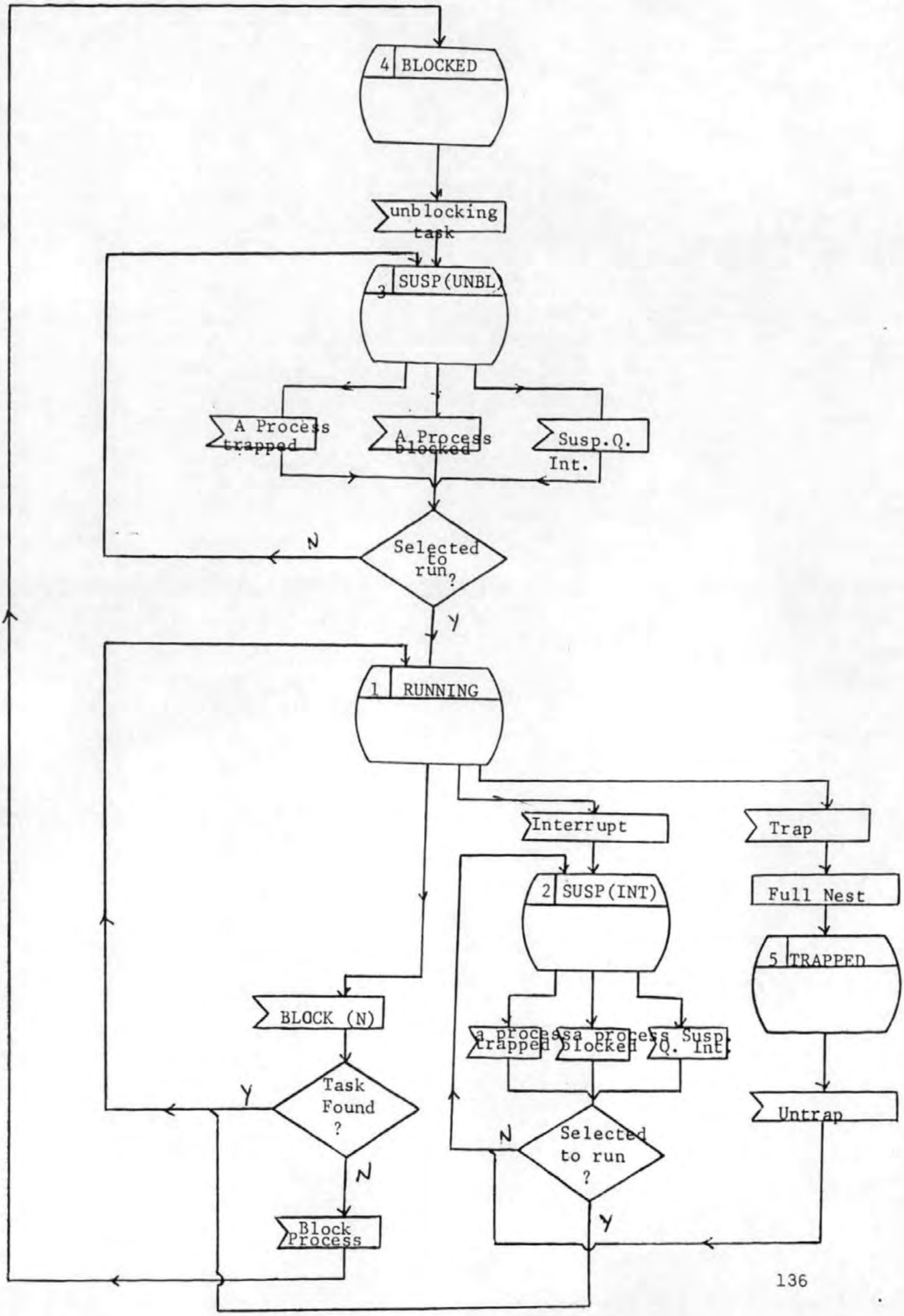
The process allocator is entered either as a result of a 'process allocator call' by a running process or as a result of a hardware interrupt mechanism. Interrupts are triggered by chosen peripheral devices when they require attention and by the process allocator itself when intending to schedule the system as a whole.

PROCESS STATES:

Processes in the system pass through a number of states while they are performing their processing functions. Basically, a process is always in one of four states (Figure 4.2). When a process has its own register contents set in the registers of a CPU, it is in the running state and will be executing instructions. A change of state is most likely to occur when a process executes a call to the process allocator or an interrupt occurs and is steered by the interrupt unit to the CPU in which the process is running. A process will 'call to block' when it is waiting for the arrival of a particular event which will be signalled by the arrival of a task of a particular priority at its input queue. In this case, provided that the task has not arrived yet, the process allocator will change the state of the process from running to blocked with a parameter between 0 and 15 to indicate the priority of the expected unblocking task. Thus, there are virtually fifteen states within the blocked state, that is from block (0) to block (15)

When an interrupt occurs on its CPU, the process state will be changed from running to suspended (Interrupted). When an unblocking task occurs, the process will then be changed from the blocked state to suspended (Unblocked). The differentiation between the two types of suspended state determines the degree of de-nesting later when the process allocator decides to run the process. A process is also

FIGURE (4.5) : A PROCESS STATE TRANSITION DIAGRAM



transformed to the suspended state when another process calls the process allocator to untrap it. Finally, a process state is changed from Running to Trapped if a trap is detected by the process allocator or the process calls to be trapped.

TASKS:

The process allocator maintains a pool of blocks of words, each block is 10 words long, and it uses these blocks as message or task carriers between the different processes (Fig. 4.3). The process sending the task is called the 'calling process' and that receiving it 'the called process'. The first word in the task block is used to store its location with respect to other task blocks. The second word stores the task priority which determines its position in the process input queue.

The third word is used to store an index supplied by the calling process and referred to as the incoming task index. This index is used by the process allocator to index down a per-process table to extract information such as the identity of the called process, the task priority and the value of the incoming task index which will be passed to the called process. Thus, in general, the task index changes as the task is handed from one process to another. This gives processes the freedom to choose task indices best suited to them and reduces the chances of changes in one process affecting the others. The rest of the task block (Words 4 - 10) are used to store the information written by the process into the general registers G1 - G7.

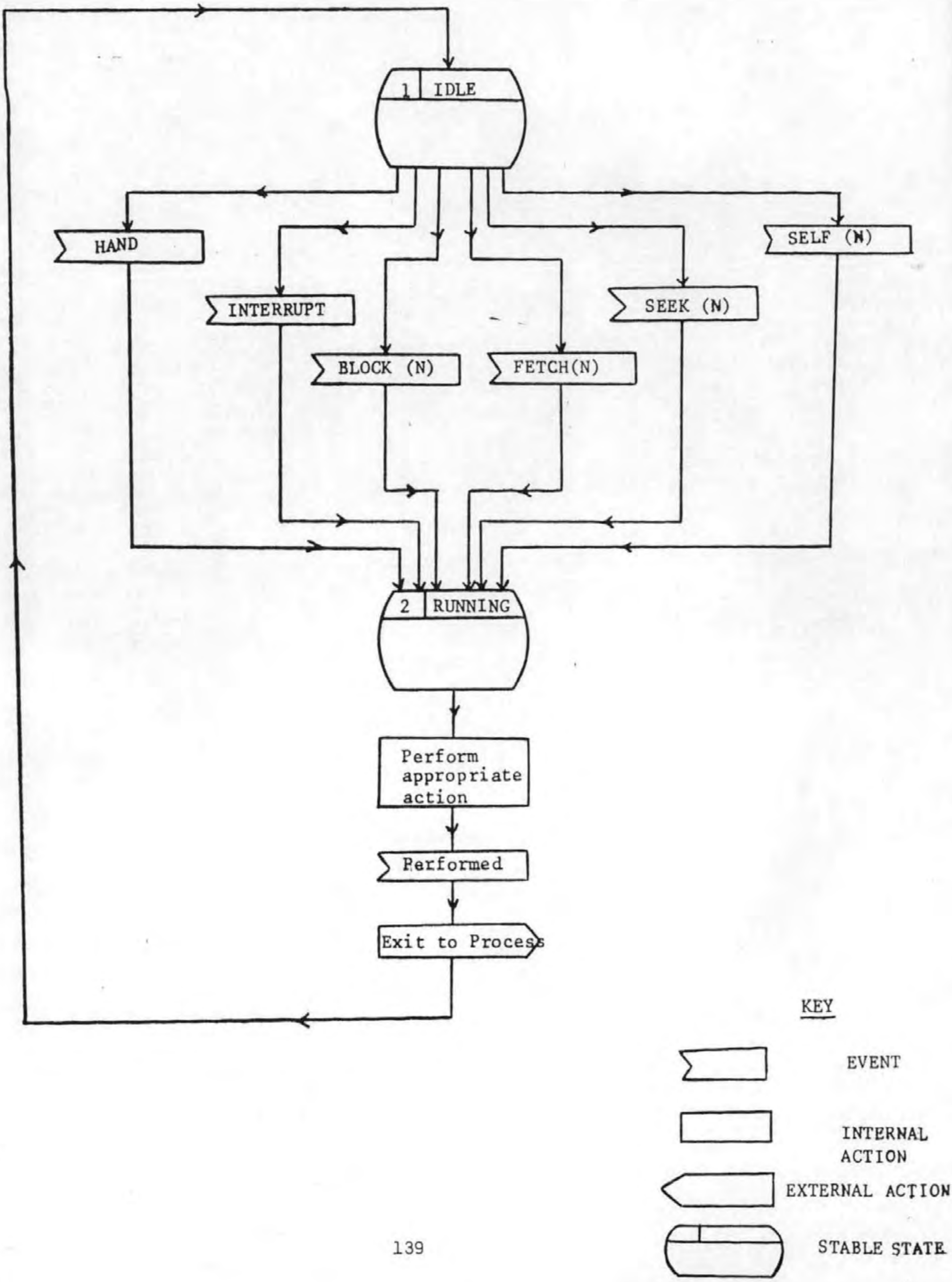
CALLS TO THE PROCESS ALLOCATOR (Refer to Figures (4.6.1) and (4.6.2))

A process wishing to pass information to another process, will load the information into the general registers G1 - G7, loads an outgoing task index in G0 and executes the instruction HAND. This will activate the powerful microprogram sequence of the process allocator. The process allocator then links a free task block out of the free tasks list and stores the information as mentioned under tasks. It then links the task to the process input queue at a point behind all tasks of equal or higher priority. Priorities are in the range 0 - 15; the lower the number, the higher the priority. If the called process is in the blocked state and the task just handed is unblocking, then the process state will be changed from blocked to suspended (unblocked). It will also be inserted in the suspended state map so that it may be selected to run later. The process allocator then, checks the priority of processes running on other CPUs. If the priority of the process just suspended is higher than the priority of any of the running processes, the process allocator will trigger a special interrupt level, known as the 'suspended queue interrupt', in the interrupt triplicates unit. This will force the multiprocessor system to re-schedule itself,as will be explained later in the interrupt handling sequence of the process allocator.

If a running process requires a task of a minimum lower priority to be linked out of its input queue and loaded into the general registers, it will execute FETCH(N). The process allocator will check the priority of the tasks in the input queue. If a task of priority higher or equal to N is found, it will be linked out of the input queue.

The information will be loaded into the general registers and the task block returned to the pool of free task blocks. The process allocator then sets the condition codes to 1 or 0 according to whether the required task is found or not, and exits back to the process.

FIGURE (4-6-1) : PROCESS ALLOCATOR STATE TRANSITION DIAGRAM (STD) - LEVEL 0



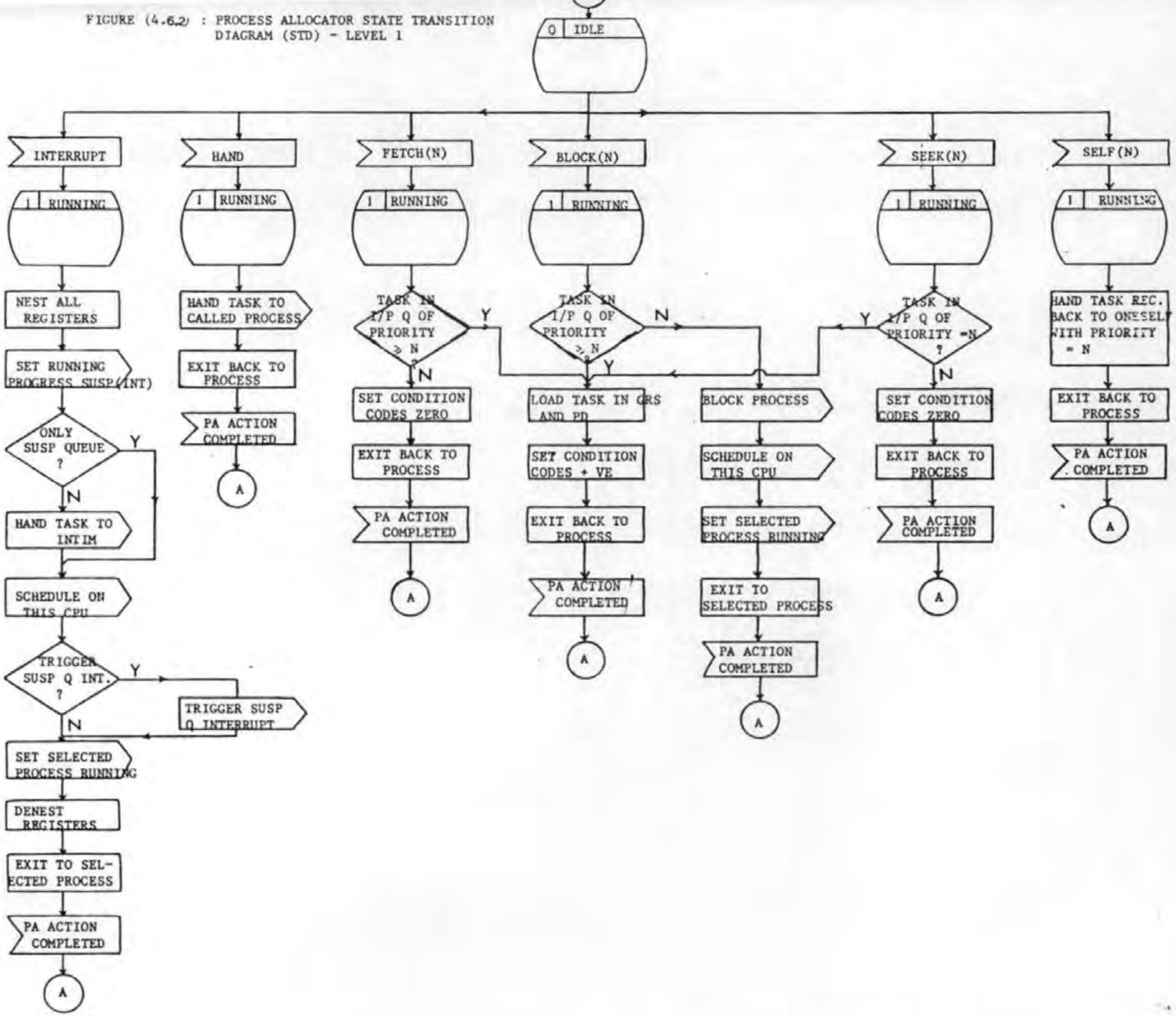
Thus, the calling process will always continue from the next instruction and will know whether the task is fetched or not by scanning the condition codes.

SEEK(N) is identical to FETCH(N), except that a task of a particular priority, N, is required in this case. As tasks are ordered according to their priorities, the process allocator has to examine the whole queue of tasks to see if the requested one exists. In case of FETCH(N), only the first task in the queue need be examined.

If a process executes BLOCK(N) and a task of priority greater or equal to N is found, the sequence is the same as that for FETCH(N). If, however, there is no task in its input queue of priority at least N, the process allocator nests the process's non-general registers into its process descriptor and sets it blocked at level N. It will remain blocked until it is handed a task of priority at least equal to N. While it is blocked, it is not allowed to run on any CPU. The process allocator will then select another process to run in its place, from the pool of suspended process.

A process may hand a task to itself using the instruction SELF(N) where N is the priority of the task. The called process is identical to the calling process. The incoming task index is copied straight from the outgoing task index. Hence, the process allocator does not need to access the process's task index table. The process has, also the other option of handing a task to itself using the instruction Hand where the translation using the TIT (Task Index Table) will result in a called process index identical to that of the calling process.

FIGURE (4.6.2) : PROCESS ALLOCATOR STATE TRANSITION DIAGRAM (STD) - LEVEL 1



A process may also call to TRAP. The process allocator then nest all the registers (scratch pad) and sets the state of the process to trapped in the process descriptor. A special task will be loaded and handed to the storage allocator or the trap process according to the nature of the trap. The process allocator then re-schedules on this CPU. Since the process load does not increase, there is no need for system scheduling using the suspended queue interrupt mechanism.

Processes in master mode may call the process allocator to UNTRAP another process. When the process allocator is entered, it checks whether the calling process is in master mode; if not it branches to the trap sequence mentioned above. The state word of the trapped process is set to suspended (interrupted) in its process descriptor and the process is included in the suspended state map. The process allocator then performs a system scheduling.

In general, processes are either periodic or non-periodic (or aperiodic) A non-periodic process will pick up all its tasks using a BLOCK(N) instruction. Thus, subject to CPU availability, it will process its tasks as they arrive. To reduce the process allocator Lockouts occupancy, several processes are made periodic. Such a process will handle all its tasks when running. When its input queue becomes empty it is blocked and incoming tasks are piled up into its input queue and not handled until the process is unblocked by the arrival of a periodic unblocking task.

TASKS TO AND FROM PERIPHERAL PROCESSES

Tasks handed from processes in the processor utility (PU) to peripheral processes are indicated to the process allocator by the fact that the called process index is out of range. The process allocator then links the task to the peripheral process input queue and performs

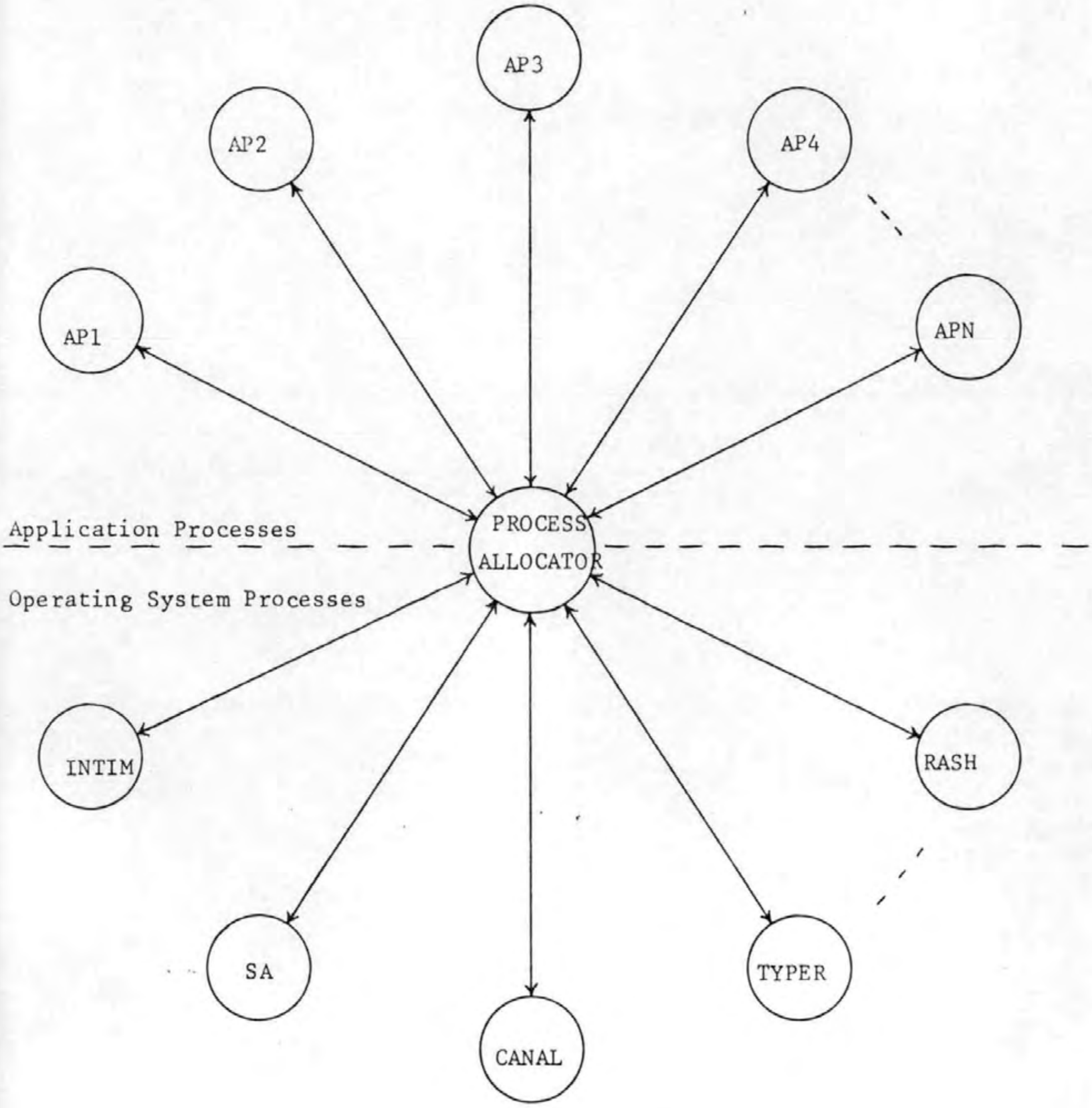


FIGURE (4.7) : MK II BL SOFTWARE STRUCTURE a'kick-off' I/O. The process allocator performs this 'kick-off' I/O to cause the peripheral process (or controller) to search down its queue for new tasks. When the peripheral controller wishes to hand a task back to a PU process it alters the task details, writes in a new outgoing task index, clears the priority word and triggers an interrupt. When the process allocator is serving an interrupt it will look down the input queues of the peripheral controllers corresponding to the triggered levels. On finding a task with cleared priority, the process allocator links it out of the input queue and hands it to the appropriate PU process using the outgoing task index to index down the peripheral controller task index table.

TASKS BETWEEN THE PU AND BACK-UP STORAGE

Mark II BL storage devices,such as drums or discs,have peripheral controllers associated with them. Some PU processes such as the storage allocator, communicate with these by passing tasks to effect the transfer of information between working and backing storage. The peripheral controller uses a DMA multiplexer to effect the transfer which will trigger an interrupt when it is completed.

TASKS BETWEEN THE PU AND PPU

To the PU,the PPU looks like a peripheral controller with one or more peripheral processes. The peripheral controller responds to the 'kick-off' I/O and triggers an interrupt to the PPU. The PPU process allocator performs I/Os to load the task into registers and uses the task index table to index down a TIT in its own data (one TIT per peripheral process) to hand the task to a process on the PPU.

INTERRUPT SERVICING

In the interrupt unit there exist interrupt levels grouped into four groups 0 - 3. Group 0 comprises the DMA interrupt levels which are dealt with by the process allocator itself. For Group 1 (Timeouts and Clock), Group 2 (CPU and DMA faults) and Group 3 (Man/machine) the process allocator hands these to INTIM process to deal with. They are called ordinary interrupts.

Interrupts are also classified as immediate or non-immediate interrupts, regardless of whether they are DMA or ordinary interrupts. An example of an immediate interrupt, is the clock interrupt which is triggered every 10 msec and the suspended queue interrupt.

When an immediate interrupt is triggered, the interrupt unit sends a signal to the CPU running the lowest priority running process. The process allocator in the interrupted CPU finishes performing its current instruction and branches to its interrupt servicing sequence. The process allocator nests all the registers of the current running process and sets its state to suspended (interrupted) in its process descriptor. It will also include it in the suspended state map. The process allocator then performs I/Os to staticise the triggered interrupt levels as bits in the CPU registers. It then checks whether any of the DMA levels are triggered and if so services them. The ordinary interrupt levels are stored in a task and handed to the INTIM process. The process allocator then selects the highest priority suspended process to run on this CPU, deletes it from the suspended state map and sets its state to running in its process descriptor.

According to whether the selected process was suspended (interrupted) or suspended (unblocked), the process allocator de-nests the general registers from its process descriptor or loads the first task from its input queue respectively. In either case, the non-general registers which store the environment of the process are de-nested from its process descriptor.

The interrupt is steered by the interrupt unit to the CPU running the lowest priority process. This is known as the LCPU. Whenever this value changes, the process allocator involved must inform the interrupt unit.

After sending the interrupt to LCPU, the interrupt unit starts a time-out. If LCPU does not send a reset within this time-out, the interrupt triplicates unit sends an interrupt to the next CPU and starts the time-out. This pattern continues until one CPU responds within the time-out. When a CPU responds, it checks whether it is actually the LCPU. If not, it services the interrupt in the normal way, but hands a modified ordinary task to INTIM specifying the value of LCPU.

SCHEDULING

A main function of the process allocator is to ensure that a process is allowed to run only when all higher-priority processes have run.

In the process allocator data, a two-dimensional bit array defines the indices of processes eligible to run on that CPU. The process allocator always checks this when selecting a process to run. In general, the majority of processes may run on any CPU. However, TRAP processes whose function is to investigate CPU faults and background processes to monitor store locations are assigned on a fixed per-CPU basis.

When a running process is blocked, that is to say it has executed a BLOCK(N) instruction and a task of priority N or higher is not found, or when it is trapped or interrupted, the process allocator must select another process to run in its place. It does this by searching down a bit map where the bit position indicates the value of the process index (The Suspended State Map). Finding a suspended process, the process allocator then checks whether that process is allowed to run on that CPU, and if not, continues searching until it finds one that is allowed. The process allocator then sets to zero, the corresponding bit in the suspended state map and changes the state of the process to running in the process's process descriptor. It then compares all the CPUs, finds the new value of LCPU and sends it to the interrupt unit. The general registers are de-nested from the process descriptor if the process was suspended (interrupted) and a fresh task loaded into the general registers if it was suspended (unblocked). Regardless of the type of suspended state, the non-general registers, which store the process environment are always de-nested from the process descriptor.

When a process allocator executes a HAND instruction, services an interrupt or untraps a process, it is highly probable that a high-priority process becomes suspended. It is desirable to get it running as soon as possible instead of waiting for a lower priority process to be blocked. For this purpose, a facility is provided for a process allocator to trigger a special interrupt the 'suspended queue interrupt'. The process allocator compares the priority of the highest priority suspended process with that of the process running on LCPU. If it is higher, it triggers the 'suspended queue interrupt'. This is an immediate interrupt and is steered to LCPU, forcing its process allocator to re-schedule. This sequence is repeated if necessary until there is no suspended process of priority higher than that running on LCPU. This system scheduling using the 'suspended queue interrupt' mechanism is, therefore, usually employed when the process load increases.

OVERLOAD CONTROL

This is another of the process allocator responsibilities. There are two distinguishable overload conditions: system overload and process overload. A system overload occurs with high traffic when it is not able to handle calls at the rate they are arriving. This can, ultimately, lead to system rollback when the process allocator free task list becomes empty. A process overload occurs when a process cannot process incoming tasks at their arrival rate, either because of its low priority or a hardware fault. For example, if the storage allocator receives a large number of tasks when a store block fails its input queue length increases considerably.

When the process allocator detects an overload condition, it hands the task that causes the overload noramlly, but also hands a special task to an overload control process. Subsequently, when the process input queue drops below a certain level or the free task list grows above a certain level, the process allocator sends another task to the overload control process to discontinue its remedial action.

LOCKOUTS

Lockout or semaphores are used to protect the critical data of the process allocator from simultaneous illegal accesses. Three lockouts exist within the process allocator. The suspended state lockout, SUSPLO, is held while including a process in the suspended state map or when removing one from it.

The free-task queue lockout, FREELO, is engaged while removing a free task block from or inserting a task block into the free task list. The interrupt lockout, INTLO, is engaged when accessing the interrupt triplicates unit to identify the interrupted levels, updating the value of LCPU or triggering the suspended queue interrupt.

PROLO, is a per-process lockout and is engaged while linking a task into or out of the process's input queue. PROLO is also engaged while interrogating the state of the process and its input queue and while taking it into or out of a blocked state.

It is crucial that lockouts are held to the minimum. One of the objectives of simulating the process allocator is, in fact, to study the lockouts' occupancy under different circumstances, as it is crucial to the optimum performance of the process allocator that the lockouts are held for the minimum possible time.

4.3.2.2 The Interrupt and Timing Process (INTIM)

The responsibility of this real-time operating system process is to deal with external interrupts (as opposed to traps) and associated timing. This can be stated more precisely as (GEC 75B):-

a) Timing and Interrupt Handling

INTIM maintains a calendar which is constantly being updated on clock interrupts. In response to timing tasks, INTIM sends time data in the response task. INTIM uses the clock interrupt to perform timing operations for processes. For example a process may request a response task to be sent after a specified time interval or the occurrence of a particular interrupt. A process may also request periodic response tasks with specified periodicity to be sent or request to cancel the actions requested by tasks involving timing.

b) Interrupt Unit diagnosis and re-configuration

Whenever a process allocator decides that a fault has occurred in the interrupt system, it sends a fault task to INTIM which will identify the cause and isolate the faulty unit among the interrupt triplicates and informs the process allocator.

c) I/O Devices handling

I/O man/machine communication is handled by INTIM in conjunction with two other processes, TYPER and CANAL, that is INTIM simulates DMA for man/machine communication. INTIM services I/O device interrupts and passes the characters input in a task to the CANAL process. Output tasks specifying device number and address of the buffer of characters for output from TYPER processes are sent to INTIM. The characters are outputted to the appropriate device under interrupt control. Thus, when a process wants to wait for an interrupt, say, it sends a task to INTIM specifying the source of interrupt and generally, a covering time-out, so that if the interrupt does not occur within this time-out, the process will be informed. The process mav call to block waiting for the task such as a periodic process. In this case, when the interrupt occurs, the process allocator forms a task containing the identities of processes waiting for that interrupt and hands that task to INTIM which unblocks the specified processes.

Timing of the actions of a process can be achieved in a number of ways. In periodic processes, timing is generally done by counting the initiations of the process. In a non-periodic process timing can be done by the use of clock tasks. A further method of timing is available through the observation of a real-time clock count, maintained by INTIM.

4.3.2.3 The Storage Allocator

This real-time operating system process is responsible for the storage management. Generally, one can identify three main functions for this process.

One of its most important functions is to deal with requests for pages not in main memory. A process will be trapped, that is to say nested and stopped running, if it tries to access a page not in memory. The process allocator then hands a task with the page details to the storage allocator. The storage allocator accesses the system page table to see whether memory space has been allocated to this page. If not, an overwrite search is commenced to select a page to be overwritten. If the selected page status is read/written it has to be dumped to the drum. The storage allocator then generates a task containing the necessary track and sector information and hands the task to the appropriate drum using the DMA task facility of the process allocator. When the page transfer is complete, the trapped process is untrapped allowing it to continue.

The storage allocator also handles request tasks from processes. When a process does not want to be trapped by a 'not-in-core' trap, it handles a request task to the storage allocator to fetch a page from the backing store. When that is done, the storage allocator returns a response task back to the process. This procedure allows a higher throughput for the process.

A process may, also send request tasks to the storage allocator to open or close one of it's files. A file is a group of pages with consecutive virtual addresses, each page having the same status as a process. When a file is occupying part of the virtual address range of the process, it is said to be open. A process usually has three open files; the execute file, a read-only file and a read/write file. If the whole file does not need to be accessed or a file that belongs to a different process is required to be read, then the part-file mechanism may be used. The part-file request to the storage allocator always causes a copy of the information to be taken from or written to the requested file, that copy being transferred to or from an open read/write file of the requesting process. Information transferred can be any number of consecutive words up to a page, or a number of pages.

The combination of the 'Trap not-in-core' and the requests for pages and files gives the required flexibility and control in the management of the store while permitting the efficient use of both the main and backing store. The storage allocator brings into core the minimum number of pages to allow a process to run, and pages of low priority are only brought in when requested

4.3.2.4 Thrash and Crash Processes

These are a suite of processes collectively known as RASH primarily provided to assist in debugging and commissioning Mark II BL system. RASH provides the man/machine interface and process data (file data, task index tables, process descriptor) for a user to control and run test modules. The control facilities to the programmer include starting and stopping the test module and timing it. The test module used for performance or functional testing can be run by one RASH process only or by up to six processes running at the same time. Each of the RASH processes, known as a test process, is controlled individually using the commands of the man/machine language.

An additional facility available to the programmer is the ability to pass data items to the test module before it starts running. A programmer can also link a number of test modules in a chain to execute consecutively and store within RASH tasks to be handed to other processes periodically or one-shot only, for debugging purposes.

The test modules include general purpose test-modules which test various aspects of the system. These could be run at any time to exercise the system, and thereby provide a heavy workload for system testing.

Each test process has an execution counter which is incremented every time the process executes the test module and this facility is also used for timing test modules against periodic timing tasks.

4.3.2.5 Other Operating System Processes

Operating system processes described in the previous sections of this chapter are those most important and relevant to this simulation research project. Other operating system processes are, therefore, not described here. These include CANAL process and TYPER process which deal with input/output respectively. TRAP process which handles software traps, CAP (CPU Access Ports) process for faulty CPU diagnosis, background process and others.

CHAPTER 5

THE MARK II BL SYSTEM SIMULATOR

5.1 INTRODUCTION

As already mentioned in Chapter 2, the advantages of using SIMULA are related to the concepts provided by the language. At the heart of those concepts is the CLASS concept, a generalisation of the PROCEDURE concept in ALGOL 60. Also, the ability to prefix classes and blocks by other classes, so that they inherit their attributes and actions, allows the construction of hierarchical tree structures. This feature makes the language extendable and application oriented. In fact, that is how the list processing and simulation concepts are developed to convert the general purpose SIMULA language into a process-based simulation language.

SIMULA is a language for sequential processes which can only be executed one instruction at a time (BIRT 73). To enable the simulation analyst to model truly systems with several simultaneously operating processes, SIMULA has mechanisms which create the illusion of parallelisms, *viz* quasi-parallel programming. This is a common term for the mode of operations of co-routines. Only one routine is executed by the computer at any one time, and the sequential execution of a routine is only interrupted at a sequencing (or scheduling) statement. With respect to the sequential co-operating processes definition (DIJK, 68), a quasi-parallel system may informally be defined in either of the following ways: (PIEN 73).

> the sequence of operations between a sequencing statement and the next is a critical section, *i.e.* their executions are mutually exclusive.

- in a process an atomic (indivisible) operation may be defined at will, as a sequence of non-sequencing statements (preceded and terminated with a sequencing statement).

Thus a SIMULA process is executed in 'chunks', each chunk representing an activity initiated and terminated by events represented by the execution of sequencing statements. During an activity, a process is active and interacts with other system components (processes) resulting in a change of system state. Since simulated time is only changed using the sequencing statements, an activity does not consume any simulated time.

The system to be simulated (Mark IIBL) is a multi-processor multi-programmed computer system. In a multi-programming system, the processes are executed, one at a time, by one processor. The active phases of the processes and their scheduling sequence are dynamically defined externally to the processes (*i.e.* by an operating system). With reference to a system of sequential processes accessing common storage, the atomic operations in the multi-programming system are the indivisible (by hardware) store and fetch instructions. For a multi-processing system, the atomic operations are the same, but the active phases of the processes are of infinite length, since each process is executed by a separate processor. By contrast, in a quasi-parallel system, the atomic operations are defined at will within a process and coincide with the active phases. Thus quasi-parallelism is the most general of the three concepts, in the sense that the logical operations of the other two systems, or a mixture of them (such as Mark II BL), may be described by a quasi-parallel system. Hence, it is quite possible to construct an 'exact' model of Mark II BL system in SIMULA using its quasi-parallel programming approach.

5.2 DESIGN PHILOSOPHY OF THE SIMULATOR

The simulator follows the same modular approach of Mark II BL software and hardware. It is structured for ease of debugging and understanding by those using the package for software development. For the same reason, the simulator must model the system and its data structures in a one-to-one translation. This should result in a high correspondence between the model and the actual system.

The simulator ought to mimick in great detail the actual system, in order to carry out useful investigations and experiments into the system's resource management policies and structures.

On the other hand, the simulator must be general enough to be used in conjunction with different application software structures for different exchange types. These two conflicting requirements call for a careful choice regarding the level of detail of each component in the simulator. While infrequently used processes, procedures and data structures are discarded, others whose operations are important in reproducing the behavioural characteristics of the system are explicity modelled.

The simulator must retain the same interfaces between the system software modules or processes and must allow models of the application processes to use the same calls to the process allocator as they use in the actual system, for communication and task passing.

The transformation of a software process to its corresponding model must be made as simple and as easy as possible for the software design engineer. All that is required from him or her to do is to strip the different activities of a process of the actual codings representing those activities and replace the codings by the actual times consumed by those activities. The timing information of the different activities or missions within a process is readily available and is usually calculated from the number of instructions executed and the computer cycle time.

The simulator must allow the interrogation of the simulated CPU registers, *e.g.* the general registers G0 - G7, and allow the follow-up of individual messages and hence telephone calls.

It must permit the modelling of the system components at different levels of detail and the expansion of a process model from a macro to a micro level without affecting other parts of the simulator. The modular nature of the simulator must facilitate building up a library of software models, such that a software engineer can easily assemble the model structure of a particular application from that library.

The simulator must allow the software design engineer to input parameters, such as the simulated time and the number of CPUs in the configuration, on-line and obtain the run-result in a fairly short time.

The simulator must have a tracing facility that is triggered when required to give detailed system interactions and global table contents.

5.3. PROBLEMS IN SIMULATING A MULTI-PROCESSOR SYSTEM

A multi-processor system may be defined as a system possessing one or more sets of resources, with one or more identical members in each set, operating under central stored-program control and capable of processing one or more programs at any given point in time. Although much money was and is being spent in this sector, yet there is little basic understanding of the components interactions and the internal working of the system (HUTC 73).

Unlike job-shop simulations, simulations of multi-processor systems have to take account of some additional complications. Frequently, in a multi-processor, a program's routine requires resources of several different types simultaneously and, having once captured them, it must continually compete with all current routines to maintain their possession. In contrast, for a job-shop situation, once a job has acquired a resource (*e.g.* machine) it keeps the resource for the entire machining time without interruption. Whereas in job-shop simulations, the life cycle of a job is simply a linear list of a few entries in which it acquires resources and releases them, in a multi-processor system each incident of conflict between the active competing components must be modelled. Hence, it is necessary to describe each routine of each program to a degree of detail determined by the objectives of the simulation.

The resource management policies in a job-shop are easily simulated by routines for major functions such as assignment of jobs to machines. In the multi-processor system, the management functions are more complex and are performed by software, the operating system, which comprises programs that compete for and consume resources which might otherwise be assigned to workload programs. The time requirement of the operating system to carry its management functions can be highly significant in the multi-processor system.

Thus, due to the difficulties such as those mentioned above and the necessity of handling both minute details and higher-level activities, in addition to the more conventional requirements, some of the problems of systems simulation in general are emphasized for multi-processor systems. One of the problems is the degradation of the ability to differentiate between times of events occurring. This problem is particularly associated with discrete-event simulation languages that maintain a monotonically increasing master clock for the simulation *e.g.* SIMSCRIPT, SIMULA. Languages that use the concept of relative timing, *e.g.* CSL (BUXT 62) are free of this problem. In such languages, future events occurrence times are calculated relative to 'now' and stored in 'time cells' associated with the events. 'Now' means simulated time zero. Thus an event with the smallest time-cell value is the next event to occur.

On the other hand, event times for monotonically-increasing master-clock types of languages are calculated during the course of the simulation with added to the master clock time (t_m) to schedule the time of the occurrence of that event (t_e). The master-clock time value is stored in a finite number of bits, typically one computer word and floating point notation.. At any $t_m > t_e > 0$, where t_e is the time to elapse for the occurrence of the event e, there will be a t_e which is so small such that

$$t_m + t_e \simeq t_m$$

This value of t_e, designated by $\hat{t}_e$ could be defined by the following equation (HUTCH, 73)

$$\hat{t}_e = t_m/b^n$$

where

$\hat{t}_e$ is the largest t_e which can be added to t_m without increasing t_m.

t_m the master clock time

b the number base of the computer

n is the number of bits used for the mantissa of the floating point number.

In ICL SIMULA implementation on System 4 - 72, the current simulated time can be retrieved using a SIMULA SYSTEM - defined procedure TIME. This procedure is of the type LONG REAL,*i.e.*the timing information is stored in two computer words. A system 4 - 72 word is 24 bits long and a REAL number has 9 decimal digits. Therefore,

$$\hat{t}_e = t_m/_2 39 \approx 1.82 \times 10^{-13}\, t_m$$

If the time unit (or grain) is taken as 1 micro-second, then the maximum length of time to which the simulation could be run before degradation occurs is

$$t_m = \frac{10^{13}}{1.82} \times \frac{10^{-6}}{3600} = 1.525 \times 10^3 \text{ hrs.}$$

This t_m value is well above any anticipated period for which the simulator may be run. The danger of having a $t_e < \hat{t}_e$ is to force the simulator into an endless loop for a recurring event with the simulated time remaining unchanged as $t_e + t_m = t_m$. A second danger stems from the fact that,with the degradation of the ability to differentiate between events as t_m increases, the accuracy of the measurement system changes as the simulation progresses, contributing to the degradation of the simulation results. The penalty of using a double-precision word to store the timing information is a slight increase in execution time .

This time-grain problem normally occurs in hierarchally - structured models of different layers, with each layer being at a different level of detail (Figure (1.2)). With reference to Figure (1.2), the micro level, level 1, represents the model of the operating system, where the process allocator is described at a micro-level with a time grain of one microsecond. This is thought to be necessary in order to carry out simulation experiments with the scheduling algorithms, lockout occupancy and interrupt servicing policy. The other processes that communicate with the process allocator are either at the intermediate or macro level. The time grain at the intermediate level is typically a millisecond and at the macro level is a second.

An alternative solution to the differences in time grains of a multi-level hierarchally-structured model is to summarise the performance analysis results of level 1 in a form easily usable in the next layer above, *i.e.* level 2 (KOBA, 78). Usually a scaling factor or a random variable with some distribution is used as summarised statistics for an interface of two layers of different levels of details.

Another problem in simulating multi-processor systems is that of the execution of simultaneous events in a sequential machine. Again, this problem is not unique to multi-processor systems, but the fact that the simultaneous-events density is higher in the case of multi-processors, especially when investigating resource allocation algorithms, makes them a candidate for the category of 'worst case'. The problem with the execution of simultaneous events sequentially is that the order in which events are executed depends on the simulation system itself and not the system under study, *e.g.* the simulation system (language) may adopt the algorithm that the first event placed in the event list at this time shall be executed first. Under these conditions, the outcome of running the same model with identical data on different computers could easily be very different (HUTC 68). This problem could escalate when some simultaneous events schedule further simultaneous events and so on. A number of techniques are suggested to deal with such situations, e.g. the use of a suspense set for simultaneous events. This problem presented a major challenge in the simulation of the interrupt-handling mechanism of the Mark II BL system, as will be shown in Chapter 6. The simulation of a multi-processor system must not only reflect the logic of the system algorithms, but must also retain the sequence of occurrence of simultaneous events. This is achieved in this simulation by paying particular attention to the type and location of the language (SIMULA) scheduling statements.

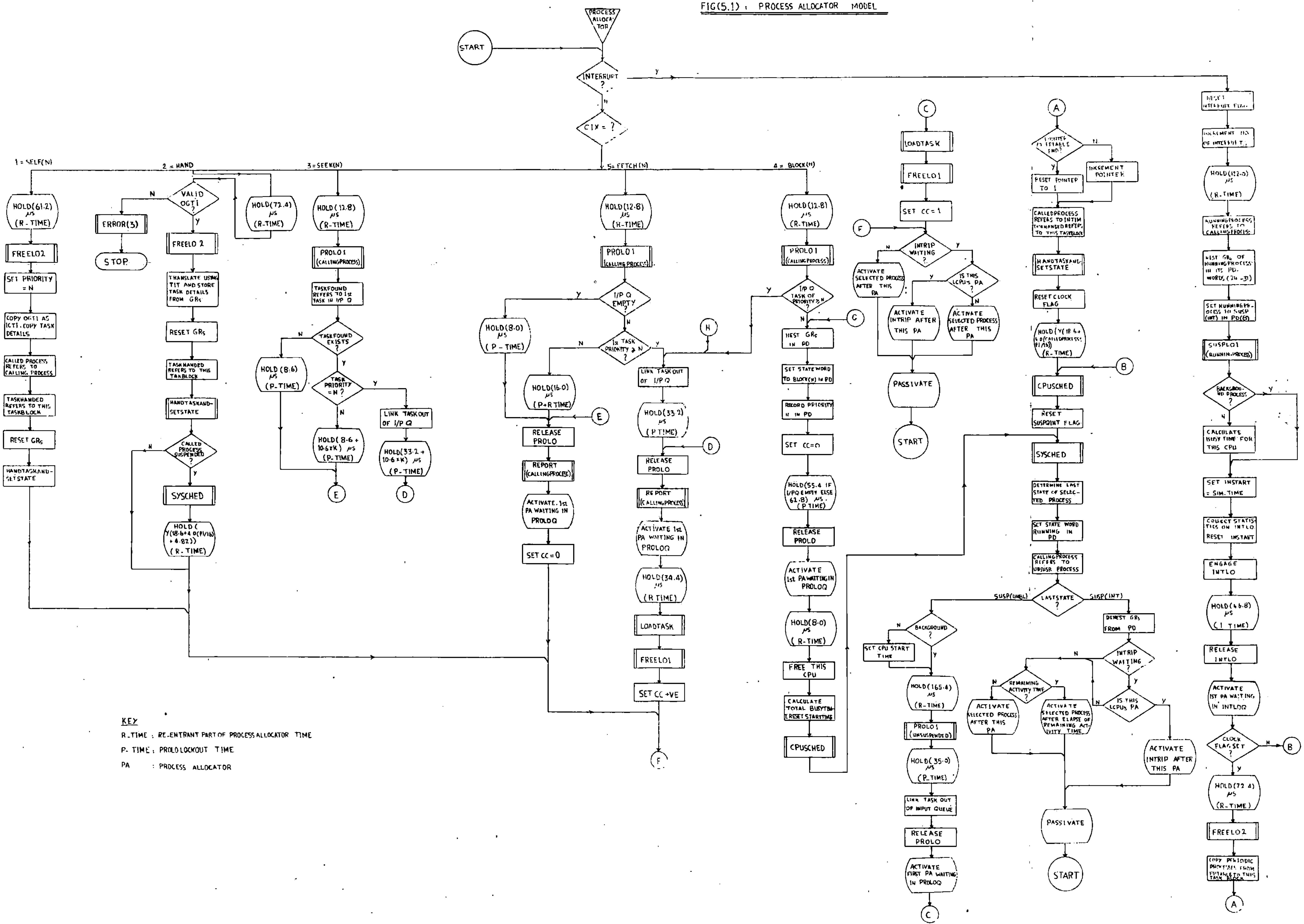
5.4. THE SIMULATOR PROGRAM OF MARK II BL

By far the biggest process in the simulator program is that of the process allocator (Figure (5.1), 719-1713). As mentioned before, this is because the process allocator is central to the real-time operating system and a detailed study of the system's resource management policies requires a detailed simulation of the process allocator. The other simulation modules (processes) representing the other system hardware components and operating system processes are:

- CPU
- TASKBLOCK
- AP
- INTIM
- INTRIPLICATES
- CLOCKINTERRUPT
- SA (Storage Allocator)
- RASH
- BACKGROUND

The numbers in brackets refer to the listing of the System X simulator package in Appendix B.

FIG(5.1) : PROCESS ALLOCATOR MODEL



PROCESS ALLOCATOR
START
INTERRUPT ?
Y
N
CIX = ?
1 = SELF(N)
2 = HAND
3 = SEEK(N)
5 = FETCH(N)
4 = BLOCK(N)
HOLD(61.2) μs (R-TIME)
FREELO2
SET PRIORITY = N
COPY OGT1 AS ICT1. COPY TASK DETAILS
CALLED PROCESS REFERS TO CALLING PROCESS
TASKHANDED REFERS TO THIS TASKBLOCK
RESET GRs
HANDTASKANDSETSTATE
VALID OGT1 ?
ERROR(3)
STOP
HOLD(72.4) μs (R-TIME)
FREELO 2
TRANSLATE USING TIT AND STORE TASK DETAILS FROM GRs
RESET GRs
TASKHANDED REFERS TO THIS TASKBLOCK
HANDTASKANDSETSTATE
CALLED PROCESS SUSPENDED ?
SYSCHED
HOLD(Y(18.6+4.0(PI/16)+4.82)) (R-TIME)
HOLD(12.8) μs (R-TIME)
PROLO1 (CALLINGPROCESS)
TASKFOUND REFERS TO 1st TASK IN I/P Q
TASKFOUND EXISTS ?
HOLD (8.6) μs (P-TIME)
TASK PRIORITY = N ?
HOLD(8.6 + 10.6×K) μs (P-TIME)
LINK TASKOUT OF I/P Q
HOLD(33.2 + 10.6×K) μs (P-TIME)
E
D
HOLD(12.8) μs (R-TIME)
PROLO1 (CALLINGPROCESS)
I/P Q EMPTY ?
HOLD(8.0) μs (P-TIME)
1st TASK PRIORITY ≥ N ?
HOLD(16.0) μs (P+R TIME)
RELEASE PROLO
REPORT (CALLINGPROCESS)
ACTIVATE 1st PA WAITING IN PROLOQ
SET CC = 0
H
LINK TASK OUT OF I/P Q
HOLD(33.2) μs (P TIME)
RELEASE PROLO
REPORT (CALLINGPROCESS)
ACTIVATE 1st PA WAITING IN PROLOQ
HOLD(34.4) μs (R TIME)
LOADTASK
FREELO1
SET CC +VE
F
HOLD(12.8) μs (R-TIME)
PROLO1 (CALLINGPROCESS)
I/P Q TASK OF PRIORITY ≥ N ?
G
NEST GRs IN PD
SET STATE WORD TO BLOCK(N) IN PD
RECORD PRIORITY N IN PD
SET CC=0
HOLD(55.4 IF I/PQ EMPTY ELSE 61.8) μs. (P TIME)
RELEASE PROLO
ACTIVATE 1st PA WAITING IN PROLOQ
HOLD(8.0) μs (R-TIME)
FREE THIS CPU
CALCULATE TOTAL BUSYTIME RESET STARTTIME
CPUSCHED
C
LOADTASK
FREELO1
SET CC = 1
INTRIP WAITING ?
ACTIVATE SELECTED PROCESS AFTER THIS PA
IS THIS LCPUs PA ?
ACTIVATE INTRIP AFTER THIS PA
ACTIVATE SELECTED PROCESS AFTER THIS PA
PASSIVATE
START
A
POINTER AT PTTABLE END?
RESET POINTER TO 1
INCREMENT POINTER
CALLEDPROCESS REFERS TO INTIM TASKHANDED REFERS TO THIS TASKBLOCK
HANDTASKANDSETSTATE
RESET CLOCK FLAG
HOLD(Y(18.6+4.0(CALLEDPROCESS PI/16)) (R-TIME)
B
CPUSCHED
RESET SUSPOINT FLAG
SYSCHED
DETERMINE LAST STATE OF SELECTED PROCESS
SET STATE WORD RUNNING IN PD
CALLINGPROCESS REFERS TO UNSUSP PROCESS
LASTSTATE ?
SUSP(UNBL)
SUSP(INT)
BACKGROUND ?
SET CPU START TIME
HOLD(165.4) μs (R-TIME)
PROLO1 (UNSUSPENDED)
HOLD(35.0) μs (P_TIME)
LINK TASK OUT OF INPUT QUEUE
RELEASE PROLO
ACTIVATE FIRST PA WAITING IN PROLOQ
DENEST GRs FROM PD
INTRIP WAITING ?
REMAINING ACTIVITY TIME ?
ACTIVATE SELECTED PROCESS AFTER THIS PA
ACTIVATE SELECTED PROCESS AFTER ELAPSE OF REMAINING ACTIVITY TIME
IS THIS LCPUs PA
ACTIVATE INTRIP AFTER THIS PA
PASSIVATE
START
RESET INTERRUPT FLAG
INCREMENT NO OF INTERRUPTS
HOLD(122.0) μs (R-TIME)
RUNNINGPROCESS REFERS TO CALLINGPROCESS
NEST GRs OF RUNNINGPROCESS IN ITS PD. WORDS (24-31)
SET RUNNINGPROCESS TO SUSP (INT) IN PD(8)
SUSPLO1 (RUNNINGPROCESS)
BACKGROUND PROCESS ?
CALCULATE BUSY TIME FOR THIS CPU
SET INSTART = SIM. TIME
COLLECT STATISTICS ON INTLO RESET INSTART
ENGAGE INTLO
HOLD(46.8) μs (I TIME)
RELEASE INTLO
ACTIVATE 1ST PA WAITING IN INTLOQ
CLOCK FLAG SET ?
HOLD(72.4) μs (R-TIME)
FREELO2
COPY PERIODIC PROCESSES FROM PTTABLE TO THIS TASK BLOCK

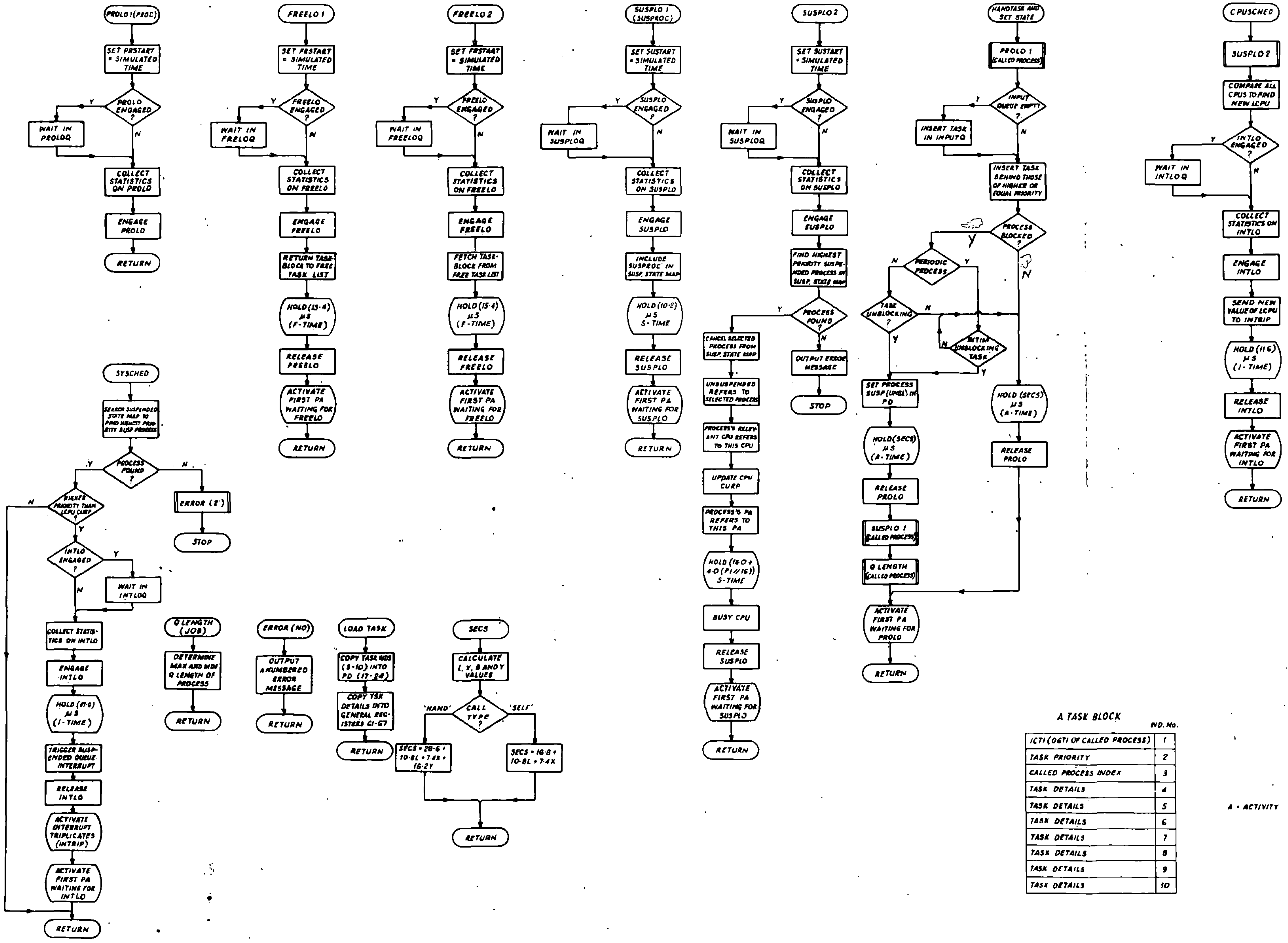
KEY

R-TIME : RE-ENTRANT PART OF PROCESS ALLOCATOR TIME

P-TIME : PROLOLOCKOUT TIME

PA : PROCESS ALLOCATOR

FIG. (5.2) PROCEDURES FOR THE PROCESS ALLOCATOR MODEL


PROLO 1(PROC)
SET PRSTART = SIMULATED TIME
PROLO ENGAGED ?
Y
WAIT IN PROLOQ
N
COLLECT STATISTICS ON PROLO
ENGAGE PROLO
RETURN
FREELO 1
SET FRSTART = SIMULATED TIME
FREELO ENGAGED ?
Y
WAIT IN FRELOQ
N
COLLECT STATISTICS ON FREELO
ENGAGE FREELO
RETURN TASK-BLOCK TO FREE TASK LIST
HOLD (15·4) μS (F-TIME)
RELEASE FREELO
ACTIVATE FIRST PA WAITING FOR FREELO
RETURN
FREELO 2
SET FRSTART = SIMULATED TIME
FREELO ENGAGED ?
Y
WAIT IN FREELOQ
N
COLLECT STATISTICS ON FREELO
ENGAGE FREELO
FETCH TASK-BLOCK FROM FREE TASK LIST
HOLD (15·4) μS (F-TIME)
RELEASE FREELO
ACTIVATE FIRST PA WAITING FOR FREELO
RETURN
SUSPLO 1 (SUSPROC)
SET SUSTART = SIMULATED TIME
SUSPLO ENGAGED ?
Y
WAIT IN SUSPLOQ
N
COLLECT STATISTICS ON SUSPLO
ENGAGE SUSPLO
INCLUDE SUSPROC IN SUSP. STATE MAP
HOLD (10·2) μS S-TIME
RELEASE SUSPLO
ACTIVATE FIRST PA WAITING FOR SUSPLO
RETURN
SUSPLO 2
SET SUSTART = SIMULATED TIME
SUSPLO ENGAGED ?
Y
WAIT IN SUSPLOQ
N
COLLECT STATISTICS ON SUSPLO
ENGAGE SUSPLO
FIND HIGHEST PRIORITY SUSPENDED PROCESS IN SUSP. STATE MAP
PROCESS FOUND ?
Y
N
CANCEL SELECTED PROCESS FROM SUSP. STATE MAP
UNSUSPENDED REFERS TO SELECTED PROCESS
PROCESS'S RELEVANT CPU REFERS TO THIS CPU
UPDATE CPU CURP
PROCESS'S PA REFERS TO THIS PA
HOLD (16·0 + 4·0 (PI // 16)) S-TIME
BUSY CPU
RELEASE SUSPLO
ACTIVATE FIRST PA WAITING FOR SUSPLO
RETURN
OUTPUT ERROR MESSAGE
STOP
HANDTASK AND SET STATE
PROLO 1 (CALLED PROCESS)
INPUT QUEUE EMPTY ?
Y
INSERT TASK IN INPUTQ
N
INSERT TASK BEHIND THOSE OF HIGHER OR EQUAL PRIORITY
PROCESS BLOCKED ?
Y
N
PERIODIC PROCESS
N
Y
TASK UNBLOCKING ?
N
Y
INTIM UNBLOCKING TASK
N
Y
SET PROCESS SUSP (UNBL) IN PD
HOLD (SECS) μS (A-TIME)
RELEASE PROLO
SUSPLO 1 (CALLED PROCESS)
Q LENGTH (CALLED PROCESS)
HOLD (SECS) μS (A-TIME)
RELEASE PROLO
ACTIVATE FIRST PA WAITING FOR PROLO
RETURN
CPUSCHED
SUSPLO 2
COMPARE ALL CPUS TO FIND NEW LCPU
INTLO ENGAGED ?
Y
WAIT IN INTLOQ
N
COLLECT STATISTICS ON INTLO
ENGAGE INTLO
SEND NEW VALUE OF LCPU TO INTRIP
HOLD (11·6) μS (I-TIME)
RELEASE INTLO
ACTIVATE FIRST PA WAITING FOR INTLO
RETURN
SYSCHED
SEARCH SUSPENDED STATE MAP TO FIND HIGHEST PRIORITY SUSP PROCESS
PROCESS FOUND ?
Y
N
ERROR (2')
STOP
HIGHER PRIORITY THAN LCPU CURP ?
N
Y
INTLO ENGAGED ?
Y
WAIT IN INTLOQ
N
COLLECT STATISTICS ON INTLO
ENGAGE INTLO
HOLD (11·6) μS (I-TIME)
TRIGGER SUSPENDED QUEUE INTERRUPT
RELEASE INTLO
ACTIVATE INTERRUPT TRIPLICATES (INTRIP)
ACTIVATE FIRST PA WAITING FOR INTLO
RETURN
Q LENGTH (JOB)
DETERMINE MAX AND MIN Q LENGTH OF PROCESS
RETURN
ERROR (NO)
OUTPUT A NUMBERED ERROR MESSAGE
RETURN
LOAD TASK
COPY TASK WDS (3-10) INTO PD (17-24)
COPY TSK DETAILS INTO GENERAL REGISTERS G1-G7
RETURN
SECS
CALCULATE L, X, B AND Y VALUES
CALL TYPE ?
'HAND'
'SELF'
SECS = 28·6 + 10·8L + 7·4X + 16·2Y
SECS = 16·8 + 10·8L + 7·4X
RETURN
A = ACTIVITY

A TASK BLOCK

	WD. No.
ICTI (OGTI OF CALLED PROCESS)	1
TASK PRIORITY	2
CALLED PROCESS INDEX	3
TASK DETAILS	4
TASK DETAILS	5
TASK DETAILS	6
TASK DETAILS	7
TASK DETAILS	8
TASK DETAILS	9
TASK DETAILS	10

5.4.1 The main program

The main program contains definitions of the global parameters used by the process allocators on different CPUs and other processes. The suspended-state map (where the identities of suspended processes are stored) is represented by a BOOLEAN ARRAY SUSPMAP (0:127) (155) and is an example of global tables. The maximum number of software processes (both operating system and application processes) is not expected to exceed 128, hence the size of the array. To reference processes regardless of whether they are operating-system or application processes, a reference array is defined, REF(AP)ARRAY P(0:127) (160). AP is an entity which prefixes all software entities in the simulator.

Global parameters include the lockouts FREELO and SUSPLO, defined as resources for which the process allocators compete. FREETASKLIST is the queue for free task blocks. LPAQ is a queue where the interrupt triplicates model or entity waits for the process allocator on LCPU to finish servicing its process allocator call or interrupt, before sending its new interrupt to that process allocator.

The simulated time for a run (SIMPERIOD) and the number of CPUs in a configuration (NUM) are global parameters that are set by the user before starting the simulation (144, 146). Pointer (151) is used to point to successive rows of the periodic processes table. The pointer is initialised to row No. 1 (4080) and is incremented or reset to 1 by the process allocator servicing the clock interrupt (1197, 1202). Some global procedures to assist in the outputting of reports and error messages are also defined (171 - 213). The main program creates instances of the entities in the simulator and refer to them by certain defined reference names e.g. (3929) or by the REF ARRAY P, (3931).

The task index tables of the entities are then initialised. The entities themselves are initialised as being in the state BLOCK(15) *i.e.* waiting for a task of priority equal to 15 or higher. For every process model or entity an input queue (INPUTQ) is defined to hold incoming task messages, and a lockout, PROLO (defined as a resource), to control the access of the process allocator to the input queue.

The main program then creates 100 task block entities and links them to FREETASKLIST. It then holds or waits for the simulated time to elapse (4085) before outputting a summary report for the run. When the main program suspends itself, it arranges for program control to pass to the clock interrupt entity (4084).

5.4.2 The Clock Interrupt (1717 - 1761) (Figure (5.3))

This entity generates instances of the CPU entity, the number of instances corresponding to the number of CPUs in the configuration and determined by the global parameter NUM. The value of NUM is either read in from the input data file or typed in by the user at the start of the simulation run. For each CPU, a background process model (*i.e.* AP CLASS BACKGROUND) is created with its state set to running (1742) in its process descriptor. The instances generated are then activated (1750 - 1751).

The clock interrupt entity then selects the CPU running the lowest priority background process as the lowest priority CPU, LCPU, and transmits its identity to the interrupts triplicates (1753). It sets the clock interrupt flag of the interrupt triplicates to TRUE signalling the arrival of a clock interrupt, and then activates the interrupt triplicates unit. This action is repeated every 10 msec to the end of the simulation run (1754 - 1760).

5.4.3 The CPU Process (379 - 411) (Figure (5.4))

Most of the hardware is represented by classes containing only data or very little actions. Data pertaining to the hardware component, CPU, is a set of 16 registers, a CPU number tag, condition codes, a reference to the current running entity and its associated process allocator and some parameters to calculate the occupancy of the background entities running on this CPU. The consumption of CPU power by the process allocator entity or any other process entity is represented by the 'HOLD' statement inside the bodies of those entities. The 'HOLD' statement represents the elapse of simulated time while the respective entity is 'holding' that system resource *i.e.* running on the CPU. The CPU occupancy by different processes entities is calculated by the process allocator when scheduling that CPU.

The only action of a CPU instance is to generate its own process allocator. It references the process running by CURP (Current Running Process). This will be a background process at the start of a run. It also makes its process allocator reference, CALLINGPROCESS, to point to CURP. When the CPU instance exhausts its actions it is terminated *i.e.* will never be active again. It remains existing as a data structure (with no actions) and is referenced by REF ARRAY C(I), (158).

5.4.4 Background Process (1796 - 1812) (Figure (5.5))

This is the simplest of all the entities. Its sole function is to occupy a CPU when there is no other useful work to be done. There is a background entity for each of the CPUs. The simulator is initialised with all the CPUs running background entities.

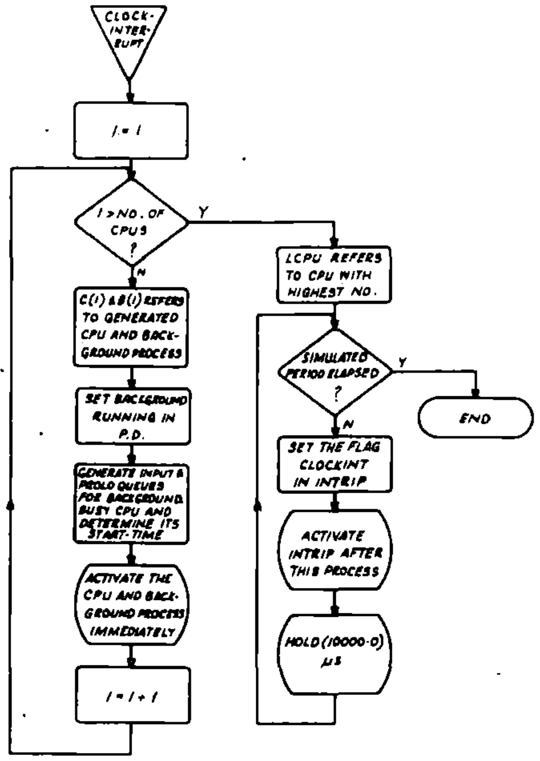

CLOCK-INTER-RUPT
I = 1
I > NO. OF CPUS ?
Y
N
C(I) & B(I) REFERS TO GENERATED CPU AND BACK-GROUND PROCESS
SET BACKGROUND RUNNING IN P.D.
GENERATE INPUT & PEOLO QUEUES FOR BACKGROUND BUSY CPU AND DETERMINE ITS START-TIME
ACTIVATE THE CPU AND BACK-GROUND PROCESS IMMEDIATELY
I = I + I
LCPU REFERS TO CPU WITH HIGHEST NO.
SIMULATED PERIOD ELAPSED ?
Y
END
N
SET THE FLAG CLOCKINT IN INTRIP
ACTIVATE INTRIP AFTER THIS PROCESS
HOLD (10000·0) µs

FIG (5.3) CLOCKINTERRUPT MODEL FLOW CHART

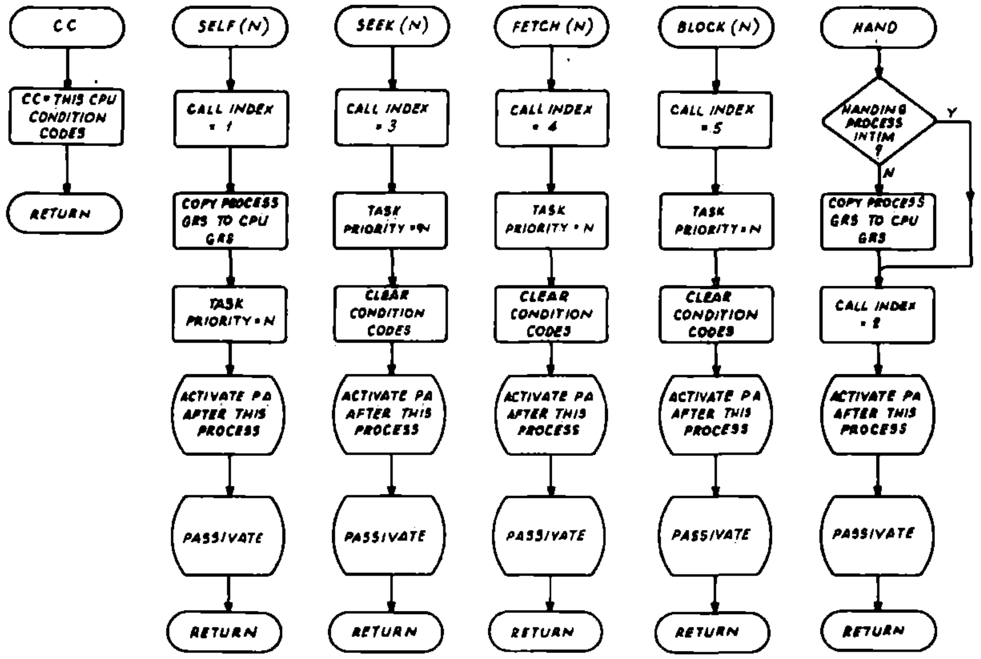


FIG. (5.7) AP MODEL FLOW CHART

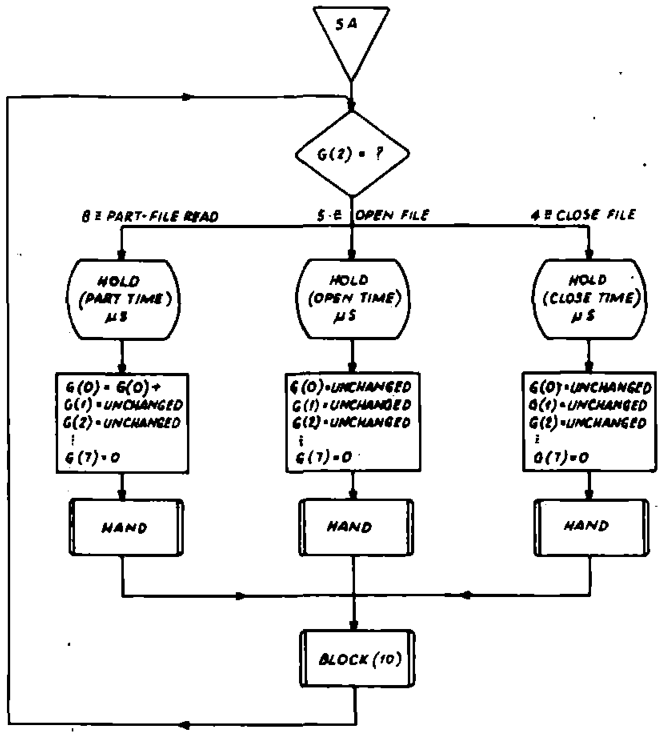


FIG. (5.8) THE STORAGE ALLOCATOR MODEL FLOW CHART

FIG. (5.4) CPU MODEL FLOW CHART

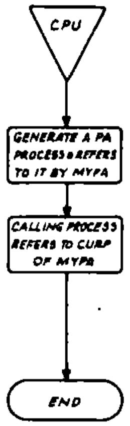

CPU
GENERATE A PA PROCESS & REFERS TO IT BY MYPA
CALLING PROCESS REFERS TO CURP OF MYPA
END

FIG (5.6) TASKBLOCK MODEL FLOW CHART

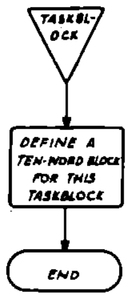
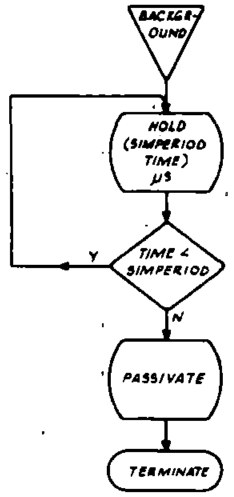


FIG (5.5) BACKGROUND MODEL FLOW CHART



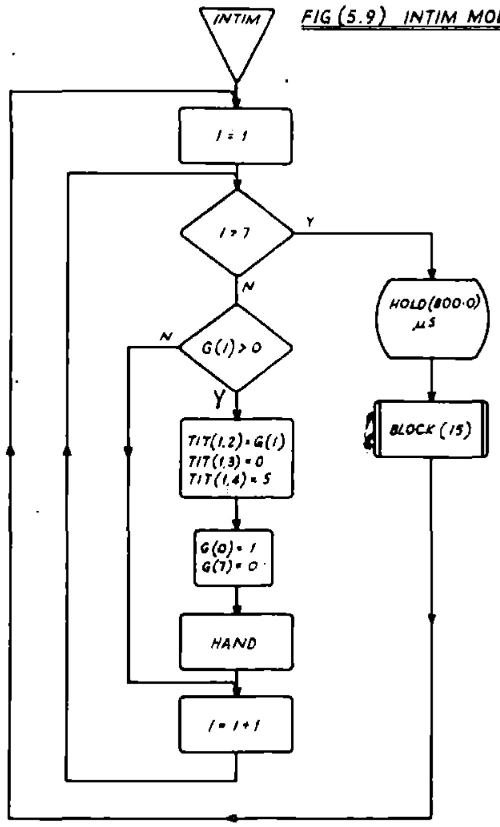


FIG (5.9) INTIM MODEL FLOW CHART

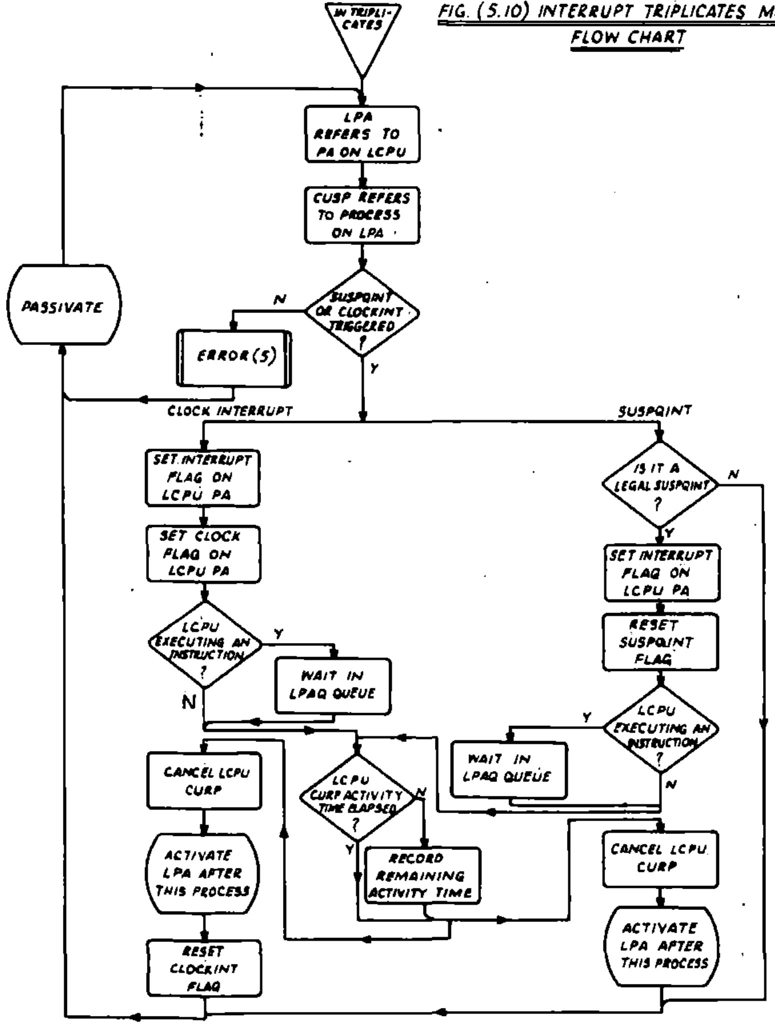

IN TRIPLI-CATES
LPA REFERS TO PA ON LCPU
CUSP REFERS TO PROCESS ON LPA
PASSIVATE
N
SUSPQINT OR CLOCKINT TRIGGERED ?
ERROR (5)
Y
CLOCK INTERRUPT
SUSPQINT
SET INTERRUPT FLAG ON LCPU PA
SET CLOCK FLAG ON LCPU PA
LCPU EXECUTING AN INSTRUCTION ?
Y
WAIT IN LPAQ QUEUE
N
CANCEL LCPU CURP
ACTIVATE LPA AFTER THIS PROCESS
RESET CLOCKINT FLAG
LCPU CURP ACTIVITY TIME ELAPSED ?
N
Y
RECORD REMAINING ACTIVITY TIME
IS IT A LEGAL SUSPQINT ?
N
Y
SET INTERRUPT FLAG ON LCPU PA
RESET SUSPQINT FLAG
LCPU EXECUTING AN INSTRUCTION ?
Y
WAIT IN LPAQ QUEUE
N
CANCEL LCPU CURP
ACTIVATE LPA AFTER THIS PROCESS

FIG. (5.10) INTERRUPT TRIPLICATES MODEL FLOW CHART

5.4.5 Taskblock (415 - 425) (Figure (5.6))

This is an entity with no actions. It exists as a data structure with a one-dimensional INTEGER ARRAY of size 10 corresponding to the size of the task block in the actual system..

5.4.6 Process AP (247 - 347) (Figure (5.7))

This entity embodies the data structure common to the processes of Mark II BL and procedures which allow a process entity to communicate with other processes entities through the process allocator. Every operating system or application process model is prefixed by AP and hence inherits its attributes (by the prefixing rules of SIMULA). The data structure in AP contains a 32-word process descriptor (defined as an array) in which information relating to the entity and its environment is stored. This information is used by the process allocator in the management of the system. The task index table (TIT) is also defined on a per-entity basis. A process allocator uses an outgoing task index to index down this table to obtain such information as the identity of the called process entity, the task priority and an incoming task index which determines the response of the entity receiving the task. Also defined is an array of size 8 to simulate the eight general registers GO-G7. This is defined here to allow the simulator user to assign the details of a task to the general registers within the application process model or entity, in much the say way as he does in the coding of the real application process.

A BOOLEAN variable PERIODIC is true for periodic processes. It is used by procedure HANDTASK AND SETSTATE in conjunction with periodic processes when changing their state from blocked to suspended.

The resource entity, PROLO, is defined to control access to the input queue by the process allocators. Each entity references the CPU in which it is running and the process allocator on that CPU by RELEVANTCPU and PA respectively. The real variable, REMAININGACTIME, is used to store the remaining time of a 'HOLD' sequencing statement when the process is interrupted in the middle of the 'HOLD', so that when selected to run again, this remaining activity time is executed first.

Calls to the process allocator available to a process are represented by procedures local to class AP. These calls are 'HAND', 'FETCH(N)','SEEK(N)', 'BLOCK(N)','SELF(N)' and a recent addition 'FBLOCK(P)'. The calls 'TRAP' and 'UNTRAP' are not relevant to the simulation objectives and hence not simulated. The body of a call procedure determines a call index - an integer value used later by the serving process allocator to branch to the appropriate servicing routine, via a SWITCH statement (1144). In case of a FETCH, BLOCK or SEEK call, the CPU conditions codes are reset to zero in the body of the procedure. The last action of a call procedure is to passivate the calling process and activate the serving process allocator. PROCEDURE CC is provided so that a process can check the condition codes of the CPU on which it is running.

5.4.7 <u>The Storage Allocator Process</u> (429 - 498) (Figure (5.8))

This is a macro-level model of the storage allocator which models the servicing of close file, open file and part file requests from processes. The storage allocator determines the nature of the incoming request from the value of G2. A value of 4 is an open file request, and a value of 5 is a close file request. The response action of the storage allocator to such a request is to 'HOLD' for the corresponding activity time and then send back a response task to the requesting process. The storage allocator picks up any more pending tasks by a BLOCK(10) instruction. Therefore, when all tasks in the input queue have been processed, the storage allocator is blocked awaiting the arrival of a fresh task. It is prefixed by AP and hence inherits all its attributes.

5.4.8 INTIM Process (562 - 609) (Figure (5.9))

The functions of the INTIM (Interrupt and Timing) process relevant to the simulator are to unblock periodic processes at periodical intervals and to send timing tasks to processes which require them. With the arrival of each clock interrupt, INTIM is handed a task containing the identities of the periodic processes to be activated (their process indices numbers). INTIM will most probably be selected to run because of its high priority. It starts by checking the general registers G1 - G7 for the identities of the periodic processes. The unblocking periodic tasks handed by INTIM to the blocked periodic processes are of a high periority (5) to ensure that the tasks will be linked to the front of the respective input queues. This priority number (5) is also used by the process allocator to identify such an unblocking task and subsequently change the state of the called periodic process from blocked to suspended (unblocked). After handling all unblocking tasks, INTIM calls to block awaiting the next clock interrupt.

5.4.9 The Interrupt Triplicates Process (613 - 714) (Figure (5.10))

This entity simulates the functions of the triplicated hardware interrupt unit. It is activated as soon as an immediate interrupt is triggered (1758,-1014). The immediate interrupts relevant to the simulator are the clock interrupt (every 10 msec) and the suspended queue interrupt which is triggered whenever a system scheduling is to take place.

This entity has two BOOLEAN attributes; SUSPQINT and CLOCKINT to indicate the arrival of a suspended queue interrupt and the clock interrupt respectively. PPTABLE, is initialised in the body of the main program to contain the periodic processes indentities to be activated at subsequent clock interrupts. The process allocators access this table to copy the identities of those processes in a task to be handed to INTIM. LCPU is a reference variable pointing to the CPU entity running the lowest priority process. This reference is updated by the process allocator whenever a CPU schedule is carried out. LPA is a reference to the process allocator on LCPU. CUSP and CUCP references the current suspended entity and the current chosen entity of LCPU.

The action part of this entity starts by updating the reference to the process allocator on LCPU, as, most probably, it has changed since this interrupt triplicates process served the last immediate interrupt. If the interrupt triplicate process is activated without an immediate interrupt occurring, an error message is outputted and the entity passivates without taking any further action.

However, if the suspended queue interrupt is triggered, the interrupt triplicates process will reset the SUSPQINT flag and checks whether the process allocator on LCPU is active or not. If it is active, then it is either servicing a process allocator call or an interrupt. In such a case, the interrupt triplicates will wait in a special queue (LPAQ) for the process allocator to finish. When it is active again, it resumes its actions by setting the interrupt flag in the process allocator (LPA). It checks whether the process on LCPU is running.

This is, of course, represented by the fact that the process is executing a 'HOLD' statement to represent in SIMULA the passage of simulated time corresponding to the execution of a particular activity. If that is the case, the interrupt triplicates will calculate the remaining activity time for that process and store it in REMAININGACTIME. Later on, when this process is selected to run again, that remaining activity time is first executed. The process allocator on LCPU, *i.e.*LPA, is then activated to serve the interrupt and the triplicates unit passivates awaiting the arrival of the next immediate interrupt.

On the other hand, if the immediate interrupt is that of the hardware clock, the action of the triplicates unit is the same as for the suspended queue interrupt, except that the CLOCK flag is set on LPA instead of SUSPQINT.

In the interrupt triplicates unit of the real system (MK II BL) no consideration is given to detect a particular sequence of events which might lead to an unnecessary servicing of an interrupt (and hence reduce the system throughput). Such a sequence of events was noticed from close examination of a detailed output trace during the verification and validation stages of the simulator. An example of such a sequence of events is the case where the interrupt triplicates, being activated because of a suspended queue interrupt, is waiting for LPA (the process allocator on the CPU running the lowest priority process) to finish servicing its current call. That call could as well be a call to block 'BLOCK(N)', and the process allocator on not finding the requested task of priority N blocks the running process and selects another one. It is then quite probable that the selected process is of a higher priority than the process intended to be selected for LCPU when servicing the suspended queue interrupt.

In such circumstances, after servicing the block call, the process allocator enters the interrupt servicing routine, de-nests the previously selected process, but selects it again on priority basis, nests its registers and sets it running. This unnecessary delay could have been avoided if the interrupt triplicates process checks the priority of the process on LCPU against the highest priority suspended process before initiating the process allocator on LCPU to service the suspended queue interrupt. This additional feature to the actual system is included in the simulator and is found to result in an increased throughput and reduced process allocator overhead. More about that in Chapter 6.

5.4.10 The Process Allocator Process (719 - 1713) (Figures (5.1) (5.2))

This is the biggest and most detailed entity. It models the following functions of the process allocator at a micro level of detail:

(a) Handling of communication between different processes.
(b) Interrupts Servicing.
(c) Individual CPU scheduling as well as overall system scheduling.

However, this entity does not model DMA interrupts servicing nor the calls to trap and untrap a process. These are thought to be irrelevant to the fulfillment of the objectives of the simulator outlined in 5.2.

A number of procedures are defined as attributes to this entity to model specific functions. These procedures are described below.

The process allocator model is activated either by a process allocator call from the process model running on its CPU or by the occurrance of an immediate interrupt where the interrupt is steered by the interrupt triplicates unit to this CPU. In the former case, the activation is done within the procedures modelling the process allocator calls (see 5.4.6), whereas in the latter, the activation is done by the interrupt triplicates entity (5.4.9).

When activated, control either passes to the interrupt servicing branch of the process allocator (1141) or to the appropriate call servicing routine using a SWITCH statement in conjunction with a particular call index indicating the call type (1144).

The following procedures are defined as attributes to the process allocator (797 - 1134):

<u>PROCEDURE PROLO 1 (PROC)</u>: (797 - 814)

Checks the mutual exclusion primitive that guards the input queue of the process referenced by PROC. This primitive is referred to by the name PROLO Lockout. As mentioned before, each process has such a lockout to regulate the access of the different process allocators in a particular multiprocessor configuration to a process input queue. If PROLO is engaged by another process allocator, then this process allocator will wait until it it released in a special queue. When released, the first process allocator waiting will engage PROLO.

<u>PROCEDURE FREELO 1</u>: (828 - 845)

This procedure will check whether the lockout FREELO which guards the pool of free task blocks, is free and if not the process allocator joins a special queue and passivates until activated by the process allocator releasing the lockout. The process allocator then holds for 15.4 micro-seconds which is the time to return a free task block to the free task list. When the free task block is returned, FREELO is released and first process allocator waiting, if any, is activated.

PROCEDURE FREELO 2: (848 - 873)

This is similar to FREELO 1 procedure except that it fetches a free task block instead of inserting one. For the sake of storage efficiency, only 100 task blocks are generated - in the main program initialisation section - and inserted in the free task list (4082 - 4083). If then the freetask blocks pool is exhausted a new task is generated and inserted in the list. This is a diversion from the actual system where the number of free task blocks is fixed and determined at the initialisation stage of the system. If the free task block is exhausted an overload control process will be invoked which restores the level of the free task list to an acceptable one. Since no overload control strategy is simulated and the number of task blocks that could satisfy different applications and configurations is variable, this method of catering for the provision of free task blocks is convenient and more efficient.

PROCEDURE SUSPLO 1 (SUSPROC): (876 - 906)

SUSPROC references the process to be included in the suspended state map, a BOOLEAN ARRAY here. The process allocator checks SUSPLO Lockout and if engaged joins a special queue and passivates. The process allocator engaging SUSPLO, having finished accessing the suspended state map, will release SUSPLO and activate the first waiting process allocator, if any. When this process allocator is re-activated, it engages SUSPLO for 10.2 micro-seconds, the time required to insert a process in the suspended state map. The process is inserted in the map by setting the array element corresponding to the process index (PI) to true *i.e.*

SUSPMAP(SUSPROC.PI): = TRUE;

SUSPLO is then released, and the first process allocator waiting is activated, if any.

<u>PROCEDURE SUSPLO 2:</u> (909 - 960)

The actions of this procedure follows closely those of SUSPLO1(SUSPROC), however, the intention here is to locate the highest priority suspended process in, and remove it from the suspended state map. As processes' priorities are inversely proportional to their process indices, *i.e* the lower the process index, the higher the priority, the search for the highest priority process consists of scanning the suspended state map (the BOOLEAN ARRAY SUSPMAP (0:127)) from the beginning until a true-value element is found. The selected process is referenced and removed from the map by re-setting the subscript value to false. CURP references this process model or entity.

Since when initialising the simulator, a number of background processes equal to the number of CPUs are created and set running on those CPUs, there will always be at least one suspended process in the suspended state map. If no such process is found, that will be an indication of malfunction and an error message will be outputted.

In the Mark IIBL System, the time for which SUSPLO is engaged is partly dependent on the location of the selected process in the suspended state map and hence on the value of its process index. Thus the time for which SUSPLO is engaged is calculated on the basis of the selected process index, using the empirical formula

$$T = 16.6 + 4(PI//16) \text{ micro-seconds}$$

where

// denotes integer division
PI the process index value.

PROCEDURE LOAD TASK: (963 - 973)

It loads the selected task block details into the CPU general registers Go - G7. This procedure is called when a suspended (unblocked) process is selected to run and when a FETCH, SEEK or BLOCK call is executed and the requested task is found in the process's input queue. It also copies the task details in PD (17 - 24).

PROCEDURE SYSCHED: (976 - 1019)

It sets the scene for a system re-schedule when a higher priority process model (or entity) is suspended while a lower priority one is running. The suspended state map is scanned to identify the higher priority suspended process. If the suspended state map is found to be empty, an error message will be generated as that constitutes an illegal system state. This is because at least one, or more, background processes are expected to be suspended.

The established process index of the suspended process, if found, is then compared with the lowest priority process running *i.e* LCPU.CURP. If the suspended process is of higher priority, then the system ought to be rescheduled to allow the suspended process to run as soon as possible. This is achieved, in the real system, by triggering a special interrupt level in the interrupt triplicates unit; the suspended queue interrupt. This interrupt, being an immediate interrupt, will be steered to LCPU forcing it to reschedule and select the highest priority suspended process. In the simulator, the suspended queue interrupt is represented by a BOOLEAN variable, SUSPQINT. To set it TRUE, a lockout, INTLO unique to the interrupt triplicates unit has to be engaged first. If INTRIP, the interrupt triplicates entity (5.4.9) is not servicing a clock or suspended queue interrupt, it will be activated by the process allocator to service the triggered suspended queue interrupt.

PROCEDURE CPUSCHED (1022 - 1048)

Calls SUSPLO2 procedure first to select the highest priority suspended process to run on this CPU. It then compares the priorities of the processes running on all the CPUs to derive the identity of LCPU, *i.e.* the new CPU running the lowest priority process. This updated reference of LCPU is then passed to INTRIP, the interrupt triplicates unit, after engaging its lockout, INTLO.

REAL PROCEDURE SECS: (1051 - 1068)

This procedure calculates the time taken to insert a task in a process input queue. That time will depend on a number of parameters. They include whether the task inserted is the last in the queue, the queue was empty to start with, the process is in a blocked state etc. The time, also includes that required to engage the PROLO lockout.

PROCEDURE HANDTASKANDSETSTATE: (1071 - 1134)

It inserts the task to be handed into the process's input queue ranked by its priority. The task priority is obtained from the task index table of the calling process *i.e* the process handing the task.

The procedure then checks to see whether the called process is a periodic one (*i.e* the process handed the task). This will be indicated by the flag PERIODIC in the called process being set TRUE. If this is the case, the procedure will check whether the task just handed is an INTIM, periodic unblocking task. This is indicated by the task having a priority value of 5. The called process will also be checked for a blocked state. If both conditions are true, the process state will be changed to suspended (unblocked) and the process inserted in the suspended state map.

If the called process is not a periodic one but is blocked, the procedure will check whether the priority of the task just handed is greater or equal to the priority of the requested task (indicated by the level of the block).

If that is the case, the process state will be changed to suspended (unblocked) and inserted in the suspended state map as before.

At the end of a service routine, the process allocator always slects the next process to run as follows:

The process allocator checks the queue LPAQ. If INTRIP is not waiting then the selected entity is scheduled to be activated after the process allocator entity. Otherwise if LPAQ is not empty, INTRIP's reference CUCP is made to refer to the selected or calling process and INTRIP (which is the entity waiting in LPAQ) is scheduled to be active after the process allocator. In either case, the process allocator passivates awaiting a service call. When activated by a service call it goes out of the queue and repeats the service cycle.

The above-mentioned procedures matches closely the equivalent routines of the process allocator in microprogram and represents its most important attributes.

The process allocator model is always in one of two states : either servicing a call or an interrupt or waiting passively for a service call. (Figure (4.6.1) is a gross state transition diagram (STD) of the process allocator life cycle

By having the process allocator model waiting passively or 'going to sleep' until awakened by a service request, considerable saving is made on the model runtime as compared to the other alternative of having it checking continuously for the arrival of a service request. The former case is coded in SIMULA as:

```
WHILE NOT (CALL FOR SERVICE) DO PASSIVATE
ACTION -
```

Whereas the latter case is coded as:

```
WHILE TRUE DO
ACTION -
```

The penality in the former case which is more efficient is that the activation of the process allocator must be made explicitly by the call requesting entity. That entity or process model will transmit the requested call service index to the process allocator activates its process allocator and then passivates.

There are five types of calls for service a process can issue to its process allocator. These are:

a) HAND - to hand a task to another process.

b) FETCH(N) - a request to fetch a task of priority N or less from the process input queue.

c) SEEK(N) - as for FETCH(N) except that the task priority should be exactly equal to N.

d) BLOCK(N) - as for FETCH(N) except that the process wishes to stop running and be blocked if the requested task is not found.

e) SELF(N) - as for HAND except that the task is inserted in the process's own input queue.

The calls to TRAP and UNTRAP a process are irrelevant to the objective of this simulation and hence not considered.

When the process allocator model is activated either by its running process or the interrupt triplicates it checks whether the requested service is for an interrupt or a process allocator call. This will be indicated by the BOOLEAN variable INTERRUPT, which is local to the process allocator,

being TRUE or FALSE respectively. This variable is always set by the interrupt triplicates before 'awakening' the process allocator to handle the interrupt.
If it is an interrupt, the process allocator model will nest (store away) the general registers of its CPU (GO - G7) into the process descriptor of the running process (PD(24:31)). The process descriptor is a per-process array (PD(0:31) to store the environment of an interrupted process and any other relevant information. The CPU condition codes will also be stored in PD(23). The running process state is then changed to suspended (interrupted) (PD(8) = 2) and the process included in the suspended state map, SUSPMAP. The process allocator then check INTOLO lockout, and when free engages it for a period of 46.8 micro-seconds. The BOOLEAN variable CLOCK is checked next for a clock interrupt. It is set by the entity CLOCKINTERRUPT (1756) every 10 msec.

If it is a clock interrupt, then it is time to check the list of periodic processes and see if any are due to be activated. The process allocator fetches a free task block from the free tasks list after engaging FREELO lockout. This task block is used to store the process indices of the periodic processes to be activated at that clock interrupt. The process indices are read from the periodic processes table, PPTABLE; a two-dimensional Array indexed by a pointer which is incremented at every clock interrupt. The minimum length of the arry PPTABLE is determined by the periodicites of different processes as the least common multiple of the periodicities *e.g* if process X has periodicity of 2 (activated every other clock interrupt), process Y periodicity 3 and process Z periodicity 4, then the minimum length of PPTABLE is 12. When POINTER reaches the value of 12 it is reset to 1. The Array PPTABLE is part of the interrupt triplicates model data structure.

The process allocator will hand this task to INTIM which will be blocked waiting for it and hence will be suspended (unblocked). The process allocator then schedules its CPU (CPUSCHED) which sets the reference variable UNSUSPENDED to refer to the selected suspended process. This is then followed by a system schedule (SYSCHED) to ensure that INTIM will soon be run. The process allocator checks whether the process is suspended (Interrupted) or suspended (unblocked) by examining the value of PD(8). A value of 2 indicates the former state and that of 3 the latter. This information is required to restore the values of the general registers.

The selected process state is then set to running (PD(8) = 1) and reference by CALLINGPROCESS. This reference is used in the communication between the selected process and its process allocator (1227).

If the selected process was suspended (interrupted) then the general registers and condition codes value are de-nested (copied back) from the process descriptor of the selected process (1230 - 1233). The de-nesting of the condition codes is to cater for the case of a process issuing a SEEK, FETCH or BLOCK call to its process allocator. The process allocator finds the requested task, but before exiting back to a process an interrupt signal arrives and the process pre-emptied, *i.e.* suspended (interrupted). To enable it to check on the result of its last call when allowed to run again, the value of the condition codes need to be nested and de-nested later. A positive value of the condition codes indicates that the requested task is found.

The process allocator next checks to see if any interrupts are pending. This will be indicated by the interrupt triplicates waiting passively in LPAQ. If no interrupt is pending, it then checks whether the selected process was last interrupted while processing *i.e.* in the middle of a 'HOLD' statement, which represents the passage of time taken by the activity on which the process is engaged. If this is the case, then the remaining activity time would have been calculated by INTRIP, the interrupt triplicates and stored in the real variable REMAININGACTIME. So if this variable is greater than zero, it will schedule the selected process at a simulated time equal to the current simulated time plus REMAININGACTIME, otherwise it is scheduled after the process allocator.

On the other hand if INTRIP is waiting for the process allocator in LPAQ to finish its current service call, then INTRIP is scheduled after the process allocator passivates and a reference of the selected process is passed to INTRIP.

If the last state of the selected process is suspended (unblocked) *i.e* PD(8) = 3, then the unblocking task will be linked out of the input queue and its contents loaded into the general registers before either scheduling the selected process or INTRIP in the manner described above.

When activated by a service call, the process allocator always checks for the arrival of an interrupt first by checking the flag INTERRUPT. If not an interrupt then it is a process allocator call. The process allocator uses a call index transmitted by the calling proces (CIX) in conjunction with a SWITCH statement to branch to the appropriate call servicing routine. (1144) .

For a FETCH(N) call (1350 - 1440), where N is the minimum priority of the task to be fetched, the process allocator first engages the calling process's PROLO lockout. This may imply a delay. All the lockouts are simulated as resources for which all process allocators compete. Having engaged PROLO, it scans the input queue.for the task requested. If the task is found, it will be unlinked from the input queue and PROLO released. The task contents (words 3 - 10) are copied to the CPU general registers (Procedure LOADTASK). Lockout FREELO is engaged and the task block returned to FREETASKLIST, so that it can be used again. The condition codes are set to indicate a successful fetch. On the other hand, if no appropriate task is found, then PROLO is released and the condition codes are set to zero. In either case, the process allocator entity will reach by then the end of FETCH(N) call servicing routine. What remains is to decide on the next entity to schedule. If the interrupt triplicates unit is waiting to interrupt this process allocator then the triplicate unit (INTRIP) is activated else the calling process is activated. The process allocator passivates awaiting the next cycle of service.

For a SEEK(N) call, the course of action taken by the process allocator is identical to that for a FETCH(N) with the exception that the task is to have exactly a priority equal N (1443 - 1495).

A BLOCK(N) call (1498) is identical to FETCH(N) in case the requested task is found. If the task is not found, then the process state is changed to blocked in its process descriptor (i.e. PD(8) = 4) and the priority of the requested task (N) is stored in PD(9). The condition codes are set to zero and PROLO released. The process allocator model then executes a CPUSCHED. Depending on whether the selected process is suspended (interrupted or unblocked) the working environment of the selected process is restored as described above starting at the label RUNSSELECPROC (1215).

If the call index translates to a HAND call (1606 - 1672) the process allocator determines the value of the outgoing task index (OGTI) from the register G(0). This OGTI is supplied by the calling process wishing to hand a task, to enable the process allocator to obtain information such as the identity of the called process, incoming task index for the called process and the task priority. Such information is obtained by using the OGTI as an index down the calling process TIT (301). An error message will be outputted if the array subscript storing the called process index is found to be zero.

The CPU general registers are defined by the process model ARRAY G(0:7), an attribute of entity AP (301) *i.e.* instead of writing the task contents into the CPU registers (G(0:7) local to CPU entity), they are copied to the process model G(0:7).

The same is done when requesting a task. This has the advantage that inside its body, the process model can access and interrogate G(0) - G(7) without the need for the remote accessing of the CPUs G(0) - G(7) using the dot (.) or INSPECT constructs of SIMULA (*e.g.* RELEVANTCPU. G(0) vs G(0)). This strategy gives the software engineer the illusion that he is dealing with a real process as he can refer to the CPU registers directly, the thing he or she is accustomed to in practice. In turn this will make it easier for the software engineer to design and code models of real processes.

The process allocator on servicing a HAND call, engages FREELO lockout to fetch a free task block. It loads the information into the free task block both from the process's TIT and its registers. The reference variable TASKHANDED references this task block before calling the procedure HANDTASKANDSETSTATE. If the called process is suspended by the procedure, then SYSCHED is called to re-schedule the whole system in case the process just suspended is of a higher priority than a running process. The reference TASKHANDED is reset to none and the activation of the next process to run is identical to that of the FETCH(N) call.

A SELF(N) call (1675 - 1712) is issued by a process wishing the task to be inserted in its own input queue, with priority N. The process allocator entity fetches a free task block - after engaging FREELO lockout. The OGTI is copied straight as the ICTI (Incoming Task Index). The Incoming Task Index is used by a process to determine the course of action in responde to the task received.

The reference CALLEDPROCESS is made to refer to the calling process entity. Thus there is no need to access the entity's TIT. The contents of G(0) - G(7) are copied to the task block and HANDTASKANDSETSTATE called in followed by SELECTPROC as before.

The initialisation section of the simulator creates the process models (entities), hardware models, resources, queues *etc.* in a configuration. It then initialises the TITs of the process models and starts the simulation running by scheduling CLINT. The simulator progresses in simulated time where the simulator components act and interact reproducing the system dynamic behaviour. At the end of the simulated period a summary report is outputted.

5.4.11 Activity Durations

In the majority of cases, the duration of an activity is of a fixed length of time. In such a case the duration of an activity is a function of the number of program steps that constitute the activity and the speed of the computer. This activity is modelled by modelling the events that constitute the cause and effect of the activity plus a statement that models the passage of the activity time. As mentioned fore, this corresponds in SIMULA to the statement HOLD(T), where T is the activity time. This procedure was followed in calculating the durations of such fixed time activities.

Another type of activity will depend on some other additional parameters. For example, in searching the suspended state map for a suitable suspended process, the time of a search will depend also on the location of the process index in the suspended state map.

In this case the time T of this activity may be expressed as

$$T = 16.6 + 4 \times PI//16 \quad \mu sec.$$

where

PI is the process index
// is the integer division.

The constant 16.6 represents the fixed time overhead. The denominator 16 stems from the fact that Mark II BL is a 16-bit machine, and that the position of the bit representing the process (Bit = 1 if process in map, = 0 if not) reflects the process index and hence its priority. The number 4 is the time to scan a whole word (16 bits).

Another example is the time required to insert a task in a process's input queue. This will depend on factors such as the length of the queue, whether the task is inserted as the last task, the process is in a blocked state and the task is unblocking. Denoting these parameters by X, L, B and Y respectively, then

$$T = 28.6 + 10.8L + B + 7.4X + 16.2Y \quad \mu sec$$

This formula is used in conjunction with a HAND call. In a SELF(N) call, the expression reduces to

$$T = 16.8 + 10.8L + 7.4X \quad \mu sec$$

In a SEEK(N) call, the seek time depends on the location of the required task in the input queue. Thus the time taken in a successful seek is

$$T = 33.2 + 10.6K \quad \mu sec$$

and for an unsuccessful seek

$$T = 8.6 + 10.6K$$

where K is the ranking of the task in the input queue. For an unsuccessful seek, K is the queue length.

The duration of servicing a HAND call depends, also, on whether the process handed the task is suspended as a result and whether a suspended queue interrupt is triggered consequently. Denoting these two factors by say, Y and Z, where Y and Z are INTEGER quantities that take the values 0 or 1, this activity duration can be expressed as

$$T = Y(18.6 + 4.0\,PI//16 + 4.8Z)\ \mu\text{sec.}$$

Within the model of the process allocator, the activity of servicing a call or an interrupt may be considered to be made up of a number of smaller activities. When developing the model, the HOLD statements representing the time of each individual 'mini-activity' is located immediately before, within or after the 'mini-activity' model as appropriate. Hence the overall duration of the activity is split up into a number of smaller durations scattered throughout the activity. Although that may entail a smaller increase in the model run time, it is found that by so doing, the model behaviour is more consistent with that of the real system. The detailed dynamic model behaviour is revealed by a trace incorporated in the simulator (see Chapter 6).

5.5. CONCLUSIONS

The description of the simulator program presented in this Chapter represents the basic model of Mark II BL system,*i.e* the models of the relevant hardware and operating system processes. In addition to these models of application processes and application hardware have to be added to make up the overall model of a particular exchange configuration. The above basic model of Mark II BL can be grouped in a SIMULA class and named say MARK II BL *e.g.*

```
CLASS MARK II BL;
BEGIN
ENTITY CLASS CPU; ---;
ENTITY CLASS TASKBLOCK; ---;
ENTITY CLASS INTRIPLICATES: ---;
'
'
'
'
ENTITY CLASS PROCESS ALLOCATOR: ---;
'
'
'
END ** MARK II BL;
```

To run a particular exchange model, it is required to prefix the block containing the description of the application models by CLASS MARK II BL. Hence,the application model will run on top of the Mark II BL model.

Moreover,if the SIMULA compiler supports EXTERNAL COMPILATION, the package (CLASS MARK II BL) is only needed to be compiled once and used as an object-code module to prefix a particular exchange model. This will result in considerable saving in computer time if the model is to be run frequently. Chapters 6, and 7 discuss models of specific applications.

CHAPTER 6

THE VERIFICATION AND VALIDATION OF MARK II BL SYSTEM SIMULATOR

6.1 THE VERIFICATION PROBLEM

One of the best definitions of simulation is by Sayre and Crosson in a discussion of a model of the human mind (MIHR 72):
"A simulation model is a symbolic (as opposed to physical or material) representation of a phenomena or system, yet in contradistinction to mathematical models, the symbols of a simulation model are not all manipulated by a well-formed discipline, such as algebra, the integral calculus, numerical analysis or mathematical logic. Indeed, it is becoming more apparent that such simular models are of a more general nature than those restricted to mathematical operations for their solution and/or evaluation."

The development of such a simulation model of a real or proposed system is a purposeful orderly activity, which is in essence an extension of the scientific method of enquiry (MIHR 71). Purposeful, in the sense that modelling goals are first well defined and understood. It is orderly and planned because a well defined, partially-iterative procedure is available for model development and hence the realisation of the stated goals.

But the credibility of any modelling effort rests on the firm demonstration that the simulator represents reality.

Unfortunately, the problem of verifying and validating computer simulation models has received little attention in the literature. Simulation analysts usually have very little to say about the way one goes about using a simulation model or the data generated by such a model on a digital computer. Most of those who made the attempt restricted themselves to purely graphical (as opposed to statistical) techniques to prove the "goodness of fit" of their simulation model (NAYL 67). An explanation of this phenomena, due to NAYLOR et al., is that "In part, the reason for avoiding the subject of verification stems from the fact that the problem of verifying or validating computer models remains today perhaps the most elusive of all the unresolved methodological problems associated with computer simulation techniques." As a solution to the problem, they suggest a multi-stage verification. The first stage of the methodology calls for the formulation of a set of postulates or hypotheses that describe the behaviour of the system of interest. This set of postulates is normally derived from the analysts' already-acquired knowledge of the system under study or of similar systems which have already been successfully simulated.

The second stage of this multi-stage verification calls for the verification of the postulates adopted, subject to the limitations of existing statistical tests.

The third stage consists of testing the models' ability to predict the behaviour of the system under study. It envisages two approaches here to accomplish that goal; namely, historical verification and verification by forecasting. The essence of these approaches is prediction; for historical verification is concerned with retrospective predictions, while forecasting is concerned with prospective predictions. Historical verification in this sense involves the choice of one historical path along which the system was or could be driven and subsequently comparing the output data from the system with that outputted by the model when it is driven along the same path, *i.e.* under the same environmental conditions. This is actually the approach adopted in the validation of the Mark II BL simulation model, as will be explained later. Though in this case the output data generated by the system cannot be used as a check on whether the model did actually point out the best policy to follow, the actual outcome of an alternative policy, strategy or design selected can be compared with the outcome predicted by the simulation model on which basis the respective policy, strategy or design is chosen. This is the essence of verification by forecasting. On the other hand, Van Horn (HORN 72) depicts two stages to accomplish verification and validation. Firstly, we must understand the behaviour of the simulator itself in terms of relations that exist between inputs and results.

Secondly, and this is often the more difficult task, we have to translate 'learning' from the simulation to 'learning' about the actual system. These two stages are defined respectively as verification and validation. Alternatively, Fishman and Kiviat (FISH 68) divide simulation testing into three categories:

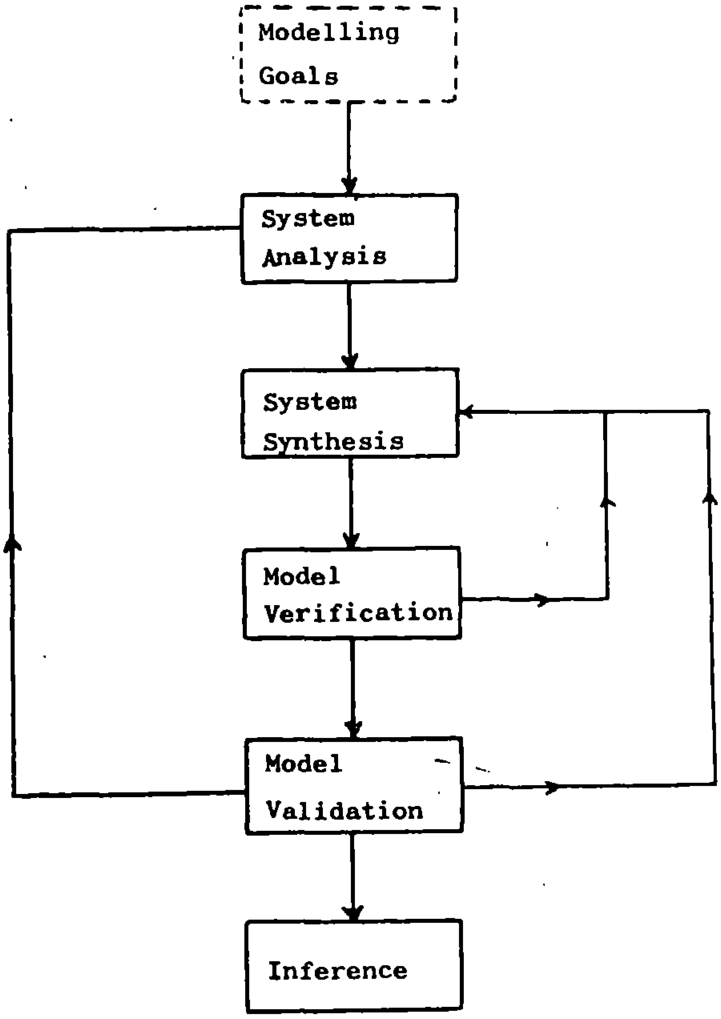
"1) - Verification: ensures that a simulation model behaves as an experimenter intends.

2) - Validation: tests the agreement between the behaviour of the simulation model and a real system

3) - Problem Analysis: embraces statistical problems relating to the analysis of data generated by computer simulation".

A simulation project normally begins by stating clearly the objectives of the exercise in the form of the questions to be answered together with performance measures appropriate for answering them. Following this initial stage, model development is governed by five additional stages given by Mihram (MIHR 72) as follows (Fig. (6.1)):

FIGURE (6.1) : THE MODELLING PROCESS



1) System Analysis

The study of a system in order to ascertain its salient elements and to delineate their interactions, relationships and dynamic behaviour mechanisms. Here, the model entities are identified and their attributes specified. The state variables and transformational rules are enumerated.

2) System Synthesis

The construction of a complete logical structure of the system elements and their interactions to provide a symbolic model of the system. Since this stage realises the model and the computer language influences the realisation, the latter ought to be selected at this stage. Appropriate support data are also determined and collected.

3) Verification

Comparison of the model responses with those which would have been anticipated if the model algorithms structure were prepared as intended. Here, the model is debugged. Tracing routines are very helpful in the verification of the model logical structure by enabling the simulation analyst to analyse closely the dynamic sequence of events as the model components interact in simulated time. If the model fails to compare favourably with predicted theoretical values or known system performance, then a return to the previous stage is necessary (Fig (6.1)). For stochastic models, one-sample statistical tests are appropriate.

4) Validation

The comparison of responses of the verified model with available information regarding the corresponding behaviour of the simulated system. Failure of the model to compare favourably with the real system necessitates a return to Stage 2 (model synthesis). An improper comparison may force the analyst to re-think the abstraction selected in Stage 1 (system analysis).

5) Inference

The contrasting of model responses under alternative input conditions. Experiments using the verified and validated model are designed and conducted. Comparisons are made and recommendations and conclusions drawn to satisfy the goals set at the outset of the simulation study.

Thus, the end result of a simulation model construction is the creation of a credible system representation from which inferences regarding the actual system's performance and behaviour can be made without resorting to costly and time consuming experimentation with the actual system.

Most simulation analysis goals fall into two categories: to ascertain the static effects of the model's performance (which will be indicated by the value of one or more of its state variables or descriptors at the end of a specified period T), or to determine the dynamic performance by observing the behaviour of the state variables during a simulated period of T units, say . The static effect of a model's performance may be represented by the multivariate function S(T), denoted as the simulator response at T .S(T) may be expressed as (MIHR 72):

$$S(T) = R_T(X_1,X_2,\ldots,Xn) + E_T(X_1,X_2,\ldots,Xn \; ; \; S) \qquad \ldots\ldots.(6.1)$$

where

R_T is the simulator response function observed at time T.

$X_1, X_2, \ldots, Xn$ are environmental conditions (input parameters).

E_T is a random response or 'error' of mean zero introduced by randomness in the model.

S is the random number seedor seeds employed explicitly or implicitly in stochastic models.

For deterministic models $E_T(X_1, X_2, \ldots, Xn\ ;\ S) = 0$. The dynamic characteristics are measured by a finite set of values of the form

$$\{S(t), t = 1, 2, \ldots, T\}$$

Where

S(t) is as described above.

A systematic verification procedure involves the determination of a specific set of environmental conditions $X_1', X_2', \ldots, Xn'$ for which the model reponse could be predicted provided that the model is programmed in accordance with the analyst's intentions. The determination of such input/output relationships.

$$S'(T) = R_T(X_1', X_2', \ldots, Xn') + E_T(X_1', X_2', \ldots, Xn'\ ;\ S)$$

is easier to establish once the stochasticity in the model is eliminated. This is also true for the model validation as will be shown later.

6.1.1 The Verification Experiment

The only feasible way to verify the operating system model is to model the software of a typical computer-controlled telephone exchange and run the resulting model in conjunction with the operation system model. If the model is run in a controlled deterministic self-driven mode, then the model static response at time T reduces to

$$S'(T) = R_T(X_1', X_2', \ldots, Xn') \qquad \ldots\ldots.(6.2)$$

This can be compared with the anticipated output for verification purposes. Also, the dynamic response $\{S'(t), t = 1, 2, \ldots, T\}$ must be analysed and studied carefully to ascertain the credibility of the model's dynamic behaviour. For this purpose, an appropriate trace program ~~is~~ must be imbedded in the model to disclose the model components interactions and the state descriptors values as simulated time progresses.

Unfortunately, a typical system X exchange software was not available at the time of model verification. So a model of a hypothetical suite of application programs of a particular system X exchange was developed by the author, based on the anticipated exchange architecture (LAWR 77). The application software modelled belonged to the Digital Main Network Switching Centre (DMNSC).

6.1.1.1 The Hypothetical DMNSC

The basic telephony requirements of application programs may be summarised as follows:

Detect seizure
Determine class of service

Receive and store digits
Translate code digits into routing and call charge information
Set-up own exchange trunking
Forward digits to set-up the rest of the network

Detect called subscriber answer

Detect clear forward or clear backward
Release network
Release trunking.

In addition to the above telephony functions there remains the functions of routing, diagnostics, traffic recording and maintenance control.

The application software is partitioned into a number of software modules with clearly defined interfaces. Each module, known as a process, takes care of one or more of the above-mentioned telephony functions. As long as the interfaces between the processes are strictly observed, they may be developed and tested separately by teams of software engineers, possibly in different physical locations. As an initial stage of partitioning, it may be observed that the above telephony functions could be grouped into three categories each served by an application process (Fig. (6.2)). These processes are:

a) Line Circuit Handler:
This process interfaces with the incoming and outgoing telephone lines and deals with all aspects of signalling.

b) CALL Control:
Responsible for processing telephone calls, establishing the network routing, supervising the call and upkeeping and updating of telephone traffic statistics.

c) Switch Handler:
This interfaces with the exchange switch matrix and handles all call set-up and clear-down functions.

With only three processes, there is a limit to the maximum traffic carrying capacity set when all the three processes are running simultaneously in a multi-processor system. An obvious way to overcome this disadvantage and increase the traffic-carrying capacity is to split up the individual processes on a functional basis. For example, the line circuit handler process may be split into a number of processes; a process for each signalling system *e.g.* a loop-disconnect handler, DC2 handler *etc.* The disadvantage of this method is that an appreciable increase in telephone-traffic handling-capacity is only possible if the circuits for each type of signalling are roughly equal. A better way of splitting is to provide a line circuit handler per group of lines, and to have separate handlers for incoming and outgoing groups of lines.

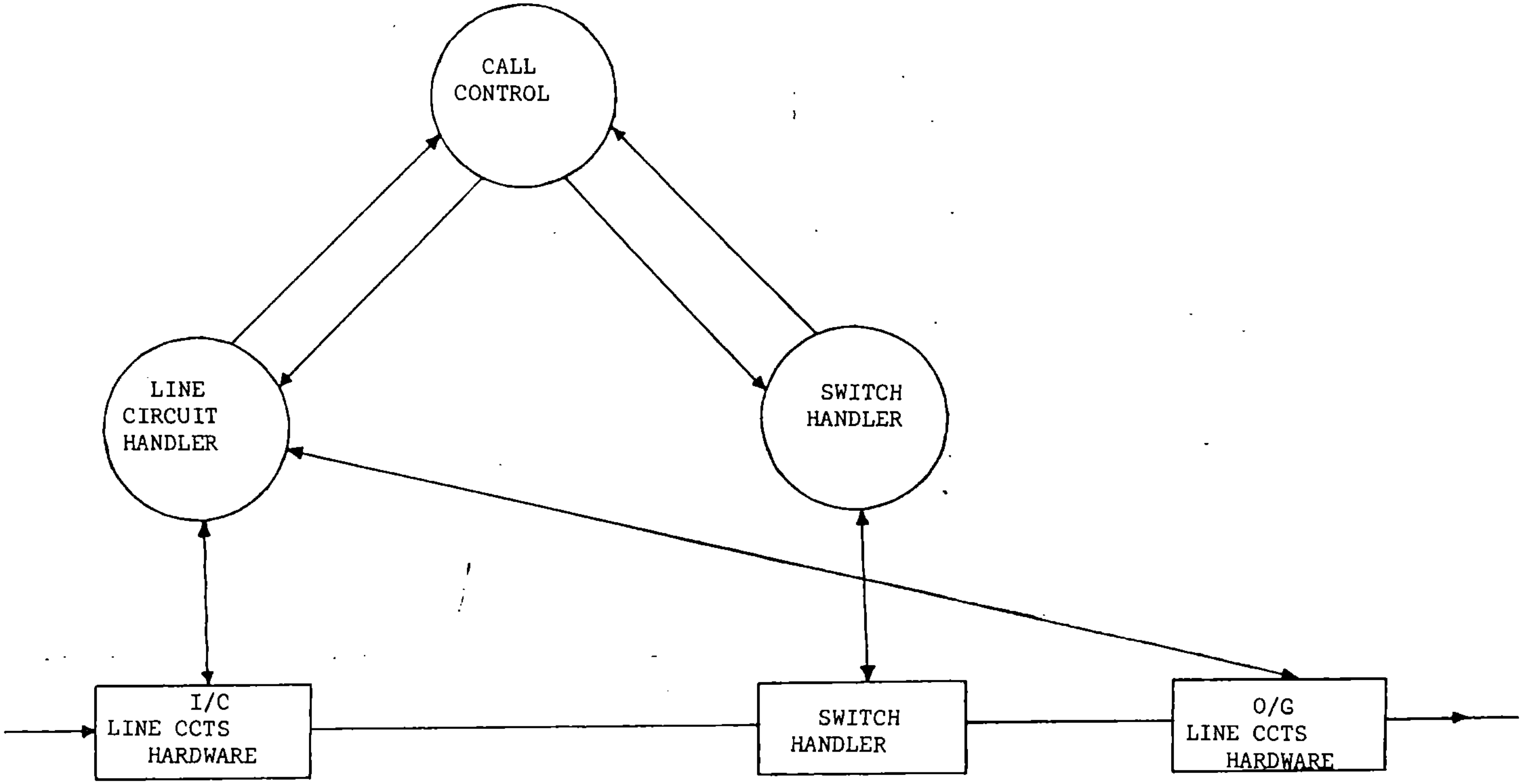


FIGURE (6.2) : BASIC STRUCTURE OF HYPOTHETICAL DMNSC

The call control may be split into three smaller processes *e.g.* call supervision, network routing and traffic recording. Call supervision may be split further into incoming call supervision and outgoing call supervision. The processes may then be replicated to deal with different groups of circuits. For a switch handler, the time to set-up a path is so short in a digital exchange (of the order of a few hundreds of microseconds) that a single process is capable of carrying a few thousands of erlangs. However, for other types of exchanges it depends on the form of trunking, *e.g.* for a cross-bar exchange, a switch handler process may be allotted to each router control with the identity of the incoming circuit determining which router control and hence which switch handler is to be invoked. The resulting software structure is shown in Fig. (6.3).

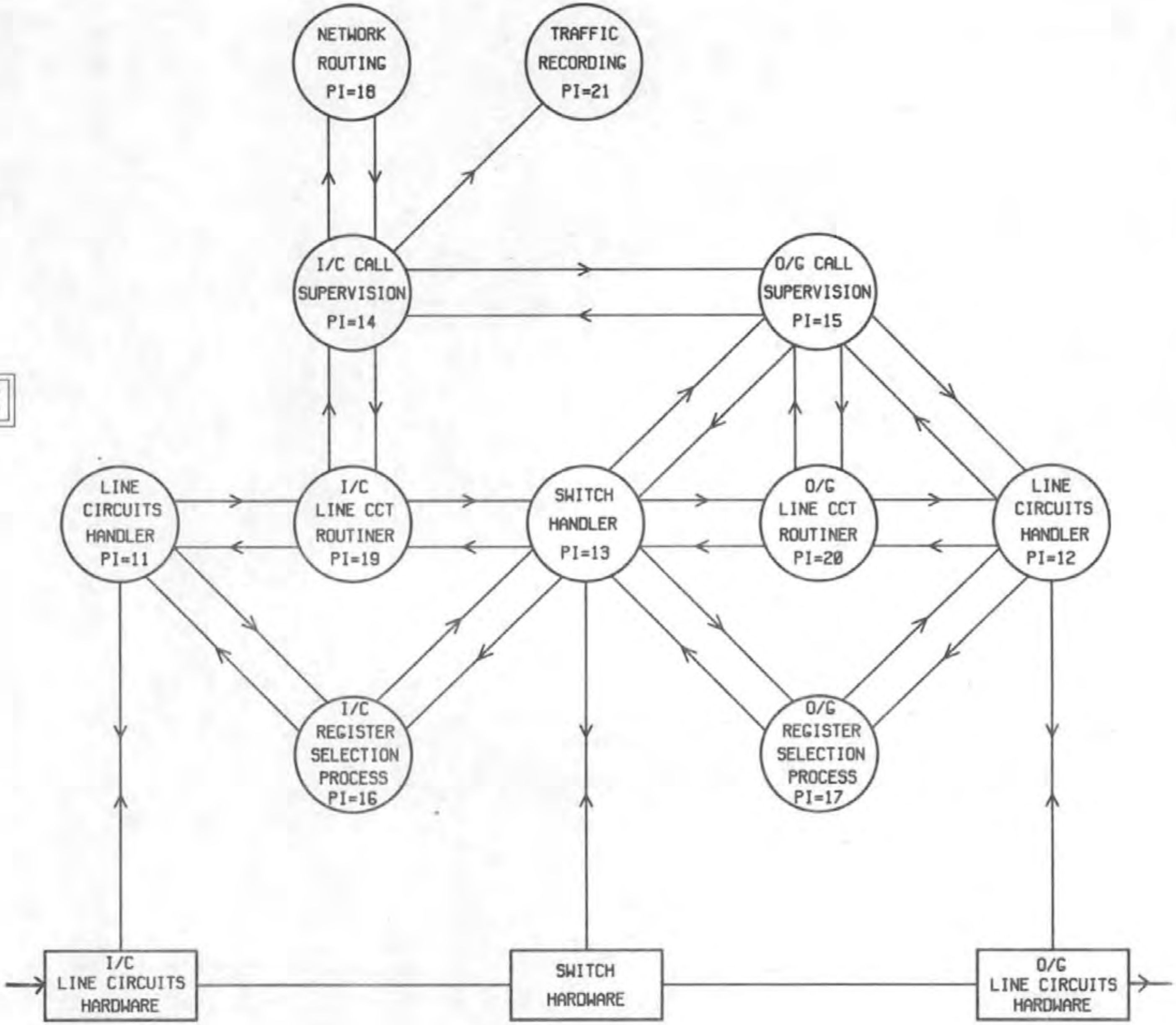
Splitting the processes is not without its own problems. The increase in the number of processes imposes a heavier burden on the operating system to handle the inter-process communication by passing tasks of information or messages between them. This may well become the limiting factor as far as the total system traffic capacity is concerned. This problem may be eased to some extent by having some of the processes periodically activated rather than activated with the arrival of each individual task.

The DMNSC exchange handles only trunks and junctions. Generally, for a certain type of exchange there is no optimum design for its software structure and there is always the trade-off between the conflicting requirements of traffice carrying capacity, economy, efficiency *etc*. The function of the register-selection process in Fig. (6.3) is to select a register if an MF sender or receiver is required and set-up the path between the line and register *via* the switch handler.

INT \ PI	12	13	14	15	16	19	11
1	12	0	14	0	16	0	11
2	0	13	0	0	0	19	11
3	12	0	0	15	0	0	11
4	0	13	14	0	0	0	11
5	12	0	0	0	16	0	11
6	0	13	0	15	0	0	11
7	12	0	14	0	0	0	11
8	0	13	0	0	0	19	11
9	12	0	0	15	16	0	11
10	0	13	14	0	0	0	11
11	12	0	0	0	0	0	11
12	0	13	0	15	0	0	11

POINTER

PERIODIC PROCESS TABLE


NETWORK ROUTING PI=18
TRAFFIC RECORDING PI=21
I/C CALL SUPERVISION PI=14
O/G CALL SUPERVISION PI=15
LINE CIRCUITS HANDLER PI=11
I/C LINE CCT ROUTINER PI=19
SWITCH HANDLER PI=13
O/G LINE CCT ROUTINER PI=20
LINE CIRCUITS HANDLER PI=12
I/C REGISTER SELECTION PROCESS PI=16
O/G REGISTER SELECTION PROCESS PI=17
I/C LINE CIRCUITS HARDWARE
SWITCH HARDWARE
O/G LINE CIRCUITS HARDWARE

ENHANCED DMNSC STRUCTURE AND PERIODIC PROCESS TABLE

The processes are table-driven. The set of processes that a process is allowed to communicate with is strictly defined in the process's task index table (TIT). The destination of a message and the response of the destination process are clearly defined for a process by the outgoing task index (OGTI) value that a process passes to the real-time operating system when requesting a task to be handed. The operating system then uses this outgoing task index value to index down the process's task index table to obtain information such as the identity of the called process, the task priority and an incoming task index for the called process. The incoming task index (together with the task contents) is used by the called process to select the appropriate action and pass further messages to other processes and so on. Thus, if the different values for the incoming and outgoing task indices are defined for each process, the task index tables can than be compiled (with suitable choice of task priorities) to determine the sequence of operating the exchange.

If this sequence can be deterministically identified before hand, then it may be used to compare with the dynamic interactions between the software processes and the operating system and hence constitute a powerful verification tool. For the sake of obtaining a detailed record of this dynamic interaction, a special trace program is written and incorporated in the model. The characteristics of this trace routine follows.

6.1.1.2 The Trace Program

The objective of the trace is to output dynamically values of the state variables and the transformational rules applied to change them. The contents of the operating system global tables,such as the map of suspended processes ready to be selected to run (Suspended State Map) and the free task list are continuously outputted whenever they are accessed by one of the process allocators, together with an indication of the value of simulated time. The suspended state map trace gives the names and process indices of the processes in the suspended state map. Other state variables such as the suspended queue interrupt (SUSQINT), the clock interrupt (CLOCKINT) and the reference to the CPU running the lowest priority process (LCPU) are also outputted, whenever their values or references change. Moreover, when a suspended queue interrupt is triggered an indication is given as to the reason for that, in the form of the identity of the highest priority suspended process, the priority of the process running on LCPU (*i.e.*LCPU CURP) and the identity of LCPU.

Whenever a running process issues a call to the process allocator of the CPU in which it is running, the identities of the call, the process and the CPU are indicated. For a HAND call, the call index, the outgoing call index and the called process identity are also outputted. This is quite a useful piece of information to use in checking against the task index tables of the calling and called processes to see whether the process allocator extracts the right sort of information regarding the called process, the task priority and the incoming task index for the called process. It can also be used to check on the reaction of the called process to the task handed which is supposed to be governed by the value of the incoming task index in most cases. In other cases, the response of a process depends on the contents of one or more of the general registers (scratch pad) *e.g.* in the validation experiment, the storage allocator response to requests from RASH replicates depends on the value of G2 which is indicative of the nature of the request (see Section 6.2.). For this reason, the contents of a task (Go - G7) are always outputted whenever the task is loaded in the simulated CPU general registers. The number of tasks in the input queue of the called process and the task priorities are also indicated and whether the process being handed the task changes state as a result of that and hence its insertion in the suspended state map.

For a FETCH or SEEK call, the identities of the calling process and its CPU are given and whether the call is a successful one or not. For a BLOCK call, in addition to that, the trace indicates for an unsuccessful BLOCK call, the identity of the newly selected process and its last state, it being suspended (Interrupted) or suspended (unblocked).

In addition, the trace program outputs error messages whenever a malfunction of the operating system occurs. Some error numbers lead to the abortion of the simulation run. A sample of the trace program output is shown in Appendix (C).

Since the process allocator is at the heart of the real-time operating system and handles the scheduling of processes, the interrupts servicing and the inter-process communication, the trace program is embedded in the process allocator. The trace can be switched on and off to run for certain durations of simulated time.

The detailed trace output makes it feasible to scrutinize the behaviour of the operating system model in a complex multi-processor environment and verify the different algorithms within the operating system.

6.1.1.3 The Verification

A great care was taken in the splitting of the software structure of Fig. (6.3) into periodic and aperiodic processes, in the choice of process indices and in the design of individual processes task index tables. This was to ensure an even distribution of load among the processes and the ultilization of all types of services offered by the real-time operating system. The following software processes were selected to be periodic. The name of the process is followed by the process index number followed by the process periodicity *i.e.* how often a process is activated.

Periodic Processes:

Line Circuit Handler 1 (LCH1)	11	11 msec
Line Circuit Handler 2 (LCH2)	12	22 "
Switch Handler (SH)_	13	22 "
Incoming Call Supervision (ICS)	14	33 "
Outgoing Call Supervision (OCS)	15	33 "
Incoming Register Selection Process (IRSP)	16	44 "
Incoming Line Circuit Routiner (ILCR)	19	66 "

The minimum length of the periodic processes table (part of the interrupt triplicates unit data) is the least common multiple of the periodic processes periodicities, in our case that of 1, 2, 3, 4 and 6 *i.e.* 12. The periodic processes table is shown in Fig. (6.3). Its pointer is incremented with the arrival of each clock interrupt or reset to 1 whenever its value reaches 12. As decribed in Chapter 5, INTIM process (Interrupt and Timing Process) accesses this table at the arrival of a clock interrupt, using the most recent pointer value to obtain the identity of the periodic processes to be activated at that time by handing them periodic unblocking tasks of high priority.

A process is to be activated at that particular clock interrupt if its corresponding entry in the row indicated by the pointer value is non-zero. On the other hand, the aperiodic (non-periodic) processes and their process indices were selected as follows:

Outgoing Register Selection Process (ORSP)	17
Network Routing (NR)	18
Outgoing Line Circuit Routing (OLCR)	20
Traffic Recording	21

All processes are table-driven. In most cases the response of an application process depends on the value of the CPU condition codes after a FETCH, SEEK or BLOCK call to the process allocator *e.g.* LCH1, OCS *etc.* When the condition codes are positive, the response of a process may be dictated by the value of the incoming task index stored in GO *e.g.* IRSP, ILCR, SH *etc.*, or by the contents of one of the other general registers *e.g.* IRSP, LCH2.

The processes and their task index tables are shown in Fig. (6.7) - Fig. (6.17). At the end of its processing cycle, a process normally picks up any remaining tasks in the input queue using BLOCK(15) call, if any, otherwise it is blocked until activated by INTIM or the arrival of a task, depending on whether it is periodic or aperiodic. This last bit of process logic is coded in a separate procedure - LASTJOB.

By following closely the detailed trace output of the system dynamics as it progresses through simulated time, and taking into account the structure of the processes and their task index tables, the logic of the simulator is verified against the expected behaviour of the real system and the intentions of the simulation analyst (Author).

In many occasions, the coding of the simulator algorithms had to be changed, especially the interrupt handling and the interaction between the process allocators and the interrupt triplicates unit. For example, if a running process was interrupted and pre-emptied during a HOLD statement, then the remaining activity time represented by the HOLD statement had to be calculated and stored. In another instance, it was noticed that the local sequence control (LCS) of a process would change its position if the process was interrupted while being selected to run. When it was selected again to run, its LCS skipped one or more statements of its coding. This unexplicable behaviour of the SIMULA system seems to have been caused by a combination of scheduling statements being executed in quasi-parallel. When the scheduling statements at the end of each branch of the process allocator were re-arranged, this phenomenon luckily disappeared. A third and rather tricky problem was the division of an activity time (*i.e.* time consumed by a particular activity which is represented by a particular code) into smaller time slices and the insertion of those in HOLD statements at the right positions within the SIMULA coding of that activity in order to obtain the correct dynamic interactions. This 'tuning' process was a delicate and time-consuming one that required a great deal of patience. This iterative process was carried out for different activities until the proper dynamic interactions were obtained.

After a large number of runs, the verification was completed.

6.2 THE VALIDATION PROBLEM

In spite of the relative importance of simulation model validation, very few papers in the literature are dedicated to the subject (HORN 72, UNGE 75). In line with Fishman and Kiviat (FISH 68), Van Horn defines validation as "the process of building an acceptable level of confidence that an inference about a simulated process is a correct or valid inference for the actual process". This definition implies that validation is not intended to be used as a proof that the simulator is a true model of the real system. As a simulator may be considered as a finite-state machine that transforms inputs into outputs, it cannot be proved that two machines are identical just by comparing input - output transformations however large a sample is used (HORN 72). Fortunately, simulation analysts are concerned with validating the insights produced by the simulator rather than proving the truth of the model in every respect. In other words, the validation objective is to validate a specific set of insights not necessarily the mechanism that generated the insights. In this respect, a large number of tools and techniques are at the disposal of the analyst. These range from various statistical tests to complementary studies and field tests (MIHR 72, FISH 67, HORN 72). In his excellent paper, Van Horn listed the following actions that could be taken to achieve the desirable confidence in a simulation model.

1. Find models with high face validity for comparison.
2. Supplement the model by knowledge available from existing or past research, experience or observation.
3. Conduct simple empirical tests of means, variances and distributions using available data.
4. Run 'Turing' type tests.

5. When adequate data is available, apply complex statistical tests.
   These include analysis of variance, regression, factor analysis, spectral analysis and autocorrelation, ψ^2 and non-parametric tests.
6. Engage in special data collection.
7. Run prototype and field tests.
8. Implement the results with little or no validation.

However, validation is characterised as problem-oriented and the real task of validation is finding the appropriate set of actions. For example, Unger (UNGE 75) proved the validity of his computer resource allocation simulation model by comparing the model output with that of a validated analytical model of a known computer configuration. Hence the first action listed above proved to be sufficient for the purpose.

As stated before, a digital computer simulator is a finite-state machine for transforming an input set into an output set. Since insight is gained from the observation and analysis of the transformation process, overall confidence in the insight depends clearly on confidence in the transformation process. One of the obvious ways to gain confidence is to compare the outputs of the simulator and the system using identical input parameters. In such situations, more often than not, simple comparison of means, ranges, variances, tabular and graphical comparisons of performance measures will suffice for the purpose (BELL 72, HORN 72).

Simulators may either be deterministic or stochastic. If stochasticity can be removed from the actual system, the validation is then a straight-forward process. One needs only to observe the output or performance of the actual system under a controlled set of operating conditions and compare these in a one-to-one basis with the responses emanating from the simulator under the same set of operating conditions, *i.e.* compare the simulator responses S(T) of equation (6.1) with Y(T), the response of the actual system.

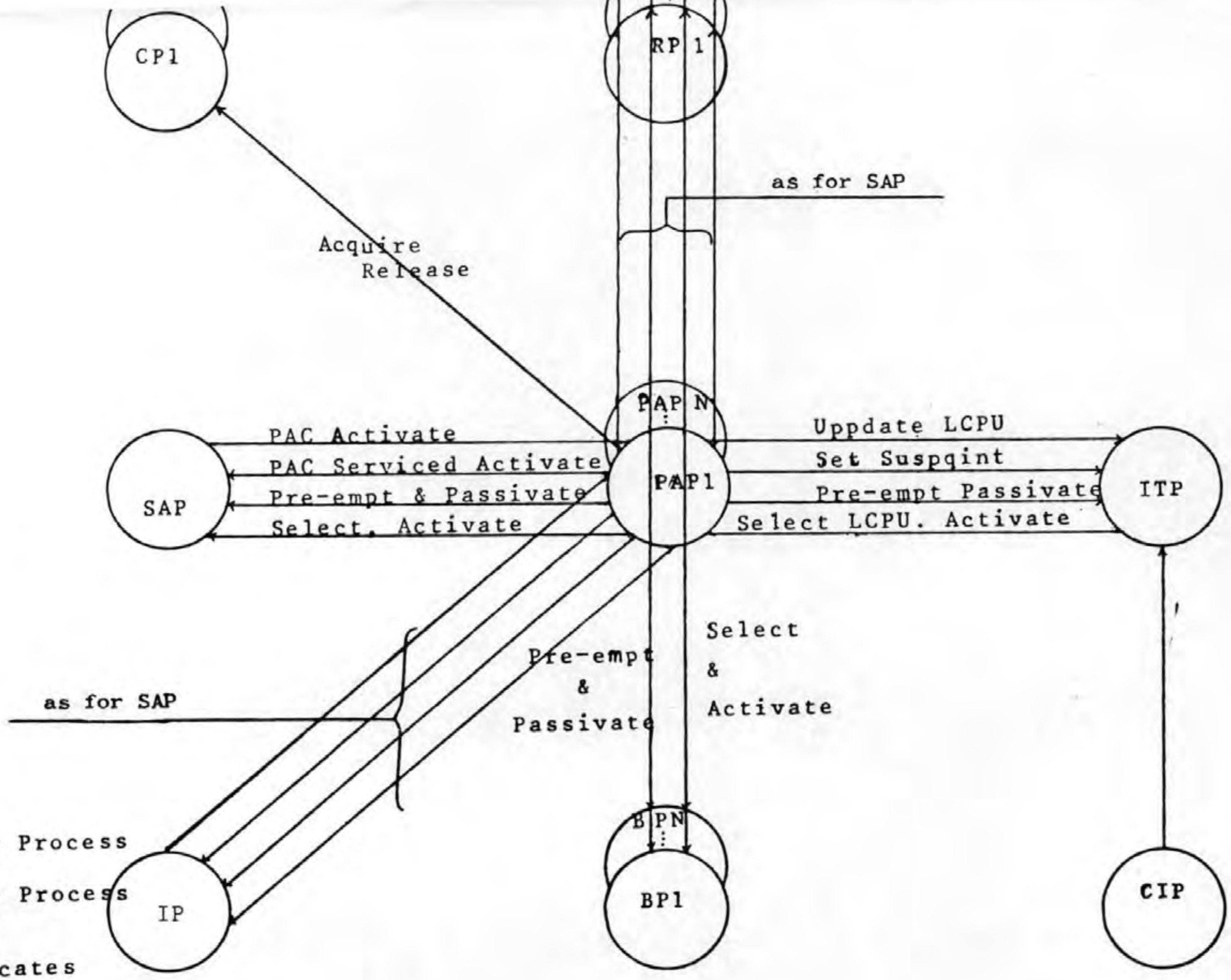
With specific regard to the Mark II BL real-time operating system model, the stochasticity represented by the system workload generator (the arrival of telephone traffic) may be removed and the system be self-driven. This can be achieved by running the system with one or more instances of a specific operating system process representing the system workload generator by demanding service from the operating system in a continuous cyclical fashion, as will shortly be explained.

6.2.1 The Validation Experiment

The Mark II BL processes involved in the validation are shown in Fig. (6.4). The validation experiment consists of nine runs of the simulation model in which one, two and three RASH process instances represent the loading of the operating system. They execute a loop continuously for a pre-determined interval of time. Every time a RASH process instance traverses its loop, it increments an execution counter. The extent of the agreement between the contents of real RASH counters and their corresponding models counters is indicative of the degree of credibility of the simulator. Models of the operating system processes in Fig (6.4) have been described before (Chapter 5) except for the RASH process. An appropriate description follows.

6.2.1.1 RASH process

The Thrash and Crash suite of processes is collectively known as RASH and has been provided within Mark II BL system software to assist in the debugging and commissioning of the real-time operating system. It allow the programmer to write his own test modules to perform functional and performance tests and control their



KEY

PAP: Process Allocator Process
RP : RASH Process
SAP: Storage Allocator Process
IP : INTIM Process
CP : CPU Process
ITP: Interrupt Triplicates process

PAC: Process Allocator Call

CIP: Clock interrupt process
BP: Background process

FIGURE (6.4) SIMULATOR STRUCTURE FOR VALIDATION EXPERIMENT running interactively using the man/machine interface provided by RASH.

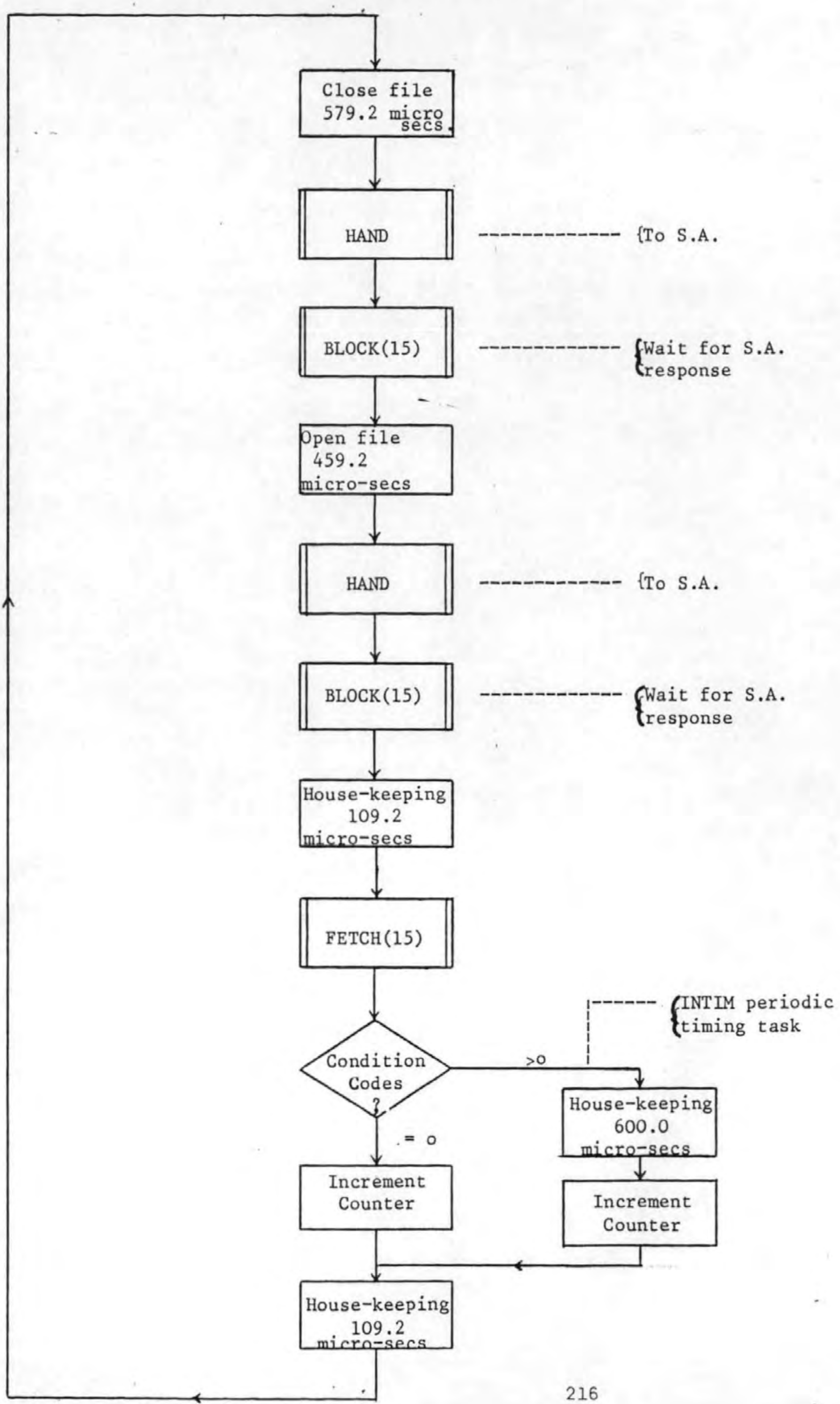
A macro-level flow chart of RASH is shown in Fig. (6.5) and the corresponding model flow chart in Fig. (6.6). The striking similarity between the two figures is yet another demonstration of the main characteristic of this simulator package - namely, the ease with which a real process can be transformed into its equivalent model in a one-to-one translation process.

There can be up to seven RASH processes running concurrently. RASHO is known as the command process and it interprets the commands of the man/machine language used by the programmer to control other individual RASH processes. A RASH process is also called a test process and a test module can be executed by up to six such test processes at the same time.

A programmer may pass data items to a test module before it starts running. A further facility is available to set a sequence of predefined test modules executing within specified processes. The aim is to allow multi-function tests to be initiated by a single command and to restart the tests after every system or process rollback. A complementary command is used to stop the execution of the processes.

There is a basic structure which is identical to all test processes. It is best to imagine each RASH process as containing a loop which it can be forced to enter by giving it a RUN command
The loop itself involves emptying the input queue and dealing with any tasks, followed by entry into the test module specified by the operator. When the module has been executed, a counter, call the Execution Counter is incremented by one to record the fact that the loop

FIGURE (6.5) : RASH FLOW CHART


Close file 579.2 micro secs
HAND
{To S.A.
BLOCK(15)
{Wait for S.A. response
Open file 459.2 micro-secs
HAND
{To S.A.
BLOCK(15)
{Wait for S.A. response
House-keeping 109.2 micro-secs
FETCH(15)
{INTIM periodic timing task
Condition Codes ?
>o
= o
House-keeping 600.0 micro-secs
Increment Counter
Increment Counter
House-keeping 109.2 micro-secs

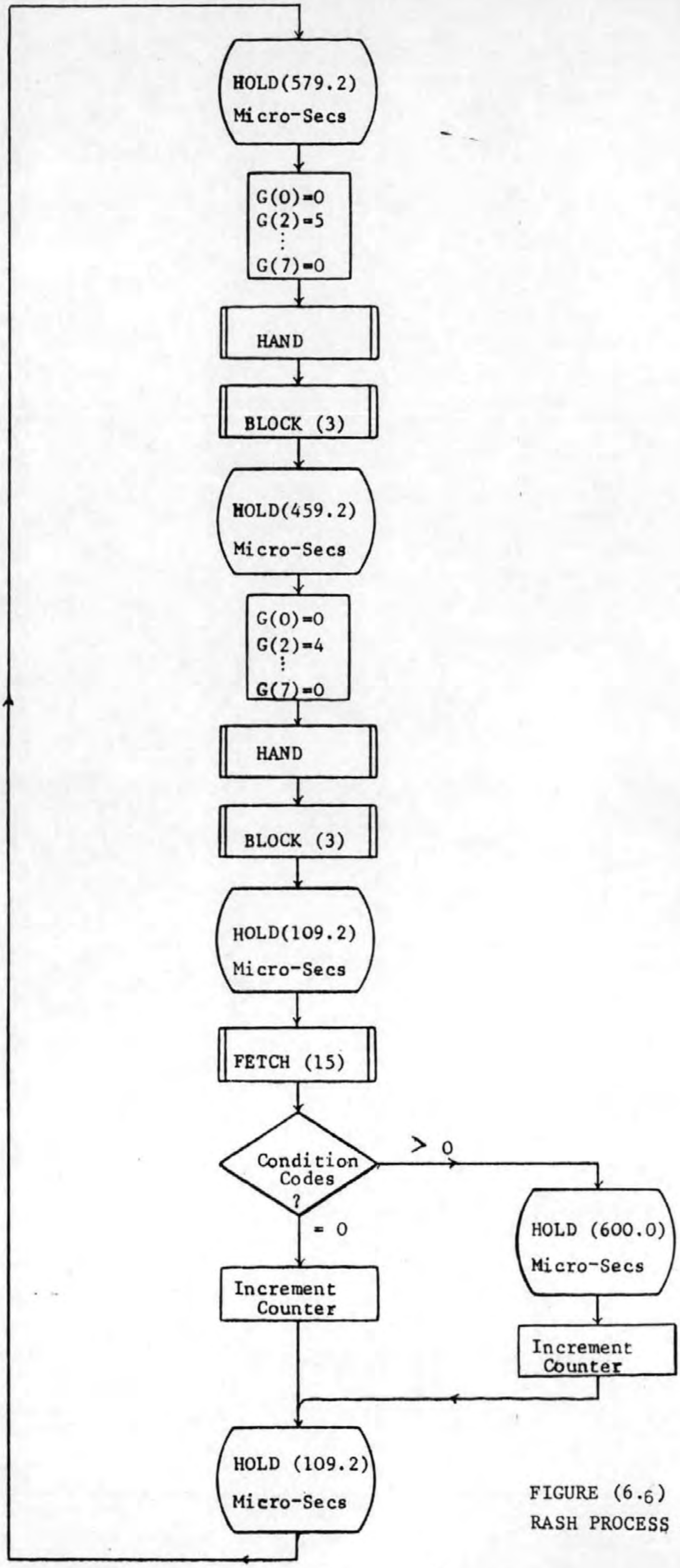


FIGURE (6.6)
RASH PROCESS MODEL FLOW CHART

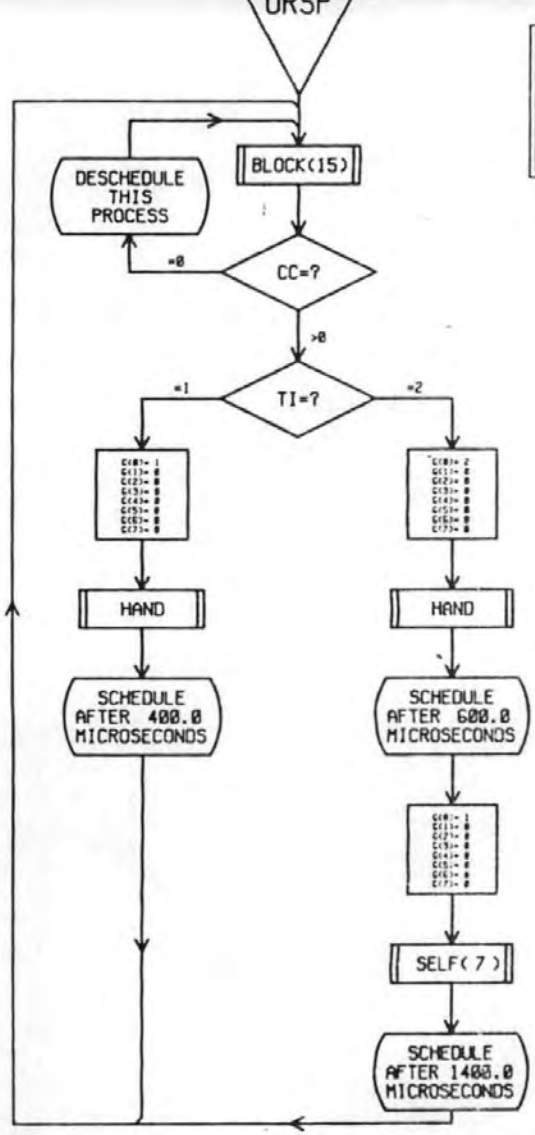
has been completed once more. The loop is then started again. Once the loop is entered it is executed repeatedly until the operator stops the test module by issuing it a FINISH command (Fig.(6.5)).

Another facility available to the programmer is the timing facility of RASH. The runt-time of test programs can be measured by counting the number of times a process executes its loop containing the test module, in a specified period of time. The programmer uses the TIME command to specify which RASH process he wishes to time and over what time interval the loop should be counted. The value of the execution counter will, therefore, be the number of times the test process (RASH) executed the test module in its internal loop, during the specified interval of time. The timing so obtained will include the portion due to RASH overhead and other operating system overheads. An estimate of RASH overhead is easily obtained by repeating the measurements with the test process executing an empty test module.

6.2.1.2 The Experiment

The procedure adopted in the validation experiment is for RASH to run an empty test module and to issue requests to the storage allocator to open and close its files. It is also arranged that it receives a time task from INTIM every 100 msec and that the time interval during which the loop is executed is 7.2 secs (Fig. (6.5)).

The system is run in this deterministic fashion to exclude the element of stochasticity or randomness that would otherwise be introduced by the input process (arrival of telephone calls). The simulator is initialised with the background processes running on the CPUs.



CLASS OUTGOING REGISTER SELECTION PROCESS
PI=17

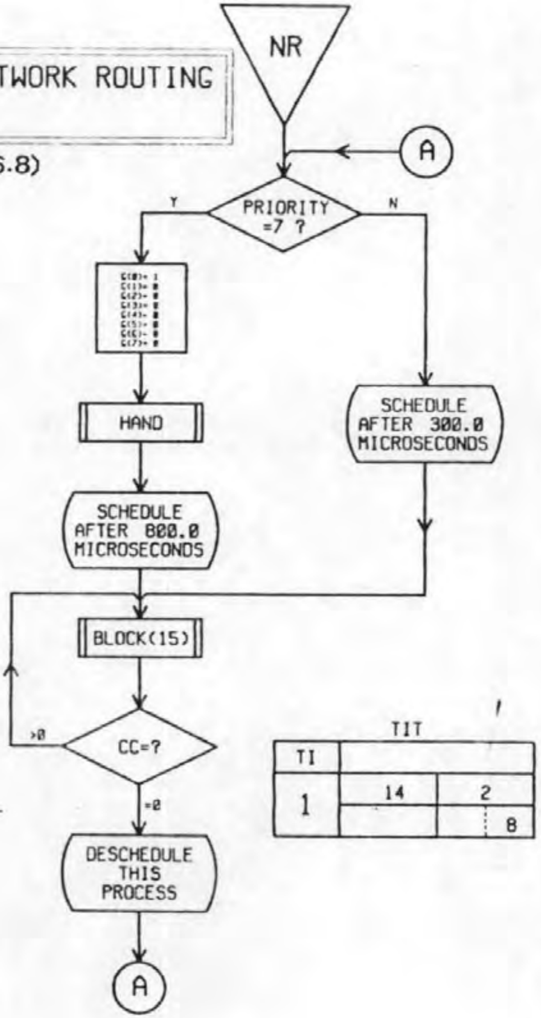
FIGURE (6.7)

TIT

TI		
1	13	6
		12
2	12	3
		5

CLASS NETWORK ROUTING
PI=18

FIGURE (6.8)

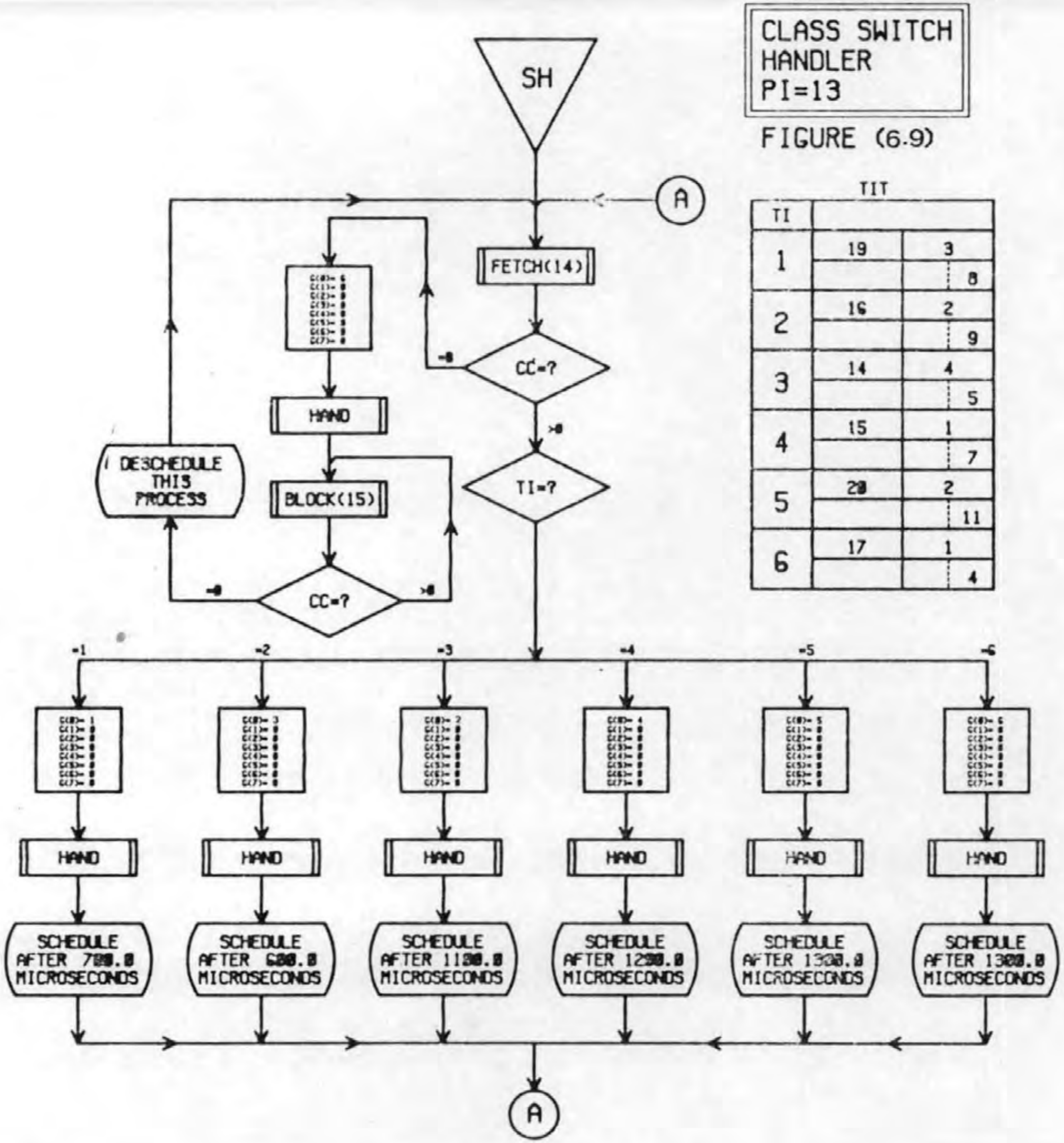


TIT

TI		
1	14	2
		8

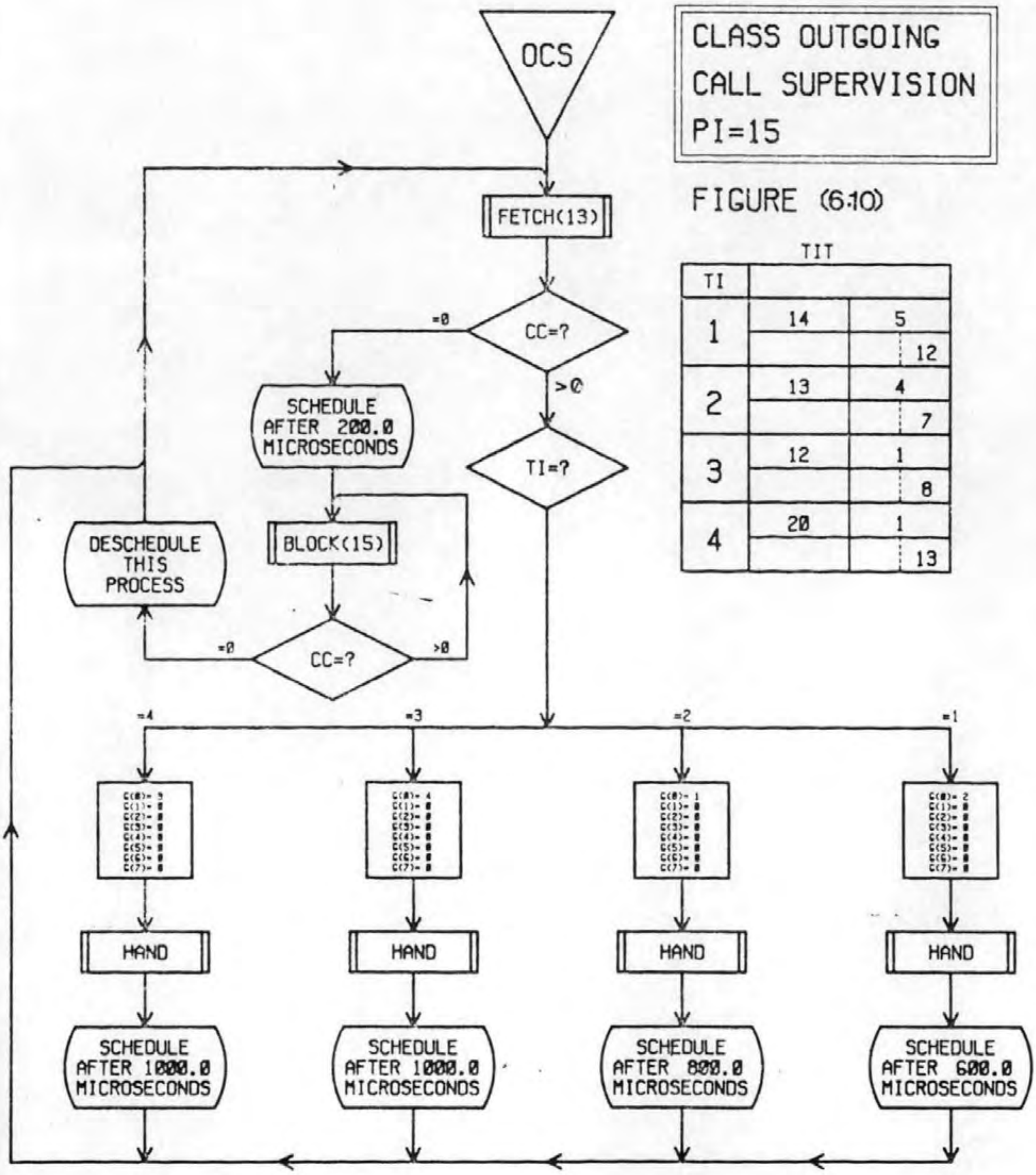
CLASS SWITCH
HANDLER
PI=13

FIGURE (6.9)



TIT

TI			
1	19	3	
			8
2	16	2	
			9
3	14	4	
			5
4	15	1	
			7
5	20	2	
			11
6	17	1	
			4

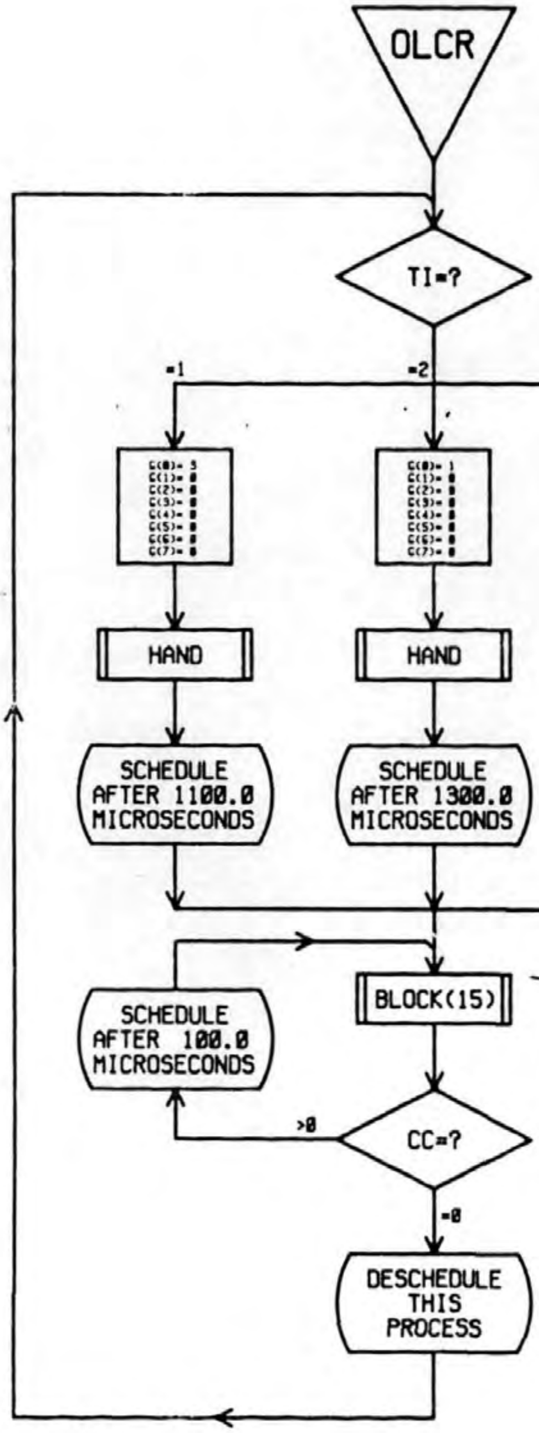


CLASS OUTGOING
CALL SUPERVISION
PI=15

FIGURE (6.10)

TIT

TI		
1	14	5
		12
2	13	4
		7
3	12	1
		8
4	2Ø	1
		13



CLASS OUTGOING LINE CIRCUIT ROUTINER
PI=20

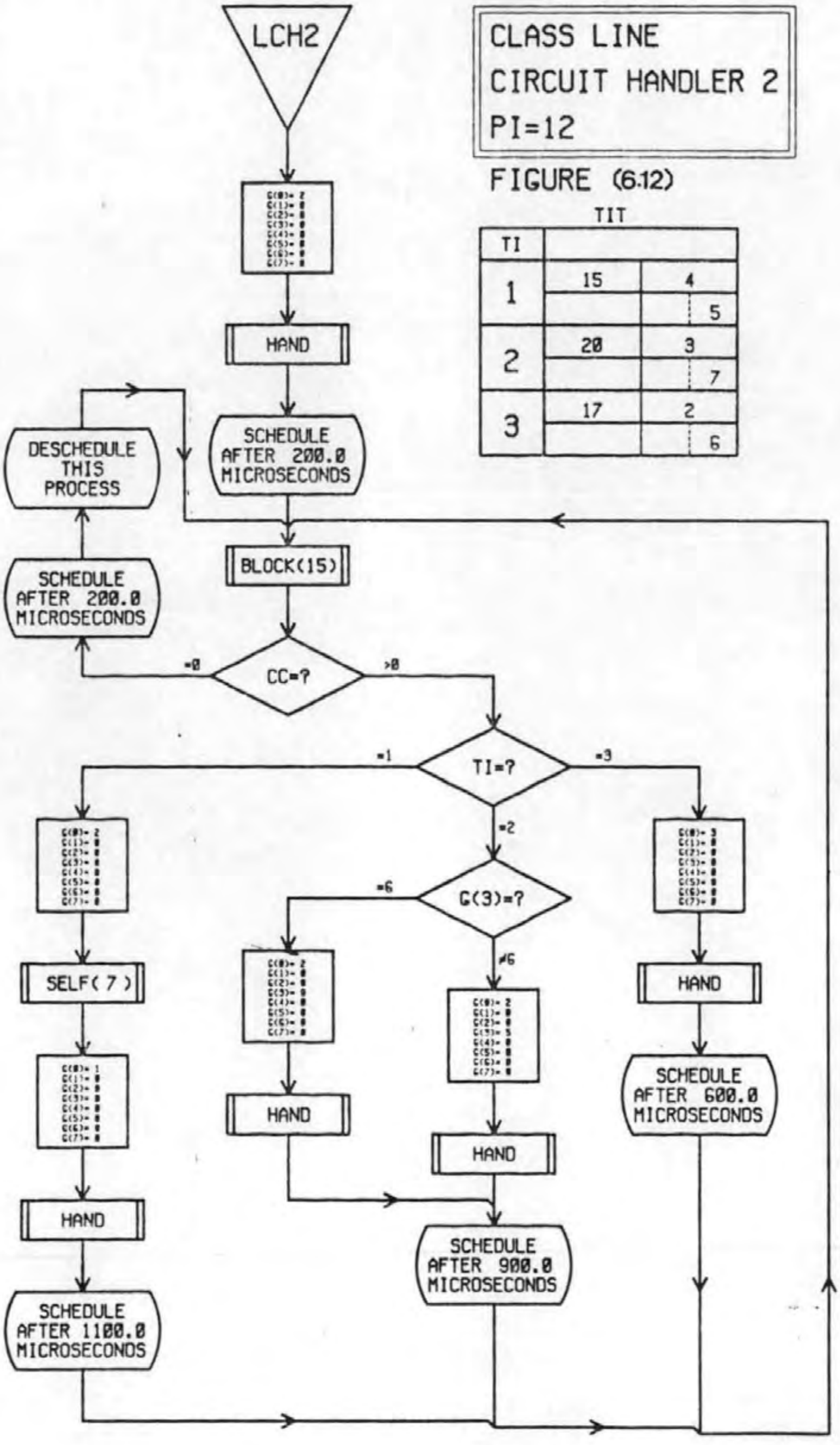
FIGURE (6.11)

TIT

TI			
1	13	5	
			6
2	12	2	
			4
3	15	3	
			7

CLASS LINE
CIRCUIT HANDLER 2
PI=12

FIGURE (6.12)



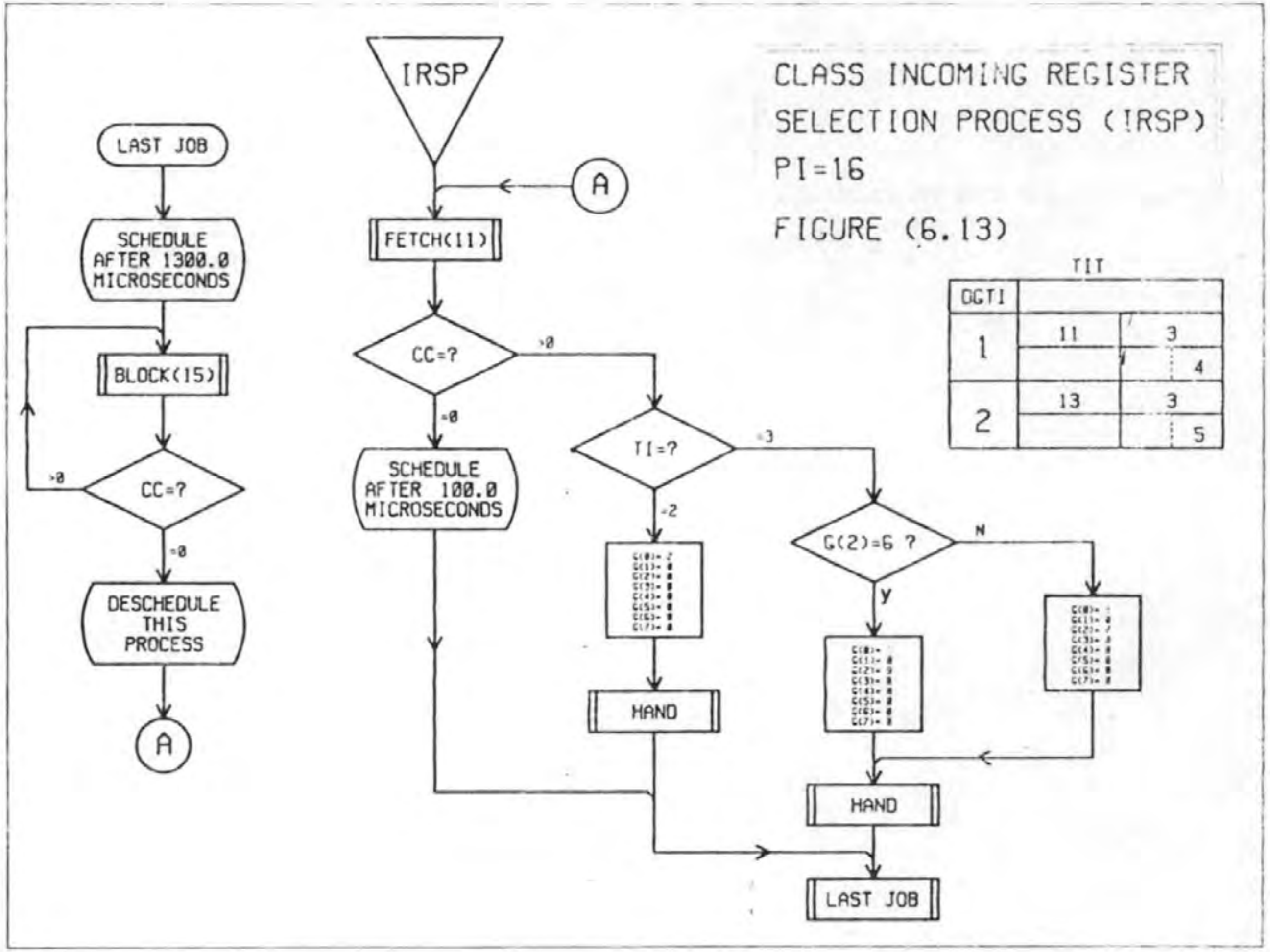
TIT

TI		
1	15	4
		5
2	2Ø	3
		7
3	17	2
		6

CLASS INCOMING REGISTER SELECTION PROCESS (IRSP)

PI=16

FIGURE (6.13)

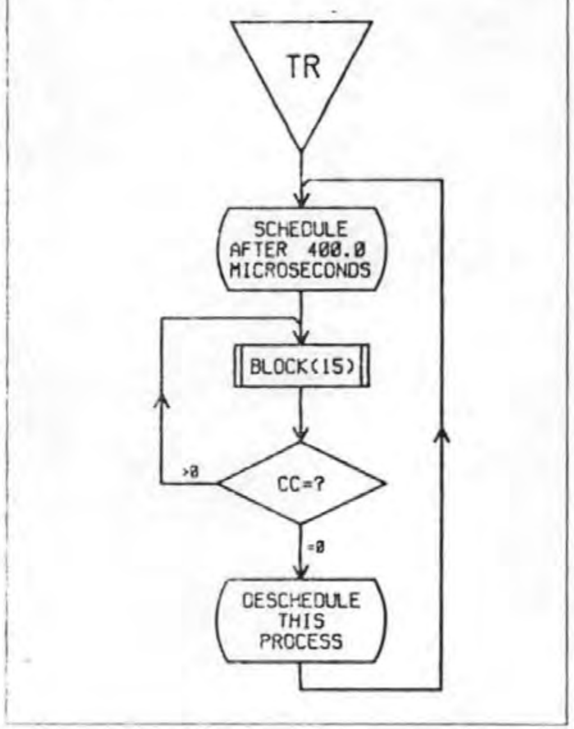


DGTI	TIT		
1	11	3	
			4
2	13	3	
			5

CLASS TRAFFIC RECORDING

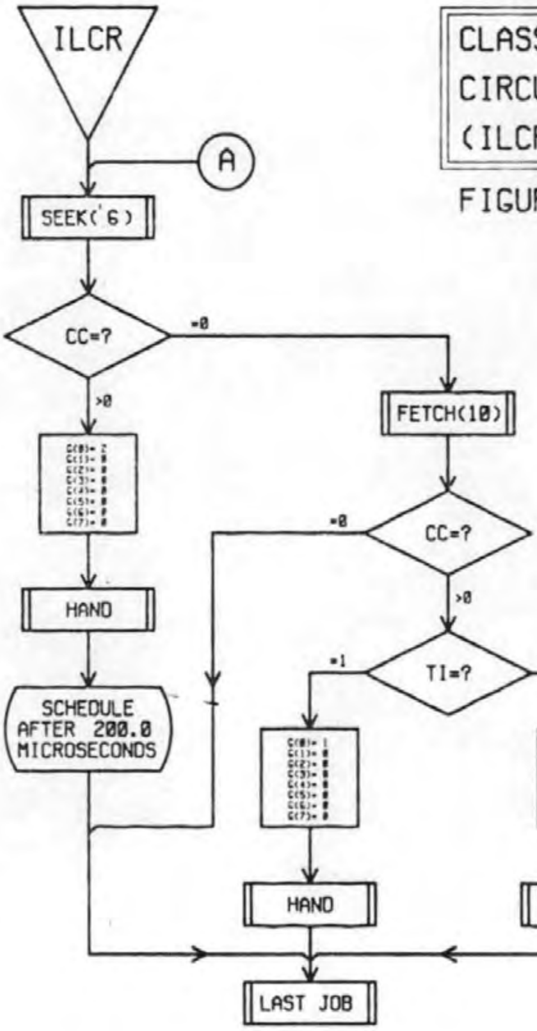
PI=21

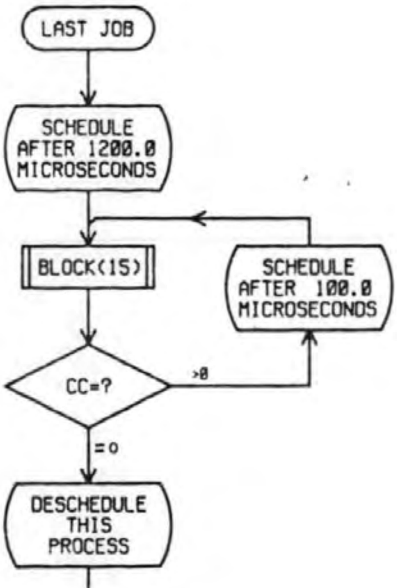
FIGURE (6.17)



CLASS INCOMING LINE
CIRCUIT ROUTINER
(ILCR) PI=19

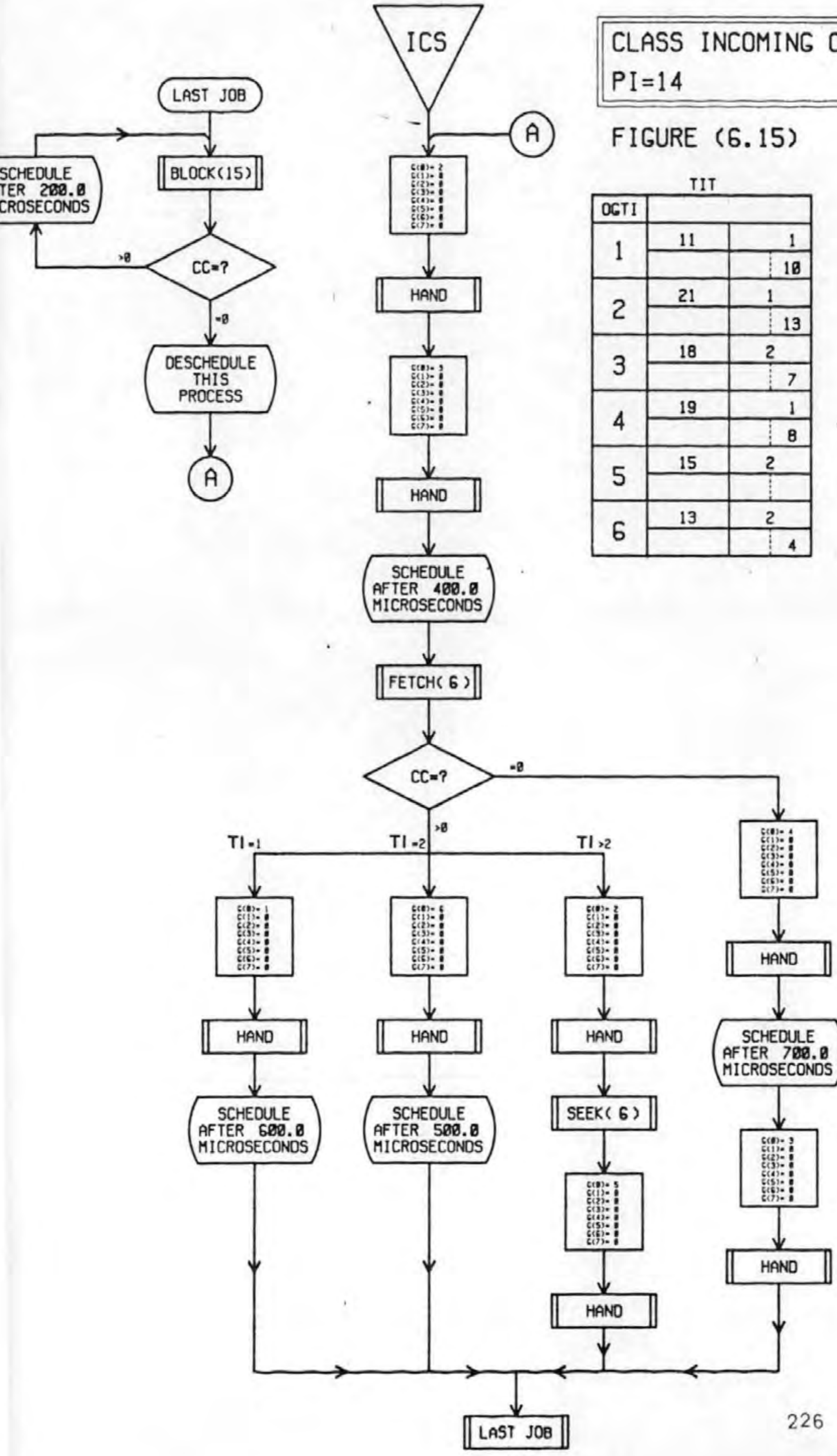
FIGURE (6.14)




LAST JOB
SCHEDULE AFTER 1200.0 MICROSECONDS
BLOCK(15)
SCHEDULE AFTER 100.0 MICROSECONDS
CC=?
>0
=0
DESCHEDULE THIS PROCESS
A

TIT

OGTI		
1	14	3
		6
2	11	2
		6
3	13	1
		10

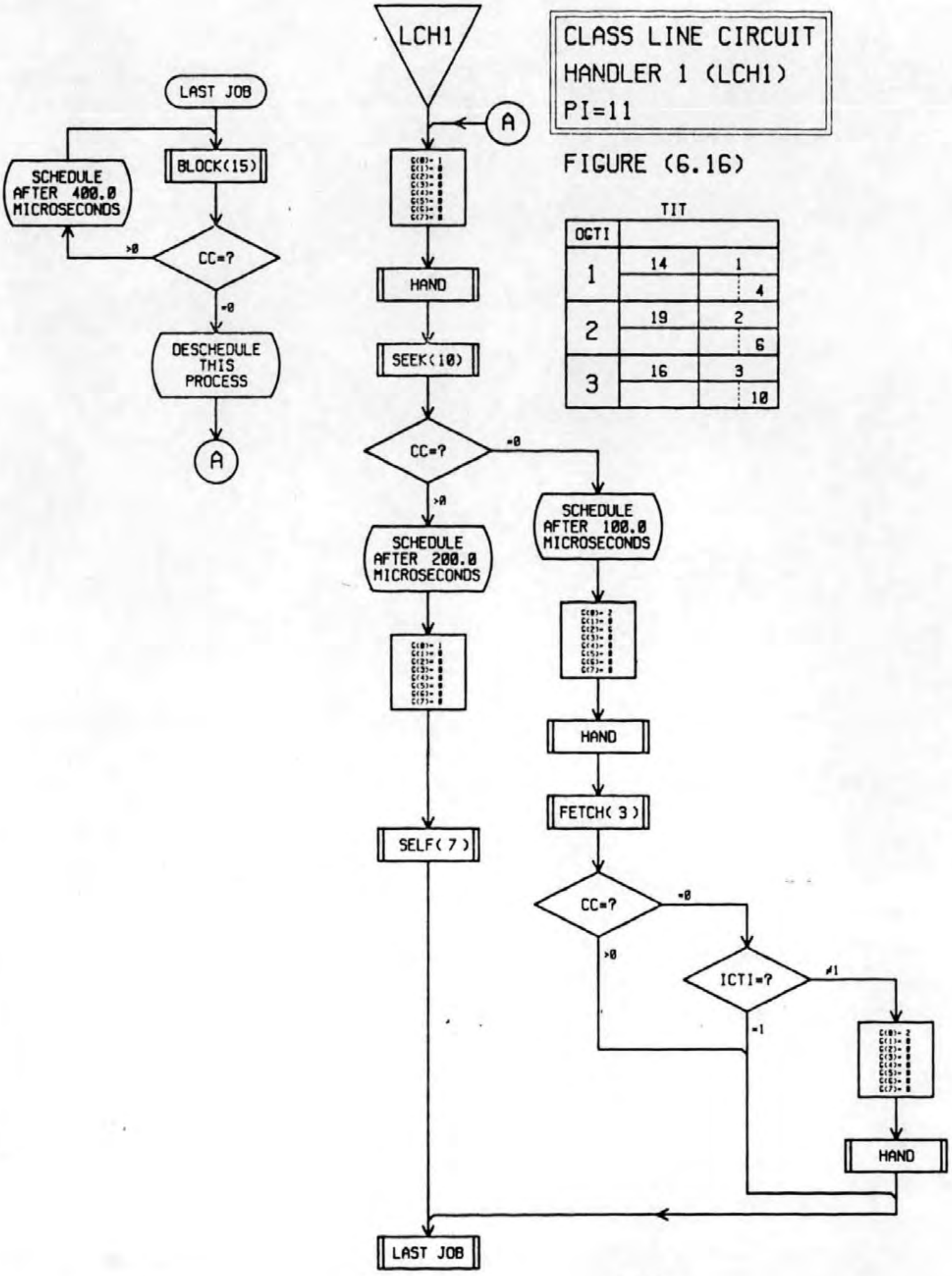


CLASS INCOMING CALL SUPERVISION
PI=14

FIGURE (6.15)

TIT

OGTI		
1	11	1
		10
2	21	1
		13
3	18	2
		7
4	19	1
		8
5	15	2
6	13	2
		4



CLASS LINE CIRCUIT
HANDLER 1 (LCH1)
PI=11

FIGURE (6.16)

TIT

OCTI		
1	14	1
		4
2	19	2
		6
3	16	3
		10

To enable RASH instances to run, INTIM (Interrupt and Timing Process) is arranged to hand them unblocking tasks when INTIM instance is first activated with the arrival of the first clock interrupt at simulated time zero. That is to say, RASH processes are considered to be periodic temporarily to enable them to start running. In reality they are set running by the software engineer typing the command RUN. This diversion from reality is insignificant and does not affect the subsequent dynamic behaviour of the simulator.

The highest priority process is that of the storage allocator (Priority = 14 i.e. the lower the priority number the higher the priority). This followed by INTIM process (Priority = 16) followed by the three RASH processes (Priority 41 - 43).

For a configuration of one CPU only, INTIM process has to wait for the storage allocator to finish any pending tasks before it can run to service the clock interrupt. The maximum delay that INTIM can experience is 4.3 msec which is the time required by the storage allocator to deal with a 'closefile' request from RASH. For a configuration of one CPU this delay is irrespective of the number of RASH instances whereas for a configuration of more than one CPU, it will depend on the number of requests pending and thus can be more than 4.3 msec. Of course INTIM does not wait for the storage allocator to finish if one of the RASH processes is running on another CPU as it will be pre-emptied by INTIM on a priority basis. The storage allocator distinguishes between an open and a close file request by examining the value of G2 of the request task. A value of 4 indicates a close-file request whereas a value of 5 an open-file request.

A value of G2 = 8 will indicate a part-file read request. Although this is allowed for in the model, the request is not issued by the RASH processes. The time required by the storage allocator to honour a part-file request would have been drawn from an empirical distribution based on data collected from the real system.

Every time a RASH process issues a request it follows that by executing a BLOCK(3) call to the process allocator to be blocked until the storage allocator response of priority 3 arrives. The requests to the storage allocator have priority 10. Following the issue of the two requests RASH process checks for the arrival of the time task from INTIM which arrives every 100 msec. Whether it arrives or not the execution counter is incremented by one to indicate that the loop is completed once more and the loop is started again. The storage allocator picks up the tasks in its input queue using a BLOCK(10) instruction. When the simulated time expires a report is outputted displaying among other things the contents of the execution counters. A number of these reports for different configurations are included in Appendix (D).

6.2.2 Discussion of Validation Experiment Results

The results of the nine runs of the validation experiment are tabulated in Table (6.1). The figures are the values of RASH processes execution counters. The columns indicate the number of CPUs in a configuration and the rows between consecutive horizontal double lines, the number of RASH processes. For a configuration of one RASH process and one, two and three CPUs, we note an increase in the number of times RASH is executed when the number of CPUs is increased from one to two. An increase from 521 to 609 is noted in the real system and from 559 to 615 in the simulator.

	REAL SYSTEM			SIMULATOR		
CPUS / RASH PROCESSES	CPU 1	CPU 2	CPU 3	CPU 1	CPU 2	CPU 3
RASH 1	521	609	529	559	615	615
RASH 1	524	485	489	560	405	405
RASH 2	0	475	482	0	405	405
RASH 1	522	355	293	556	270	270
RASH 2	0	350	292	0	270	270
RASH 3	0	315	289	0	270	270

TABLE (6.1) TIMES RASH PROCESSES EXECUTED FOR A CONFIGURATION OF 1,2 and 3 CPUs with 1,2 and 3 RASH PROCESSES

This increase is attributed to the fact that, with two CPUs available, the storage allocator will always occupy one of them, leaving the other one to be contested by INTIM and RASH processes. INTIM is activated once very 10 msec with the arrival of a clock interrupt and since there are no periodic processes for it to activate it occupies a CPU for a relatively shorter time. Hence, a RASH process is guaranteed to run (possibly with minor delays) every time it is handed a response task by the storage allocator. This is in contrast to the case of one CPU only in a configuration; here RASH is always pre-emptied on priority basis by the storage allocator and can only be run when both the storage allocator and INTIM are idle.

It is worthwhile to note that when the number of CPUs is increased to 3, with only one RASH process running in the configuration, the value of the execution counter drops for the real system whereas it remains constant for the simulator. This difference is due to some remaining randomness in the real system such as that due to store contention. The store contention effect is expected to decrease appreciably with the application of faster and higher-capacity storage modules.

When two or three RASH processes are running in a configuration of one CPU the execution counter value for RASH2 or RASH3 is zero Table (6.1). This is because as soon as RASH1 hands a request task to the storage allocator, it is pre-emptied by the latter. It is only allowed to run again when the storage allocator processes the request, hands back the response task and executes a BLOCK(10) call which is blocked as there are no further request tasks pending in the input queue, (429 - 558).

Since RASH1 is the first of the RASH processes to be selected to run on the basis of its priority, the other RASH processes will have no chance at all of being run throughout this particular configuration. We note that the request tasks from RASH have a priority of 10 whereas their responses from the storage allocator have a priority of 3. Periodic unblocking tasks from INTIM have priority of 5. There is a notable drop in the execution counters values when three RASH processes are run in a configuration of 3 CPUs instead of 2 CPUs in the real system. This drop is inexplicable in terms of the system scheduling algorithms and is only attributable to residual randomness due to factors such as the store contention. What is expected is that at least no drop should be experienced in the values of execution counters since more CPUs are available now. This expectation is more fulfilled in the simulator than in the real system.

The remainder of this section will be devoted to a statistical assessment of the credibility of "goodness of fit" of the Mark II BL simulator. The execution counters values of Table (6.1) are rearranged in a descending order of value in Table (6.2) and the table is arranged in such a way as to facilitate the calculation of different statistical parameters. The random variable X denotes the execution counter values for the real system whereas the random variable Y denotes those for the simulator.

X	Y	$x=X-\overline{X}$	$y=Y-\overline{Y}$	x^2	xy	y^2
0	0	-408.44	-386.94	166823.11	158041.77	149722.50
289	270	-119.44	-116.94	14265.89	13967.31	13674.95
292	270	-116.44	-116.94	13558.27	13616.49	13674.95
293	270	-115.44	-116.94	13326.35	13499.55	13674.95
315	270	-93.44	-116.94	8731.02	10926.87	13674.95
350	270	-58.44	-116.94	3415.23	6833.97	13674.95
355	270	-53.44	-116.94	2855.84	6249.27	13674.95
475	405	66.56	18.06	4430.23	1202.07	326.16
482	405	73.56	18.06	5411.07	1328.49	326.16
485	405	76.56	18.06	5861.43	1382.67	326.16
489	405	80.56	18.06	6489.91	1454.91	326.16
521	550	112.56	163.06	12669.73	18354.03	26588.51
522	556	113.56	169.06	12895.85	19198.45	28581.21
524	560	115.56	173.06	13354.07	19998.81	29949.71
529	615	120.56	228.06	14534.68	27494.91	52011.23
609	615	200.56	228.06	40224.24	45739.71	52011.23
$\Sigma X=6535$	$\Sigma Y=6191$			$\Sigma x^2 =$	$\Sigma xy =$	$\Sigma y^2 =$
$\overline{X}=408.44$	$\overline{Y}=386.94$			338846.92	359288.57	422218.73

TABLE (6.2) : Values for Calculation of Regression of Y on X and Correlation Coefficient

With reference to Table (6.2).

For the real system:

$$\text{The mean} = \overline{X} = \frac{1}{N} \sum_{i=1}^{N} Xi \qquad = \quad 408.44$$

$$\text{The variance} = \sigma_x^{\,2} = \frac{1}{N-1} \quad \Sigma(Xi-\overline{X})^2 \qquad = \quad 22589.79$$

$$\text{The standard deviation} = \sigma_x = 150.30$$

The corresponding values for the simulator are as follows:

$$\text{The mean} = \overline{Y} = \frac{1}{N} \sum_{i=}^{N} Yi \qquad = \quad 386.94$$

$$\text{The variance} = \sigma_y^{\,2} = \frac{1}{N-1} \quad \Sigma(Yi-\overline{Y})^2 \qquad = \quad 28147.92$$

$$\text{The standard deviation} = \sigma_y = 167.77$$

The two means of the real system $\overline{X}$ and simulator $\overline{Y}$ differ by

$$\frac{|\overline{X} - \overline{Y}|}{\overline{X}} \times 100 = \frac{408.44 - 386.94}{408.44} = 5.26\%$$

Expressed in a different way, the two means agree to within

$$\frac{|\overline{X} - \overline{Y}|}{\sigma_x} = 0.143\ \sigma_x$$

which is considered to be a very good agreement. (HORN 72).

We will now test the output of both systems for positive correlation, determine the correlation coefficient value and derive the least square linear regression equation of Y on X.

The correlation coefficient r is defined as App. (E).

$$r = \pm \frac{\Sigma xy}{\sqrt{(\Sigma x^2)(\Sigma y^2)}} \qquad \text{.........(6.3)}$$

where the quantities xy, x^2 and y^2 are as described in Table (6.2).

Substituting for xy, x^2 and y^2

$$r = \pm \frac{359288.57}{\sqrt{338846.92 \times 422218.73}} = +0.9498$$

The positive value is taken because y increases as x increases. The linear least square regression line of Y on X is defined by the equation

$$Y = a_o + a_1X \qquad ....(6.4)$$

where a_o and a_1 are obtained from the normal equations (SPIE 72)

$$\Sigma Y = a_oN + a_1\Sigma X \qquad ....(6.5)$$

$$\Sigma XY = a_o\Sigma X + a_1\Sigma X^2 \qquad ....(6.6)$$

When these two equations are solved, they result in a_o and a_1 as having the values

$$a_o = \frac{(\Sigma Y)(\Sigma X^2) - (\Sigma X)(\Sigma XY)}{N\Sigma X^2 - (\Sigma X)^2} \qquad ....(6.7)$$

$$a_1 = \frac{N\Sigma XY - (\Sigma X)(\Sigma Y)}{N\Sigma X^2 - (\Sigma X)^2} \qquad ....(6.8)$$

Using the transformation $x = X - \overline{X}$, $y = Y - \overline{Y}$ where $\overline{X}$ and $\overline{Y}$ are the mean values, equation (6.4) becomes

$$y = \left(\frac{\Sigma xy}{\Sigma x^2}\right)x \qquad ....(6.9)$$

substituting the values of Σxy and Σx^2 from table (6.2)

$$y = \frac{359288.57}{338846.92} \cdot x = 1.06\ x$$

or

$$Y - \overline{Y} = 1.06(X - \overline{X})$$

$$Y = 1.06X + \overline{Y} - 1.06\overline{X}$$

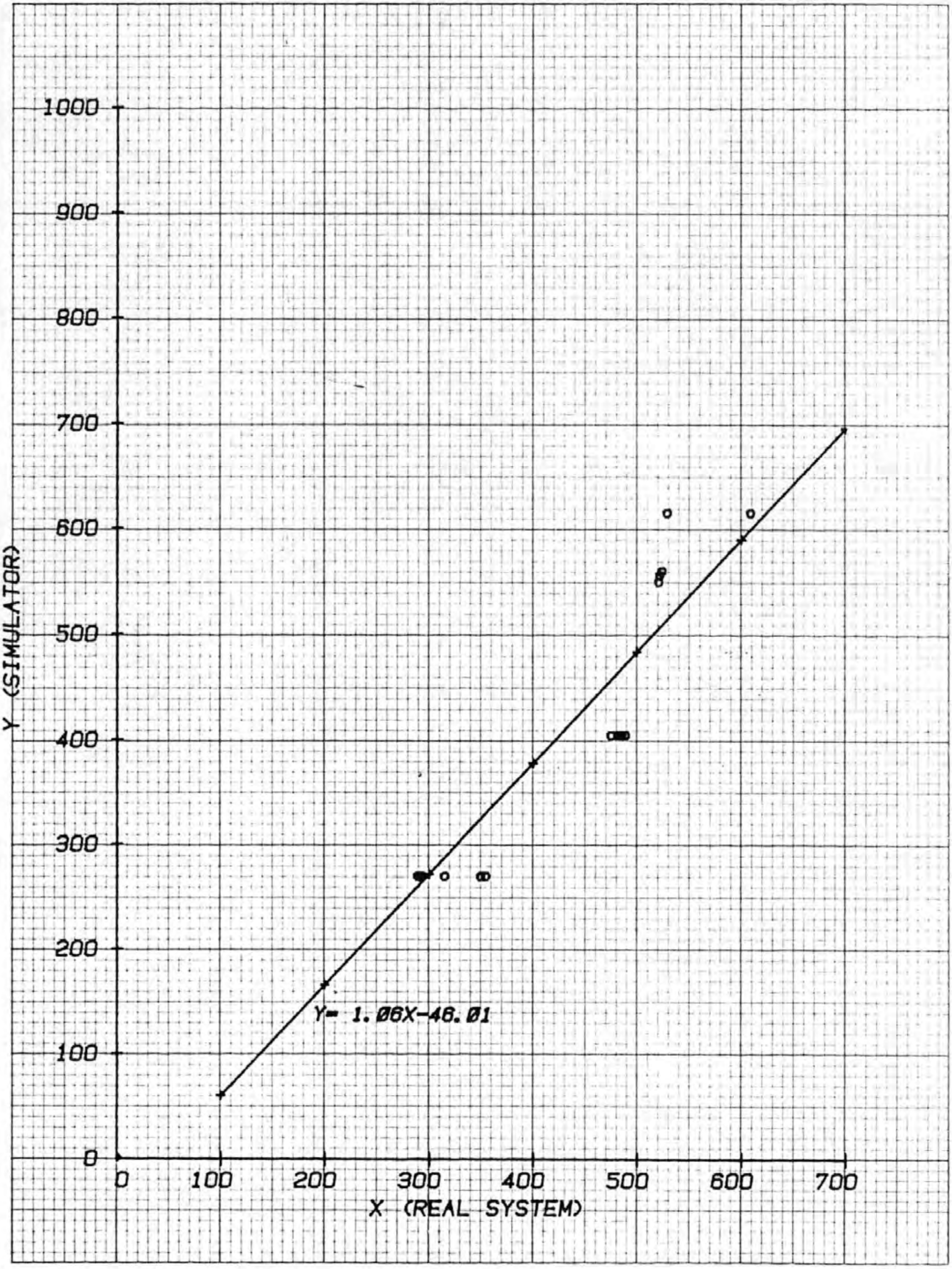
$$= 1.06X + 386.94 - 1.06X408.44$$

$$Y = 1.06X - 46.01 \qquad ....(6.10)$$

This is the least-square regression line of Y on X, which is also plotted in Graph (6.1). In the verification by regression analysis the "goodness of fit" criteria of a simulation model is that the slope of the regression line is nearly unity and the intercepts are not

GRAPH (6.1)

REGRESSION LINE OF Y (SIMULATOR) ON X (REAL SYSTEM)



significantly different from zero (COHE 61). From the high positive correlation coefficient value (+0.9498), the values of $\overline{Y}$, $\overline{X}$ which differ by $0.143\sigma_x$ only and the form of equation (6.10) where the slope is almost unity one can conclude that the credibility of the simulator or its "goodness of fit" to the real system is quite good.

The process described in this chapter seems deceptively simple. Validating a complex system such as the Mark II BL multi-processor system proved to be both a complex and time-consuming process (NOE 74). The situation is best described in Macdougal's words (MACD 74) "While the simulation model represents a reduction in size of several orders of magnitude relative to the actual system, the reduction in complexity is not equally great. The complexity of the actual system results from the interactions of its parts and the representation of these interactions is a major part of the modelling effort. There is no substitute for care in program design and implementation, and all the other prosaic aspects of the programming art"

This validation exercise, apart from building confidence in the model, enhances greatly the understanding of the system dynamic behaviour and helps to identify areas where potential performance improvements are possible as will be shown in Section 6.3. The verification and validation process is depicted in Figure (6.18).

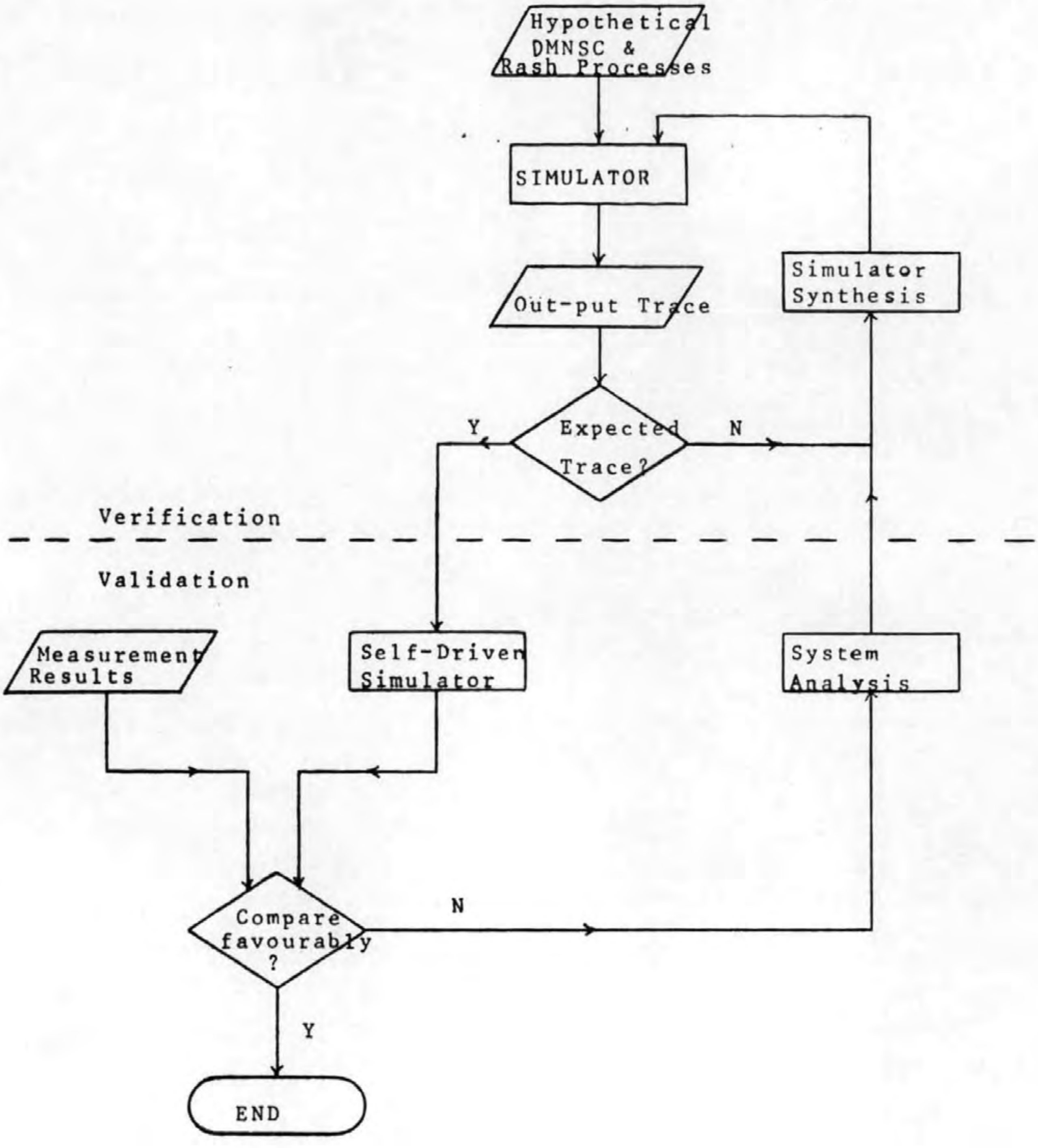


FIGURE (6.18) Verification And Validation Process

6. 3 Experiments Conducted Using the Mark II BL System Simulator

In this section two experiments will be reported. They both make use of the Mark II BL Simulation package developed and are meant to give a flavour of the type of problems that can be tackled and to demonstrate the flexibility and ease of use of the package in the System X sub-systems design - optimisation process.

6.3.1 Investigation of a New Call FBLOCK(P) to the Process Allocator

With reference to Section (4.3.2.1), FBLOCK(P) call to the process allocator is equivalent to:

```
FETCH(15)
JNZ(C+Z)
BLOCK(P)
```

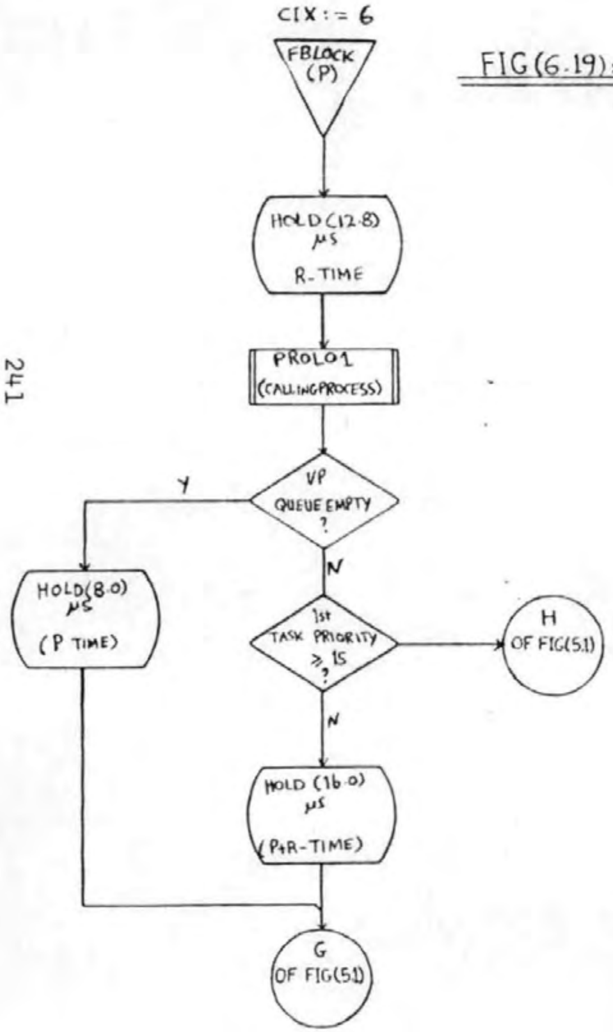
It is proposed to be used largely in conjunction with periodic processes which normally pick up all the tasks in their input queues when periodically activated by INTIM process (4.3.2.2), and then block awaiting the next activation. Thus, when the process allocator is issued an FBLOCK(P) call, it services a FETCH(15) call routine first. If a task is found in the calling process's input queue, the condition codes are set positive and the process allocator exits back to the process. On the other hand, if no task is found in the input queue, the process allocator sets the process blocked with a priority level P and schedules on that CPU. P is the priority of INTIM's periodic unblocking task.

The saving in the process allocator is that if FBLOCK(P) call is negative that is no task is found in the input queue, the process allocator does not exit back to the process which will then execute a BLOCK(P) call, but rather sets it blocked and schedules the CPU without the need to engage the process's PROLO Lockout and inspect the input queue (Figure (6.19)).

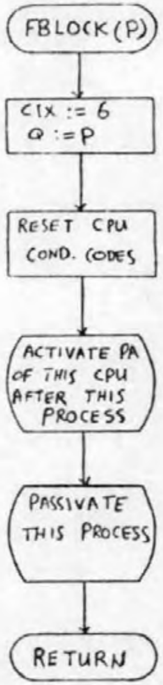
A simulation experiment is designed to investigate this new call to the process allocator and quantify the reduction in its overhead. The periodic processes NOFBLCK (Figure (6.20)) and WITHFBLCK (Figure (6.21) are an adaptation of the RASH processes. They are used in this experiment with a periodicity of 10 msec.

When activated by INTIM, the test processes check their input queues. If a task is present, they will either issue an 'Open-File' or 'Close-File' request to the storage allocator. This will be determined by the value of G(2) of the task in the input queue, which is usually a response task from the storage allocator to a previous request.

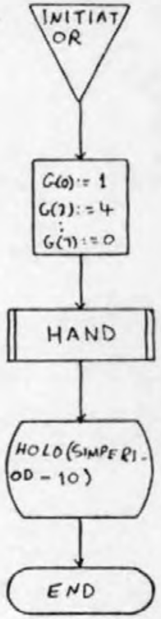
To enable the storage allocator to service up to five periodic test processes during the 10 msec clock period, the storage allocator times are reduced from 4050.0 and 4300.0 microseconds to 650.0 and 700.0 microseconds respectively. Despite that, it is noted from the output trace that the storage allocator over ran the clock period in servicing five test processes in a configuration of 2 CPUs. This particular case resulted in a rather over-optimistic figure of 23.88% in the process allocator overhead reduction. Therefore, this particular overshoot value is discarded in the final analysis. However, if a larger number of periodic processes are required this can be attained by increasing the periodicity of the test processes and/or reducing the storage allocator service times.

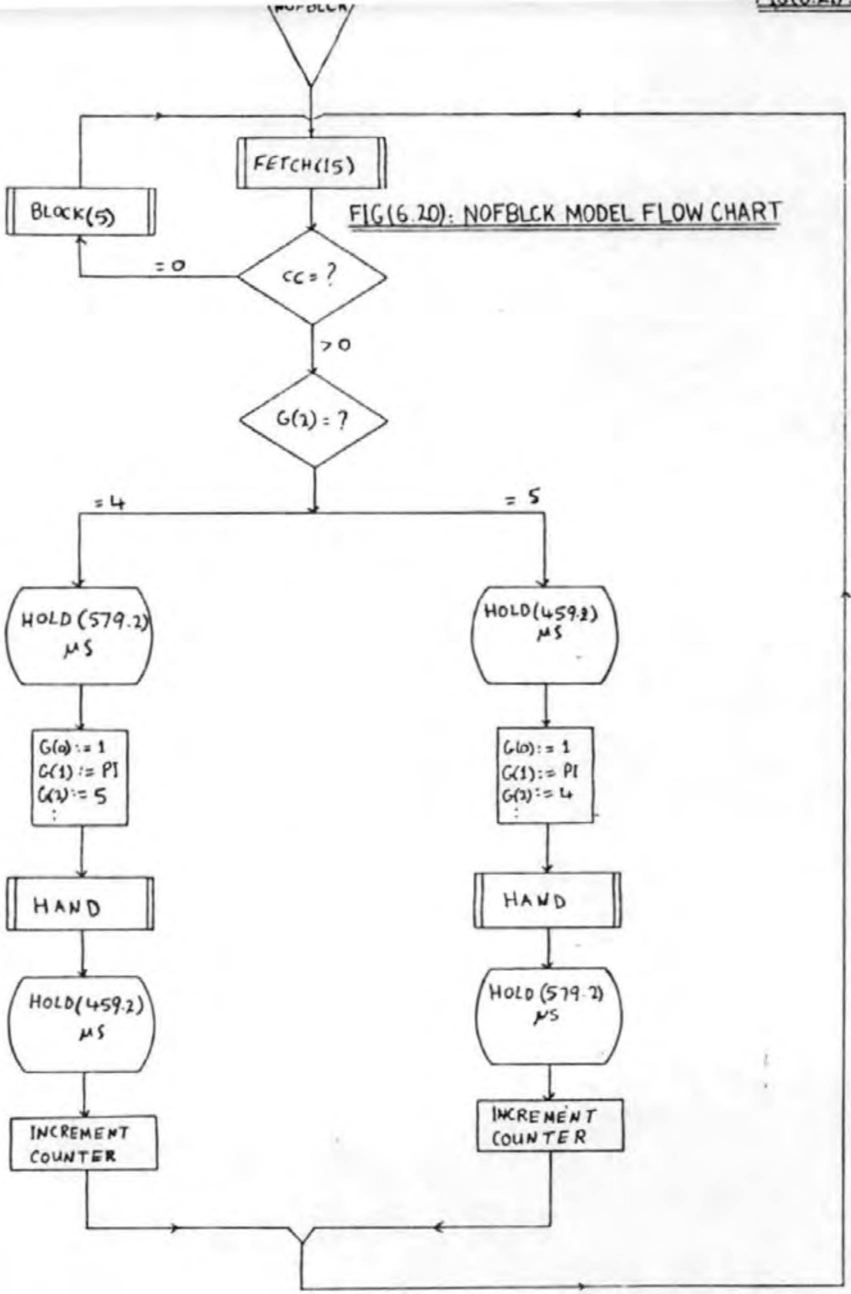


FIG(6.19): FBLOCK(P) CALL AND FBLOCK(P) PROCEDURE

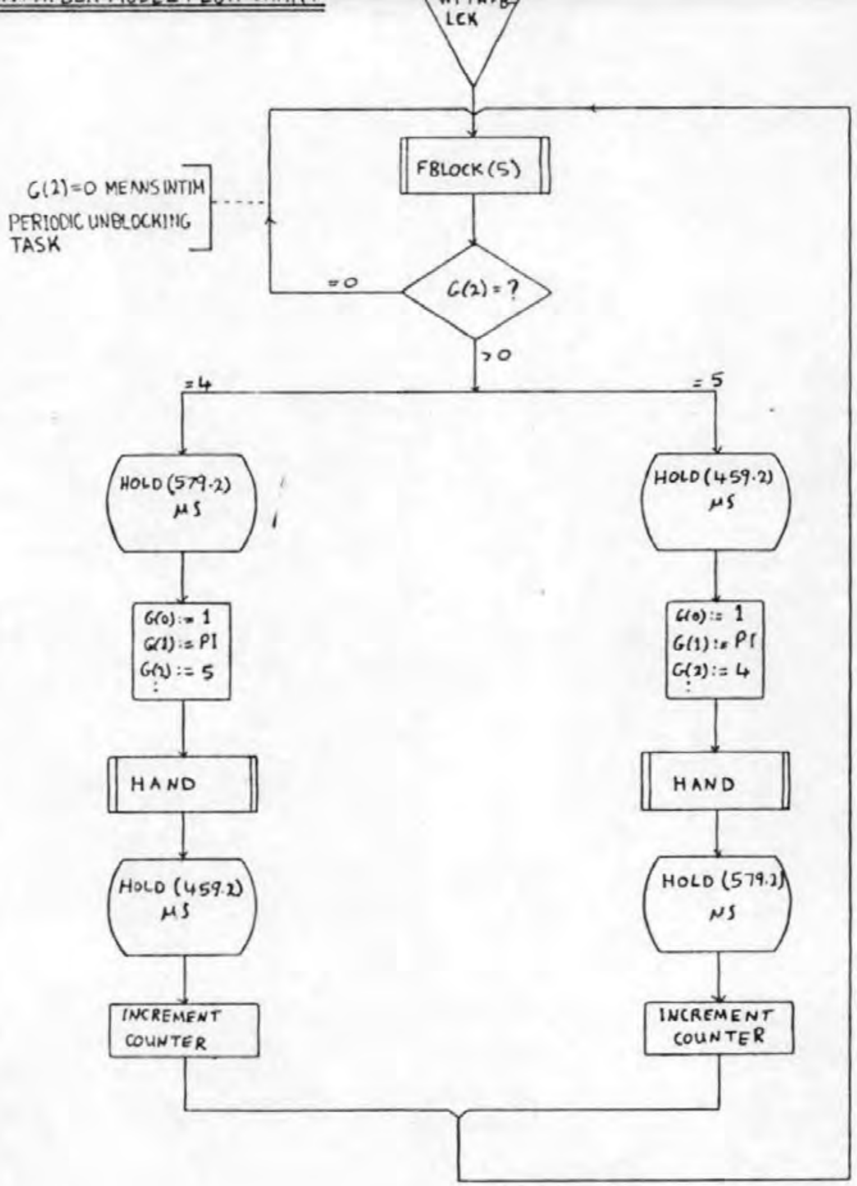


FIG(6.24): INITIATOR PROCESS MODEL





FIG(6.20): NOFBLCK MODEL FLOW CHART



To initialise the experiment, the INITIATOR process is incorporated in the model. Its function is to hand 'Close-File' response tasks to the test process, on behalf of the storage allocator (Figure (6.22)), at the start of a run. Thus when they are activated by INTIM and have checked their input queues, response tasks will be found to which they will respond by sending further request tasks to the storage allocator. The contents of a request task is defined as follows:

G(0) = 1
G(1) = PI - the process index of a test process
G(2) = 4 or 5 - for open or close file request
G(3) - G(7) = not used.

Two sets of experiments are carried. In the first set up to five instances of NOFBLCK process are used. When this process is activated it executes a FETCH(15) as in Figure (6.20). If a task is available in the input queue, it hands a close or open file request to the storage allocator according to the value of the general register G(2). A counter is then incremented and the above mentioned sequence of actions repeated for positive FETCH(15) calls. If, however, the fetch call is negative, the process instance calls to be blocked until the arrival of the next INTIM's periodic unblocking task of priority level 5.

In the second series of experiments, up to five instances of the periodic test process WITHFBLCK (Figure 6.21)) are used. This process is similar to NOEBLCK process, the only difference is that it uses the new call, FBLOCK(P). In both process types, since the storage allocator services all the test processes once during a clock period, the counter of each process instance is incremented once during that period.

The process allocator overhead portions due to FETCH(15), JNZ(C+Z) BLOCK(P) and the equivalent FBLOCK(P) are tabulated in Table (6.3) and Table (6.4) respectively. The percentage reduction in this overhead portion, calculated on the basis of the average overhead per CPU in a configuration is tabulated in Table (6.5) and plotted in Graph (6.2).

The reduction in the overhead due to the use of FBLOCK(P) call is in the range 19% to 23.59%. All the experiments are run for a simulated time of 7.2 secs, thus all process instances counters read 720. If we assume that the process allocator overhead for FETCH(15), JNZ(C+Z), BLOCK(P) is X μsecs and that due to FBLOCK(P) is X - δμsecs, then

$$\% \text{ reduction in overhead} = \frac{100\delta}{X}$$

There appears to bea variable element in the value of δ according to the number of events of finding and not finding a task in the input queue when executing FBLOCK(P) call. The variation in the number of events is dependent on the configuration of a particular run. This fact explains the variation in the overhead reduction between 19% and 23.59% Nevertheless, the reduction in the process allocator overhead by using FBLOCK(P) call is significant. Further, the incorporation of this call into the microprogrammed process allocator does not entail a major modification.

6.3.2 Tuning of Interrupts Handling in Mark II BL System

During the model verification phase, when the output trace of the system dynamic behaviour is closely examined (Chapter 6), it is felt that one of the areas where the system performance improvement is possible is the interrupts handling mechanism of the real-time operating system. For example the Suspended Queue

PROCESSES CPUS	NOFBLCK 1	NOFBLCK 2	NOFBLCK 3	NOFBLCK 4	NOFBLCK 5
1	161505.12	281770.75	589584.69	336687.94	602675.12
2	161666.62	383558.69	370890.37	1033042.00	601201.75
1	161486.94	210251.19	315323.62	315652.81	527104.50
2	0.00	210355.69	294672.19	315929.12	527062.56
3	161666.62	210143.94	315377.69	623333.00	527477.62
1	0.00	0.00	232481.06	314629.37	623927.25
2	161486.94	211922.25	232500.06	314641.31	321115.12
3.	0.00	211963.50	232069.75	314606.50	321119.56
4	161666.62	211963.62	232071.37	314607.69	321113.00
1	0.00	0.00	0.00	298655.81	
2	0.00	0.00	233235.56	297376.37	
3	161486.94	211922.25	233201.62	307523.69	
4.	0.00	211963.50	232069.75	300642.81	
5	161666.62	211963.62	232071.37	1633.86	

TABLE (6.3) PROCESS ALLOCATOR OVERHEAD (μs) DUE TO FETCH(15) JNZ(C+Z), BLOCK(P)

PROCESSES CPUS	WITHFBLCK 1	WITHFBLCK 2	WITHFBLCK 3	WITHFBLCK 4	WITHFBLCK 5
1	130898.56	301845.50	452905.19	261743.87	458820.12
2	130839.94	227131.00	309479.87	845151.31	457532.62
1	130898.56	165027.19	247841.19	247556.94	422438.94
2	0.00	165131.62	233274.19	248001.37	423194.37
3	130839.94	164919.87	245558.69	502040.50	422817.12
1	0.00	0.00	182369.44	165598.94	316083.69
2	130898.56	165071.12	182372.75	165230.06	316086.94
3	0.00	165069.81	182273.81	500110.19	316093.81
4	130839.94	165069.94	182274.75	165229.25	316085.94
1	0.00	0.00	0.00	2351.38	232780.56
2	0.00	0.00	182339.81	230294.25	231164.87
3	130898.56	165071.12	182343.12	229696.37	229984.06
4	0.00	165069.81	182273.81	230304.25	230816.37
5	130839.94	165069.94	182274.75	228772.00	231802.00

TABLE (6.4) : PROCESS ALLOCATOR OVERHEAD (μs) DUE TO FBLOCK(P)

PROCESS TYPE & NO. OF CPUS No. OF PROCESSES	NK 2 CPUS	WL 2CPUS	OVERHEAD REDUCTION (%)	NK 3 CPUS	WL 3 CPUS	OVERHEAD REDUCTION (%)	NK 4 CPUS	WL 4 CPUS	OVERHEAD REDUCTION (%)	NK 5 CPUS	WL 5 CPUS	OVERHEAD REDUCTION (%)
1	161585.87	130869.25	% 19.00	107717.85	87246.12	% 19.00	80788.39	65434.59	% 19.00	64630.71	52347.67	% 19.00
2	332664.72	264488.15	% 20.50	210250.27	165026.22	% 21.50	158962.34	123802.71	% 22.12	127169.87	101735.89	% 20.00
3	480237.53	381192.53	% 20.64	308457.83	242224.69	% 21.47	232280.54	182322.62	% 21.51	186115.66	144133.66	% 22.56
4	684864.95	553447.55	% 19.19	418304.96	332532.93	% 20.51	314621.20	249042.11	% 20.84	241166.48	184283.71	% 23.59
5	601938.40	458176.37	% 23.88	527214.86	422816.80	% 19.80	399068.85	316087.50	% 20.79			

Table (6.5) : AVERAGE PROCESS ALLOCATORS OVERHEAD AND THEIR PERCENTAGE REDUCTIONS

KEY

NK : NOFBLCK PROCESS

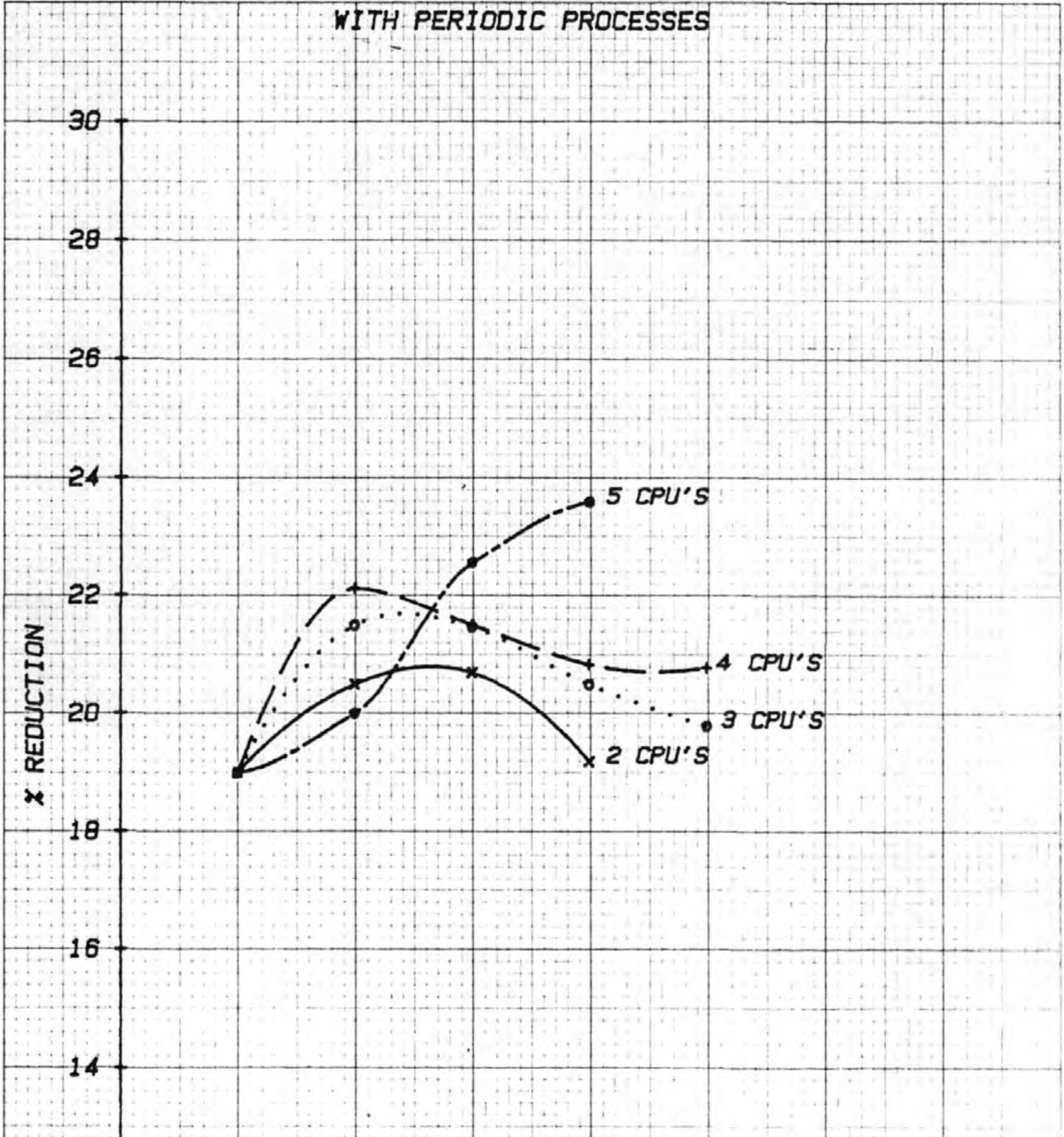
WL : WITHFBLCK PROCESS

The General Electric Company Limited. Hirst Research Centre

Report No.	Date	Drawn by	Drawing No. T	A4

GRAPH (6.2)

% REDUCTION IN USING FBLOCK (P) WITH PERIODIC PROCESSES



Interrupt is found to be triggered unnecessarily in certain cases, with the subsequent delay to the pre-empted process. Examining the runs traces closely, it is found that the interrupt triplicates reference to the CPU running the lowest priority process (LCPU) is not updated as early as it ought to be in the scheduling sequence of a CPU. Let us explain this by a simple example. Suppose that LCPU (Sec. 5.4.9) is CPU1 and its current running process is process X. Suppose a CPU schedule 'CPUSCHED' (Sec. 5.4.10) executed for LCPU and another process Y is selected from the suspended state map to run on CPU1. If, however, a third process, process Z, say, is suspended in another CPU, CPU2, before updating LCPU and the further process Z is of a priority higher than process X but lower than process Y, then the suspended queue interrupt will be triggered by CPU2 and steered to CPU1. It is steered to CPU1 because the LCPU reference is not yet updated. Thus process Y will be pre-empted from CPU1 when the suspended queue interrupt is serviced but will be selected again for CPU1 because it is of a higher priority then process Z. This unnecessary nest and de-nest of a process and the subsequent delay is avoidable if LCPU reference is updated as soon as a process is selcted to run on a CPU. This is confirmed by trial runs on the model where LCPU is updated immediately after SUSPLO2 in CPUSCHED (Sec. 5.4.10).

Another aspect of tuning the interrupt handling mechanism of the Mark II BL multiprocessor system is to interrupt a running process only if it is a background process. The modification to effect this change is a minor one both in the real system and the simulator. This is an example of the ease in incorporating design modifications and extensions in the simulator brought about by its process based nature and its multi-level hierarchical structure as explained in

Chapter 1. In the system scheduling procedure SYSCHED of section 5.4.10 instead of:

```
  IF(I < INTRIP. LCPU. CURP. PI AND I NE 0 AND I NE 127)
```

we include:

```
  IF(I < INTRIP. LCPU. CURP. PI AND INTRIP. LCPU. CURP. PI > 116)
```

where 116 is the process index of the highest proprity background process.

The two proposed modifications to the interrupt handling mechanism, namely, updating LCPU as soon as a new process is selected and interrupting LCPU only if it is running a background process are incorporated in the Mark II BL simulator. Two sets of three experiments are carried out using RASH processes as in Chapter 6.. In the first set, for a configuration of 2 CPUs and 2 background processes, 3 runs of the simulator are made with 3, 4 and 5 RASH instances respectively and using the existing interrupt handling mechanism. The same experiments are then repeated with the two proposed modifications incorporated in the interrupt handling mechanism. The statistics of interest are summarised in Table (6.6). It is evident that the modifications resulted in an increase in the throughput and an appreciable decrease in the percentage of total process allocators interrupts to total process allocators calls. The increase in the throughput is due to the fact that the process allocator overhead in servicing the interrupts is now being used to do useful processing. However, the advantages of not interrupting a running process unless it is a background one are offset at times of high traffic, when low priority processes are still running while high priority ones are waiting, with a possible loss of traffic. Nevertheless the proposal of updating LCPU as soon as possible is easy to implement in the microprogrammed process allocator without any side effects. The proposal has been

ITEM	MODIFIED INTERRUPT HANDLING			EXISTING INTERRUPT HANDLING		
	3 RASHES	4 RASHES	5 RASHES	3 RASHES	4 RASHES	5 RASHES
No. OF CPUS	2	2	2	2	2	2
Throughput (RASH CYCLES)	359	359	353	246	215	225
Total PA Calls	3316	3332	3275	2300	2050	2126
Total PA Interrupts	54	60	55	435	457	422
Background Processes	2	2	2	2	2	2
% $\frac{\text{Interrupts}}{\text{Calls}}$	1.63	1.8	1.68	18.9	22.27	19.84

TABLE (6.6) SUMMARY OF STATISTICS FOR THE INTERRUPT HANDLING MECHANISM MODIFICATIONS welcomed by the software design engineer in charge of the process allocator and may well be implemented in the process allocator for the System X Release 2.

CHAPTER 7

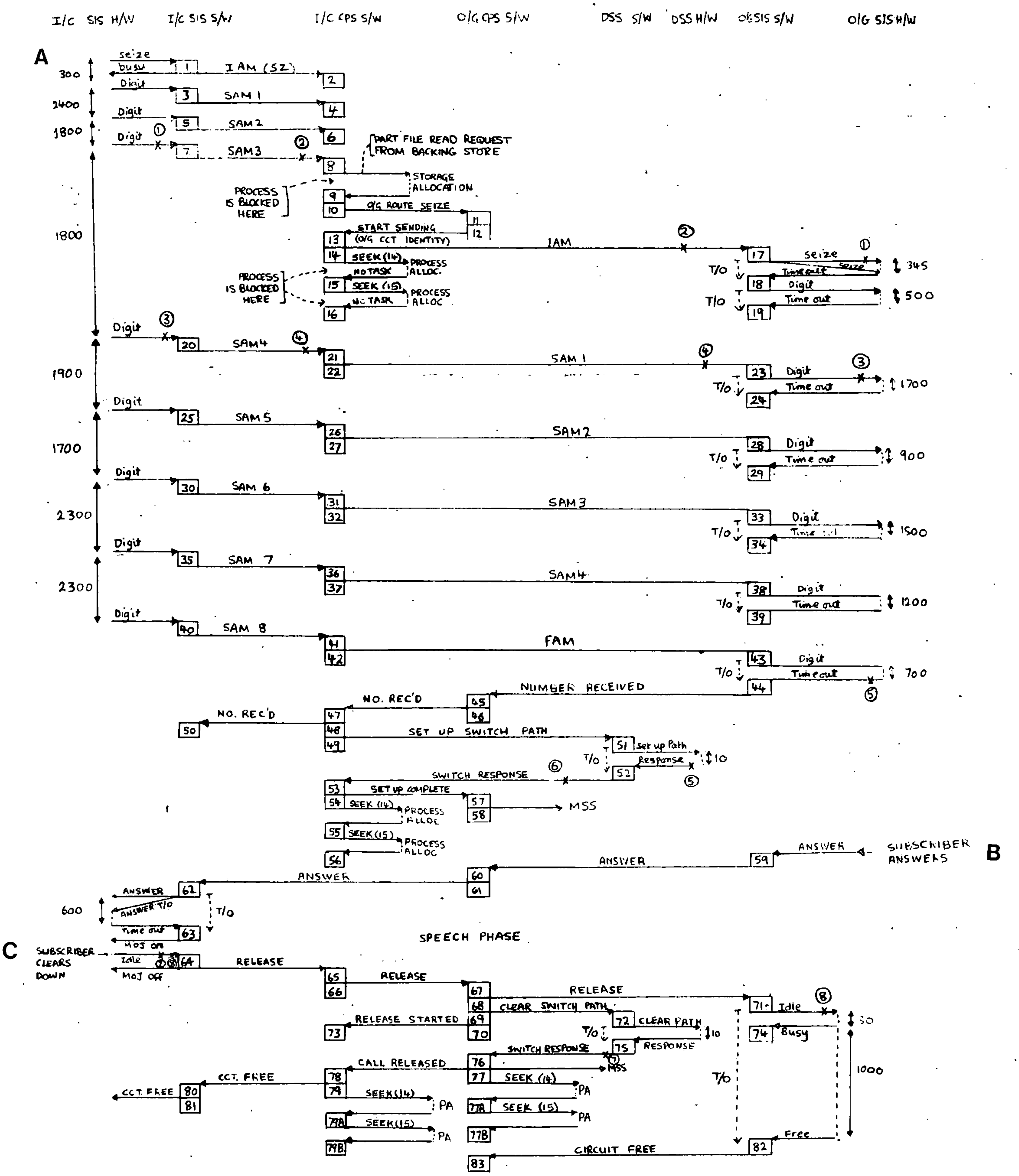
7.1 THE DIGITAL MAIN NETWORK SWITCHING CENTRE (DMNSC)

7.1.1 Introduction

The Digital Main Network Switching Centre is a digital trunk exchange, a member of the System X family of exchanges. It operates in a mixed analogue and digital environment and comes in a wide range of sizes, with the large trunk exchange anticipated to have a termination capacity of 85,000, a switch capcity of 20,000 erlangs and a processing capacity of 500,000 busy-hour call attempts (TIPP 79). A schematic block diagram of the DMNSC is shown in Figure (3.9). The exchange is assembled from a common set of System X sub-systems. The sub-systems are either hardware or software sub-systems, with most hardware sub-systems having their own software handlers which are stored and run on the processor sub-system. Examples of such sub-systems are SIS (Signal Interworking Subsystem), DSS (Digital Switching Subsystem) and MTS (Message Transmission Subsystem). The functions of these and other sub-systems have been explained in (3.6.3). The status of the DMNSC in the network hierarchy, as a trunk exchange or a regional switching centre serving a number of trunk exchanges, can be changed by software modifications, provided the junction and trunk routes can cope with the new pattern of traffic flow. All existing channel associated signalling systems are provided over analogue and digital transmission systems plus digital common channel signalling. The DMNSC is controlled by Mark II BL multiprocessor systems with the number of CPUs being dependent on the traffic carried, with pre-processing employing microprocessors. The software sub-systems have secured message interfaces. The DMNSC operates as a mutually synchronised centre at any assigned level in the synchronised network. The overall management of the DMNSC is exercised by the Local Administration Centre together with a local man/machine control point using a man/machine high level language. The exchange itself is constructed from general purpose sub-systems allowing a wide range of applications in the existing and proposed network configurations, and allowing an evolutionary updating of the network. Another important advantage of this modular construction is that it allows new technology to be introduced as it becomes viable and proven.

Apart from the design and development of the processor sub-system, GEC is also responsbile for the design and development of the Call Processing Sub-system (CPS). This is a totally software sub-system responsible for all telephony functions. This sub-system is central to the efficient functioning of the System X family of exchanges. Figure (7.1) is a typical message sequence chart of an incoming (junction) decadic to an outgoing (junction) decadic call through the DMNSC. The message sequence chart indicates the most important sub-systems with which CPS communicates. In Figure (7.1) there are eight pairs of instances in a call processing sequence marked by crosses and numbered 1 to 8. The delay distributions between each two events in a pair is indicative of the performance of the DMNSC and consequently whether that performance meets the British Post Office requirements or not.

The objectives of simulating the DMNSC are twofold. On the one hand, it is essential to evaluate the performance of the DMNSC by quantifying the delays between particular instances in the call processing sequence as mentioned above. On the other hand, the simulation of the DMNSC is required to tune the call processing sub-system. In the following paragraphs the DMNSC is described from the point of view of the CPS, as it is our main concern here.



ENCIRCLED NUMBERS INDICATE START AND FINISH OF RESPONSE TIMES IN WHICH THERE IS PARTICULAR INTEREST:

1. O/G CIRCUIT SELECTION TIME (H/W TO H/W)
2. " " " (TASK SPACE TO TASK SPACE)
3. SIGNAL TRANSFER TIME (H/W TO H/W)
4. " " " (TASK SPACE TO TASK SPACE)
5. SWITCHTHROUGH TIME (H/W)
6. " " (S/W)
7. RELEASE TIME
8. CLEAR FORWARD TRANSFER TIME

FIG(7.1): MESSAGE SEQUENCE CHART OF AN INCOMING (JUNCTION) DECADIC TO AN OUTGOING (JUNCTION)

The Call Processing Sub-system (CPS) is responsbile for supervising all the control functions relating to the passage of telephone calls through the exchange in which it resides. It is a purely software sub-system and does not interface directly with the hardware necessary for setting up the call. Other sub-systems, namely, SIS MTS and DSS have this responsbility, CPS interworking with them by means of task transfers when required. A modular approach has been adopted in the design of CPS to enable as much commonality as possible to be achieved between the various variants of CPS for the various applications such as the DMNSC and the local exchange. It was also necessary to establish clearly the true essentials of the call processing function, with the objective that the basic structure of the sub-system should be as general purpose as possible. Any differences in the call processing functions associated with the various signalling systems and circuit types should not be reflected into the foundations of the CPS design. This has resulted in the adoption of the strategy that, in order to set up a call, the call processing entities in each exchange, through which the call passes, require to communicate with each other in the best possible way. For calls originating in the new network, the medium by which this is achieved is the MTS sub-system which is effectively transparent to CPS. On the other hand, the SIS sub-system is the medium for calls originating in the old network with in-band signalling over the actual speech path. The interfaces between CPS and MTS and between CPS and SIS are designed to be as similar as possible using a common repertoire of call protocol messages with any pecularities of the existing signalling systems being effectively hidden in SIS.

CPS communicates through a defined interface with the DSS sub-system to physically set up and clear down the switch paths between the incoming and outgoing circuits. It also provides features for dealing with line circuit maintenance and the on-line updating of certain data areas under the control of MCS. It also sends statistical data related to calls it processes to the MSS sub-system and co-operates with the OCS sub-system in controlling overload situations. The above mentioned interfaces of CPS with other sub-systems will be elaborated upon later. First it is worthwhile to consider the sequence of events in processing a call in the DMNSC in more detail.

7.1.2 Basic Sequence of Events in Call Processing in the DMNSC

With reference to the message sequence chart of Figure (7.1), the basic sequence of events is as follows:-
The start of a call is indicated by CPS receiving an 'Initial Address Message' (IAM). This may be either a simple seizure message or a message containing a number of dialled digits. This message, as with all other messages, contains the identity of the appropriate line circuit. Further dialled digits are received in subsequent address messages from SIS. According to the line signalling type, the digits may be sent as soon as possible to this exchange, each digit being received in a separate address message, or if the interworking with the previous exchange can be more interactive as in the new network, then the digits may be sent on request by CPS at this exchange. In this latter case, several digits may be sent in one address message, this is en bloc working.

When sufficient digits have been received by CPS, it sends a part-file read request message to the storage allocator. This is a message to translate the digits received to network routing information which provides the details of up to five possible routings for the call in priority order. The details for each routing consist of the outgoing route identity plus the particulars on how and what digits should be sent on to the next exchange. It may be that a new set of digits has to be injected, and if so, these would be provided in the routing information.

An attempt is made to select a free circuit on the first choice outgoing route. If no such free circuit is available, the attempt is repeated for the remaining subsequent routes until all have been attempted. This is called Automatic Alternative Routing. The selected circuit is busied in software to prevent it being selected on subsequent calls using that route. The circuit selection algorithm uses two modes of working, a cyclical selection followed by a sequential selection (DENT 78A). On a successful outgoing circuit selection, the CPS instance in charge of the outgoing route sends back the message "Start Sending" with the outgoing circuit identity to the incoming CPS instance. Whether or not the incoming CPS instance is the same as the outgoing CPS instance depends on the outgoing route identity in the routing information, that is whether the incoming and outgoing routes are the responsibility of the same CPS instance or not. A CPS instance can have up to 256 routes, each route having up to 256 bands with 16 circuits in a band (DENT 78B).

Having received the "Start Sending" message, the incoming CPS instance sends an initial address message (IAM) to SIS on the outgoing circuit, containing the appropriate digits. At this point in time, a message could be received from the next exchange via SIS conveying an unsuccessful set-up such as 'Congestion' or 'Subscriber Engaged'.

If that message does come through, CPS can either abort the current set up to the outgoing circuit and repeat on the same route, attempt to re-route on an alternative route or abort the call completely applying tone to the incoming circuit via DSS if necessary (DENT 78A).

Assuming a successful set-up as in Figure (7.1), the remaining digits received on the incoming circuit via incoming SIS are passed on in subsequent address messages to SIS on the outgoing circuit. The last digit is sent in a 'Final Address Message', FAM, to which SIS on the outgoing circuit responds back by a 'Number Receiver' message indicating that no further address messages are required in order to set-up the call. This message is sent end-to-end to the SIS on the incoming circuit side.

On receiving the 'Number Received' message the incoming CPS instance sends a 'set-up switch path' message to the DSS software handler to set-up the switch path between the incoming and outgoing circuits. When DSS responds back with a 'Switch Response' message, the incoming CPS instance sends the 'Set-up Complete' message to the outgoing CPS which will have control of the call from here onwards.

When the called subscriber answers, the outgoing SIS sends the 'Answer' message which is sent end-to-end to the incoming SIS which is also responsible for starting the call charging by sending the 'Meter Over Junction' message to the originating local exchange. A similar message is sent when the subscriber clears down. This also initiates the call clear and circuit release sequences. Usually, the exact sequence of messages depends on the order in which the calling and called subscribers clear down, and whether the particular DMNSC is in control of the release. Assuming the calling party clears first, a 'Release' message is sent end-to-end. This message is sent when a network switching node has started to release its switch connection. On receipt of this message CPS initiates the release of its switch connection. An SIS equivalent is a clear forward from an incoming SIS circuit. The outgoing CPS instructs the DSS to 'Clear Switch Path' and when DSS responds back with a 'Switch Response' message, usually within a comparatively short time, the outgoing CPS sends a 'Call Released' message to the incoming CPS instance. The 'Circuit Free' message is sent when a switch connection has successfully released and a circuit is free to accept a new call. On receipt of this message, CPS frees the circuit in software and hence makes the circuit available for the selection of a new call if and when the switch connection has been successfully released.

The messages that are passed between CPS and SIS/MTS and relate to telephone calls are collectively known as call protocol messages. One of the design objectives is to maximise the commonality between the sequencing of messages on both the new and old networks, that is between CPS and SIS for the old network and between CPS and MTS for the new one. A detailed description of this common repertoire of messages is given in (DENT 78C). Each message is described in terms of the contents of the eight general registers of Mk. II BL, that is G0-G7. The messages sent between CPS and SIS or MTS are further classified into five categories according to the role which they play in the progress of a call. The first of these categories is the Forward Address Messages and these are used to forward dialled digits and service information through the network. The second category is the Forward Set-up Messages which are involved in the set-up phase of a call but do not carry any dialled digits. In general, they tend to be associated with rather particular functions such as the 'Receiver/Sender Forward Release' message which is only employed on a call incoming on a multi-frequency (MF) circuit, in which SIS has to inform CPS that it has released the receiver/sender. The third category is the Backward Set-up Request

Messages which are requests for forward set-up information from the previous exchange in the form of forward address or set-up messages such as 'Send n Digits' message. The fourth category is the Backward Set-up messages used to indicate a condition to the previous exchange such as 'Number Received' message. The fifth category is that of 'Call Supervision Messages' and these are mainly concerned with the supervision of a call. They can be forward, backward or both forward and backward messages such as 'Answer' and 'Clear'.

7.1.3 Sequencing of Messages

The message sequence chart of Figure (7.1) is one of several different types of call sequencing through the DMNSC. Sequencing of the other types of calls are covered by similar sequence charts. A considerable amount of detail on the sequencing of the call protocol messages is presented in the system design description. (DENT 78B).

Certain factors have influenced the design of the message sequence charts. The design of the sequencing of messages in the set-up phase is largely controlled by the need to minimise post dialling delay, and to minimise the number of messages carried by MTS in the new network so as to keep the processor sub-system load and MTS resources to a minimum. The requirement to minimise the post dialling delay is particularly critical in the old network, such as in the interchange of messages between CPS and SIS, because of the slow step-by-step nature of the path set-up. Therefore, for the old network, it is important to forward the dialled digits, each in a separate message, as soon as possible. This is in contrast with the new network, where since digital switching at each exchange consists of a single operation, the minimising of post dialling delay is far easier to achieve. Thus the number of messages on MTS may be reduced by packing up the digits into a smaller number of messages. To resolve the conflict when a call is switched both through the old and new sections of the network en route, while the network is evolving, the set-up protocols are divided into incoming and outgoing set-up protocols which are allocated to the circuit types. The flexibility inherent in the set-up protocols on the new network enables the problems of interworking old and new network signalling systems on the same telephone call to be overcome. Here the identity of the outgoing set-up protocol influences the mode of working of the incoming set-up protocol that is to effectively tune the flexibility built in the incoming set-up protocol to the call advantage. For example if the call is incoming on a new network, the outgoing set-up protocol must be established whether new or old before deciding to request the digits en bloc or individually from the incoming side.

For a symmetrical call supervision and clear sequence, the concepts of calling, called, first and last party clear, and of circuit selection at the calling, called or both ends of a circuit are introduced. For example on receipt of a 'Clear' message, the action taken will depend on the 'clear program' stored at the receiving switching node. The clear programp is loosely defined as the action to be taken by CPS on receipt of the 'Clear' message. The'clear program' would have been set-up as a result of path of entry information, routing information or service instructions received (DENT 78A). The messages that appear in Figure (7.1) are by no means comprehensive. Figure (7.1) covers only the messages interchanged in one typical type of call. For example one such message that does not appear in Figure (7.1) is the "re-answer' message. This message is sent in either direction when a network terminal re-establishes its hold condition before the switch path started to release, that is it cancels out a previous 'Clear' message. It is acknowledged end-to-end on the new network.

7.1.4 Circuit Selection

There are three basic modes of circuit selection of outgoing calls which are the responsibility of CPS. These are: single-way working, in which a circuit is available for selection at one end only (forward circuit selection); both-way working, where the selection is possible at both ends and backward circuit selection, in which it is the responsbility of the incoming exchange to select the circuit. CPS retains information per circuit on the mode of selection to be applied for each circuit.

To make a circuit available for selection on the incoming side the concept of pseudo circuits is introduced (DENT 78A). Pseudo circuits are provided on the route and the circuit is made available for selection here. On the arrival of an incoming seizure message on a pseudo circuit, all the circuits available for outgoing calls will be available for selection for incoming calls. Pseudo circuits are treated in the same fashion as all other circuits from which they are distinguished by having a different 'line circuit type'. They are chosen by the circuit selection algorithm only when all other circuits available for selection are busy. If a pseudo circuit is selected for an outgoing call, the switch connection will be inhibited until the next exchange selects a real circuit in place of the pseudo one.This is indicated by a 'Change Circuit' backward message to communicate the identity of the real circuit to this exchange. This exchange then checks the validity of the selection by ensuring that the circuit selected is a real free one, in which case a message 'Circuit Changed' will be sent on the pseudo circuit.

Otherwise, if it is another pseudo circuit, then it can be assumed that no real circuits are available and the 'Release' message is sent forward on the pseudo circuit.

7.1.5 The Virtual Circuit Concept

A prerequisiteto the incorporation of alternative routing in a network is the ability to prevent traffic from overflowing from a particular route to other routes when the route planned limits are exceeded. These limits are determined by the maximum traffic expected to be offered to the route and the number of circuits the route is expected to have. The most complex routing situation is where a call may be offered to up to five routes in turn, the primary, three intermediate routes and a final route. The simplest solution and that which is being adopted in CPS is to incorporate a simple cut-off of overflow traffic if the planned limits of a route are exceeded.

When the planned limits of a route are determined, a margin of capacity is allowed for limited failures or surges of traffic within the network. Now, consider a route of p circuits, let the planned traffic offered, after all alternative routes have been tried, have a grade of service g. This same grade of service may be obtained for a fully provided route with p+q circuits. Therefore, the route of p circuits, with alternative routing, has virtually p+q circuits. Use is made of this concept in setting the limit of cut-off overflow traffic. Thus once p+q calls have been offered to that route and are connected to that route and alternative routes, the planned limit will have been reached and further calls on that route will meet congestion irrespective of whether there are free alternative paths available. q is therefore, a measure of the capacity of the network to carry the overflow traffic.

Another reason for excessive overflow is the loss of circuits in a route. For small routes, the number of virtual circuits is decremented for each circuit lost. For large routes 'stepped integer adjustment' is implemented. Below a 'step threshold x' each circuit lost is subtracted from the virtual circuit allocation. At the step threshold value, the virtual circuits are decremented by the step value n plus the normal single circuit decrement. Above the step threshold an equal number of any circuits lost will be subtracted from the virtual circuits allocation. The above relationships may be expressed as follows:

Let RA = no. of circuits allocated with O/G access
RI = " " " in service " " "
VA = " " virtual circuits allocated
VI = " " " " available.
x = step threshold
n = step value.

a) when $RA-RI < x$, then $VI = VA - (RA-RI)$

$$(VI - RI) = (VA-RA)$$

b) when $RA-RI \geq x$, then $VI = (VA-RA) - n$

$$(VI-RI) = (VA-RA) - n$$

(VI-RI) is the allowed overflow from a route partially in service and (VA-RA) is the allowed overflow from a route fully in service.

7.1.6 Communication Between CPS and Other Sub-systems

DSS is the means by which CPS carries out physically any switch operation. Information between the two sub-systems is interchanged by task transfers; a request task from CPS to DSS, followed by a response task in the reverse direction. Five different requests can be issued by CPS. These are:

i) A request to allocate a path between two terminations A and B.

ii) A request to change a reserved path between A and B.

iii) A request to clear a path between A and B.

iv) A request to clear all paths connected to A

v) A request to trace paths connected to A.

Requests (iv) and (v) are only employed after a CPS rollback (OWEN 73) that is a CPS re-initialisation after a hardware or software fault. The current DMNSC design does not make use of request (ii) as the probability of blocking in DSS is thought to be extremely low. When a trace request task is issued by CPS, DSS responds with a task containing the identity of a termination to which A is connected plus an indication of whether more paths exist. The trace/response sequence is repeated until a response task indicates that no more connections to A exist. When setting up and clearing paths for a call, CPS specifies the identities of the two terminations in the task request.

CPS also employs DSS for the connection of tones such as busy, number unobtainable and recorded announcements. The request task from CPS to DSS contains the identity of the tone circuit of the appropriate type. DSS always connects the tone circuit in the backward direction that is to the calling party. This fact facilitates the connection of a tone circuit simultaneously to a number of speech circuits. The tone circuits are cleared using the usual 'Clear Path' request.

The 'Trace Path' message is issued by CPS to DSS in an attempt to establish calls in progress at the time of rollback together with the identity of other calls involved. This request is issued whenever any of the following messages are received by CPS after a rollback:

a message that starts a call such as 'IAM' and 'Seizure', a message that ends a call such as 'Release' and ' Circuit Free' or a message that changes the maintenance state of a circuit such as 'Maintenance Busy' and 'Free' from the Maintenance Control Sub-system. At the end of the first call which occurs on every circuit affected by the CPS rollback, a 'Clear All' request is sent to DSS instead of 'Clear Path' to ensure that all possible connections to the circuit are indeed cleared out before the circuit is returned to the free state.

A software sub-system with which CPS communicates is the Management Statistics Sub-system (MSS). CPS has the responsibility to provide this sub-system with the necessary information upon which traffic and related measurements can be based. The basic philosophy associated with the provision of call processing statistical information is that CPS should not keep any data of a statistical nature to itself, but rather provide this information in its own format to MSS. MSS then processes that information and incorporates it into the required measurements (DENT 78A,B,C).

The maintenance and update aspects of DMNSC are undertaken by the Maintenance Control Sub-system (MCS) in co-operation with CPS. A comprehensive range of resources such as circuits, routes and digit decodes are provided which can be updated by the maintenance staff using the online update interface with MCS through an interactive user-oriented language. Hardware and software faults generated by CPS are passed to MCS which pass them in the proper format to the maintenance staff.

Overload control in the DMNSC is the responsbility of the Overload Control Sub-system (OCS). This sub-system interrogates target processes periodically for information such as the amount of work rejected and accepted in the last monitoring period and the current workload limit. The current workload limit for each process is set by OCS and represents the maximum number of simultaneous calls in the set-up phase. When this limit is exceeded, a process simply rejects any new calls by reducing its workload limit to zero and sending a 'Congestion' message if backward signalling allows or by applying a congestion tone via the digital switch. When the workload condition disappears, OCS instructs the process to raise its workload limit to a reasonable level which is roughly seven-eight of the workload for the process at the time of the overload (DENT 78A, B, C.)

7 .2 THE DESIGN OF THE DMNSC SIMULATION MODEL

7.2.1 INTRODUCTION

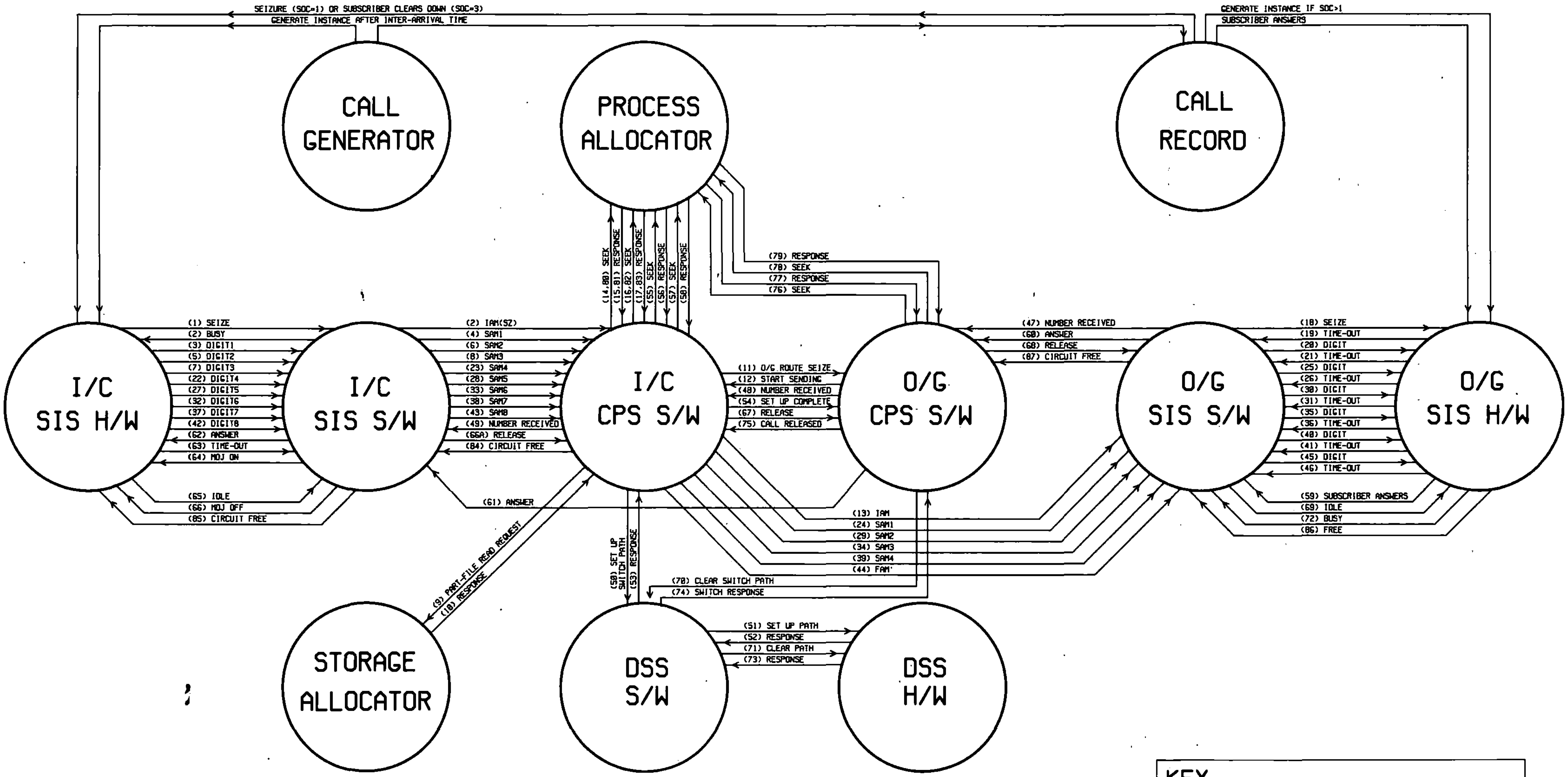
The reasons for simulating the DMNSC, namely, to tune the CPS and assess the overall performance of the DMNSC are considered in greater detail in Section 7.1.1. In the following sections we will be concerned with the design of the DMNSC simulation model based on the message sequence chart of Figure (7. 1). The square boxes in Figure (7.1) represent processing activities by the software sub-systems of the DMNSC as a result of a message or task arrival. The numbers in the boxes indicate the order in time in which messages concerning a call are processed. A message processing time is calculated by counting the number of instructions in the piece of code corresponding to a particular box and multiplying that by the average execution time per instruction. This information is supplied by the design team. A relevant term which has been cioned recently in this respect is a 'mission' (BEAR 79).

A mission is a process instance run initiated by a message arrival (after a delay if no CPU is available). Before or at the end of a mission a process instance may send tasks or messages to other process instances.

The columnsof boxes in Figure (7.1) represent the application software processes of interest in the DMNSC. These are the incoming SIS software, the incoming CPS software, the outgoing CPS software, the digital switching sub-system and the outgoing SIS software. The left and right boundaries of Figure (7.1) represent the hardware associated with the incoming and outgoing SIS software with which it communicates by passing hardware messages.

The message sequence chart of Figure (7.1) is transformed to Figure (7.2) which shows the components of the DMNSC simulation. The hardware and software entities of the DMNSC are translated in a one-to-one fashion to their corresponding model processes indicated by the big circles. The messages interchanged for each call through the DMNSC are indicated with their appropriate sequencing numbers. The Mark II BL system Model which controls the DMNSC model is transparent to the individual simulation processes as is the case with the real system.

The DMNSC model is built for an exchange of 1060 incoming and outgoing circuits and trial runs are made with a traffic of 0.6 Erlang per circuit. DSS and SIS are periodic processes activated once every 10 msec by INTIM. CPS is an aperiodic process activated by the arrival of any task of an appropriate priority (Section (4.3)). The process index of DSS is 25 while the first instance of SIS process has a process index of 26 and the nth instance a process index of 26 + (n-1). Likewise, the first CPS instance has a process index of 49 and the nth instance a process index of 49 + (n-1).



KEY

SIS=SIGNALLING INTERWORKING SUBSYSTEM
CPS=CALL PROCESSING SUBSYSTEM
DSS=DIGITAL SWITCHING SUBSYSTEM
MSS=MANAGEMENT STATISTICS SUBSYSTEM
IAM=INITIAL ADDRESS MESSAGE
SAM=SUBSEQUENT ADDRESS MESSAGE
FAM=FINAL ADDRESS MESSAGE
SOC=STATE OF CALL

COMPONENTS OF THE DMNSC MODEL AND SIGNALS INTERCHANGED PER CALL
FIGURE (7.2)

As mentioned before in Chapters 4 and 5, a process index reflects its priority, that is the lower the index, the higher is the priority. It is worthwhile mentioning here that the term instance is used in the same sense as replication or replicate, the former being the jargon of the simulation analysts world-while the latter lies in the domain of SPC software engineering. The two terms will be used interchangeably. The strategy adopted when starting or stopping the time measurements of any one of the eight response times of interest of Figure (7.1) is as follows: the starting and stopping of a particular time measurement is always performed before sending a message. For example, the time measurement of the 'Outgoing Circuit Selection Time (Task Space to Task Space)' of Figure (7.1) is stopped before sending the 'IAM' message to the appropriate SIS process instance. This strategy is adopted because the delay in sending the message to its destination through the operating system may be appreciable enough to distort the time measurement.

The number of SIS and CPS replicates in a DMNSC depends on the size of the exchange and the number of circuits allocated per replicate. In the DMNSC, the circuits are allocated to the replicate in a staggered fashion as in Figure (7.3). Hence there are 2 SIS instances or replicates and 4 CPS instances in this particular model. The model is highly parameterised with parameters such as the total number of circuits in an exchange, the number of CPUs in the controlling multiprocessor system and parameters of various distributions are passed as input data to the model in the initialisation stage. We will now consider, in more detail, the design of the individual simulation processes in the DMNSC model.

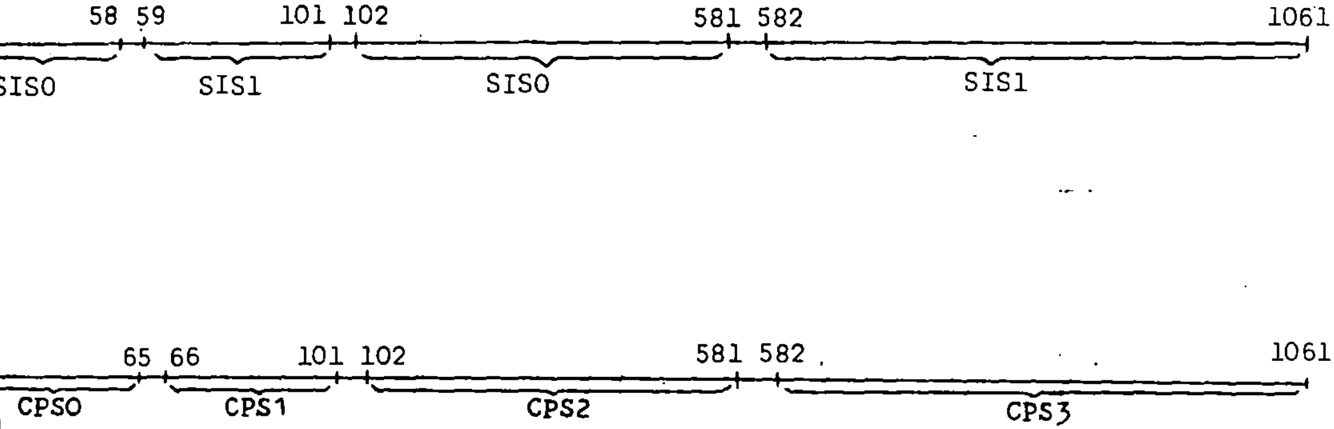


FIGURE (7.3) : CIRCUIT ASSIGNMENT TO SIS AND CPS REPLICATIONS

7.2.2 The SIS Simulation (2337 - 2702)

Although SIS is depicted as two separate software processes that is incoming SIS and outgoing SIS in Figure (7.1), in reality it is one software sub-system and indeed it is a requirement of the software design engineers that the model reflects this fact. Figure (7.4) is the SIS simulation process model flow chart. The process task index table is designed as shown in Table (7.1). The table depicts the software processes with which SIS is allowed to communicate the incoming task indices and the tasks priorities.

Since the SIS process communicates with its hardware a message table is defined as in Table (7.2). The table has twenty entries, each entry representing the nature of the hardware signal, and the associated incoming and outgoing circuits identities.

A common interface specification is adopted for all the software processes in the model. This interface consists of the general registers contents of the CPU on which an instance is running. The contents of the general registers in the DMNSC simulation are defined as follows:

G(0)	:	The outgoing task index
G(1)	:	The outgoing message identity
G(2)	:	not used
G(3)	:	The incoming circuit identity
G(4)	:	The outgoing circuit identity
G(5)-G(7)	:	not used

Each SIS instance has a table indicating the incoming messages identities corresponding to the numberical values of G(1) of the tasks received as in Table (7.3). From the above mentioned interface specification, FIG (7·$\overset{4}{4}$) SIGNAL INTERWORKING SUBSYSTEM (SIS) MODEL FLOWCHART

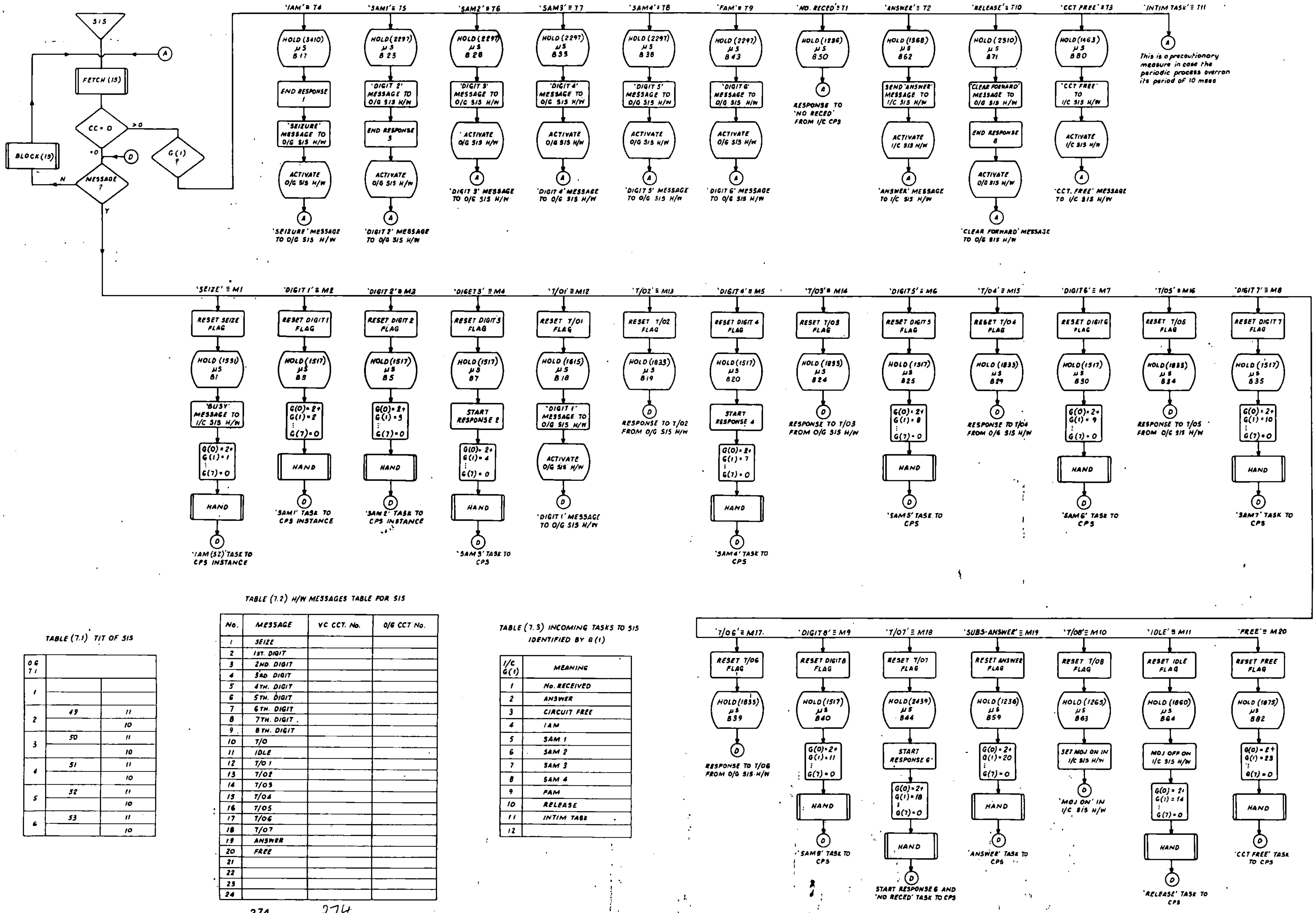


TABLE (7.1) TIT OF SIS

OG TI		
1		
2	49	11
		10
3	50	11
		10
4	51	11
		10
5	52	11
		10
6	53	11
		10

TABLE (7.2) H/W MESSAGES TABLE FOR SIS

No.	MESSAGE	VC CCT. No.	O/G CCT No.
1	SEIZE		
2	1ST. DIGIT		
3	2ND. DIGIT		
4	3RD. DIGIT		
5	4TH. DIGIT		
6	5TH. DIGIT		
7	6TH. DIGIT		
8	7TH. DIGIT .		
9	8TH. DIGIT		
10	T/O		
11	IDLE		
12	T/O 1		
13	T/02		
14	T/03		
15	T/04		
16	T/05		
17	T/06		
18	T/07		
19	ANSWER		
20	FREE		
21			
22			
23			
24			

TABLE (7.3) INCOMING TASKS TO SIS IDENTIFIED BY G(1)

I/C G(1)	MEANING
1	No. RECEIVED
2	ANSWER
3	CIRCUIT FREE
4	IAM
5	SAM 1
6	SAM 2
7	SAM 3
8	SAM 4
9	FAM
10	RELEASE
11	INTIM TASK
12	

every task interchanged between the processes is self-contained.

SIS is a periodic process, activated periodically by an unblocking task of a high priority from INTIM every 10 msec. When it is selected to run, it issues a FETCH(15)call to the process allocator to pick up the first task in its input queue if any. If a task is found which is indicated by positive condition codes, SIS checks the value of G(1) to establish the identity of the received message. Using the value of G(1), it branches off to the appropriate code representing the response of SIS to that particular message. This response always incorporates a 'Hold' or a delay and possibly the sending of a hardware message to an incoming or outgoing SIS hardware process or the stopping of the time measurement of one of the response times of interest or both. This process of fetching a task from the input queue is repeated until the input queue is exhausted which is indicated by the condition codes being zero. The process then scans its message array to see if there are any hardware messages pending. If a hardware message is detected the process branches off, using the message number, to the appropriate code servicing that particular hardware message as in Figure (7.4). A typical signal or hardware message servicing will always start by resetting the message array subscript representing the message index so that an identical hardware message may be sent immediately afterwards. The process then holds or delays itself for a simulated time equal to that required to carry out the same servicing activity in the real system. Then the SIS model may send a task to the appropriate CPS instance to communicate the arrival of the hardware message. The hardware message in this case may be a seizure that is an arrival of a new telephone call, a pulsed-out digit, a subscriber answering or clearing down or a circuit freed.

The appropriate CPS instance is indicated to the operating system by writing the equivalent outgoing task index in register G(0). This is achieved by calling the global integer procedure CPSREP (CCTNUM), Line (231), to calculate the replication number of the CPS instance. CCTNUM is either the incoming circuit or outgoing circuit number as appropriate. The value of CCTNUM is obtained from the array MESSAGE and would have been written to the appropriate array subscript when the hardware message first arrives as in Table (7.2). As the first CPS instance is indicated to the operating system by G(0) = 2, the appropriate CPS replication will thus be indicated to the operating system by the general register G(0) having the value:

$$G(0) = 2 + CPSREP(CCTNUM).$$

As we mentioned before, the other information passed in this task is a numerical indication of the nature of the outgoing task to the called CPS replication in G(1), the incoming circuit identity in G(3) and the outgoing circuit identity, if known at that instance in time, in G(4). The hardware message servicing activity may also involve sending a hardware message to the incoming or outgoing SIS hardware as well as starting or stopping a response time measurement.

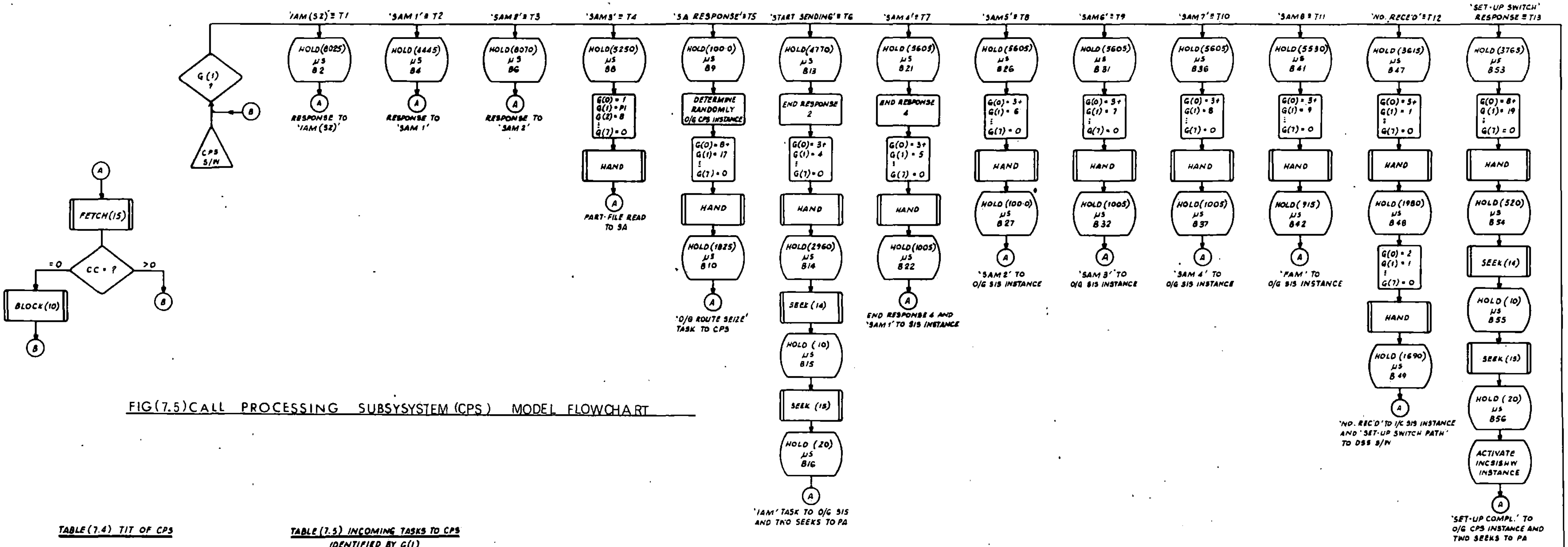
This process of checking for the arrival of hardware messages and the subsequent servicing is repeated until no more hardware messages are found. The process will have completed its servicing epoch of both software messages (tasks) and hardware messages. It will then call the process allocator to be blocked until periodically activated again 10 msec later by INTIM to go through the above mentioned routine. The hardware messages are usually exchanged between SIS instances and the incoming and outgoing SIS hardware processes. The software messages or tasks are exchanged between

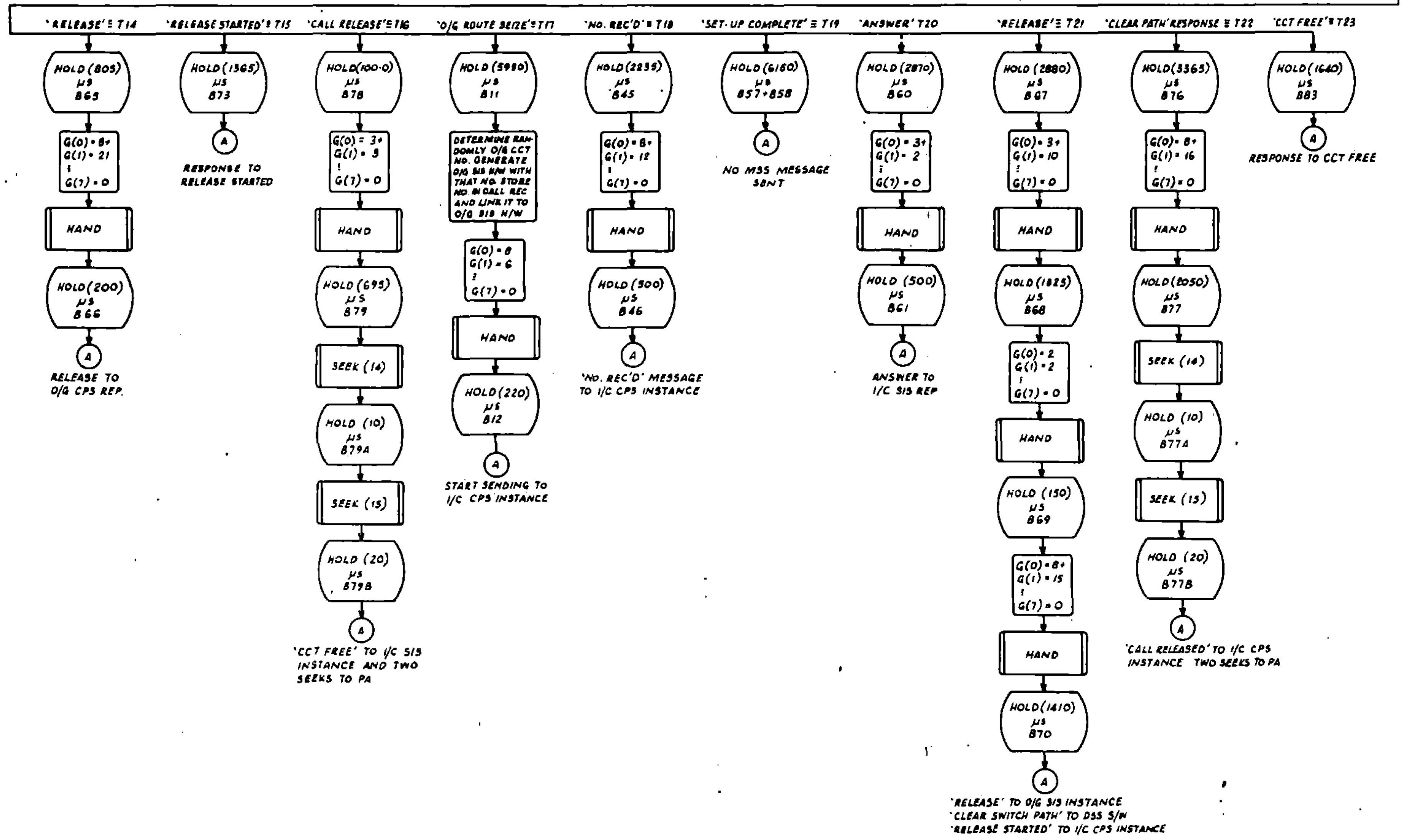
SIS and CPS instances. An SIS instance receives a total of 10 software messages or tasks and 20 hardware messages for each telephone call.

7.2.3 The CPS Simulation (2706 - 3074)

As with SIS, although the CPS process is initially modelled as two separate processes to model the incoming and outgoing sides of the DMNSC, the models are later modified and combined in a single CPS model, on the request of the DMNSC software engineers. The CPS process model flow chart is depicted in Figure (7.5). The CPS simulation task index table is shown in Table (7.4) whereas the identities of the incoming tasks are shown in Table (7.5). A total of 23 tasks per telephone call is received by a CPS instance. The CPS process, as we have mentioned, is an aperiodic process that is, it services an incoming task as soon as possible depending on the instance priority, the task priority and the availability of a CPU (Chapter 4).

When a CPS simulation instance, referred to subsequently as CPS instance, is activated and run as a result of an arrival of a task, it checks its CPU general register G(1) for the nature of the incoming task. It is interesting to note here that a CPS instance does not first execute a 'FETCH(15)' like an SIS instance. This is because while for SIS, the task contents in the CPU general registers are those of INTIM unblocking periodic task, the general registers contents when a CPS instance is activated are those of a proper task or software message from another software process model. The arrival of the task initiates a particular mission or servicing activity to which the CPS instance branches using the numerical value stored in G(1).





FIG(7.5) CALL PROCESSING SUBSYSTEM (CPS) MODEL FLOWCHART

TABLE (7.4) TIT OF CPS

OG TI		
1	14	1
		4
2	25	2
		10
3	26	8
		7
4	27	9
		7
5	28	6
		6
6	29	9
		8
·7	30	5
		8
8	49	8
		7
9	50	2
		8
10	51	4
		9
11	52	5
		7
12	53	6
		8

TABLE (7.5) INCOMING TASKS TO CPS IDENTIFIED BY G(1)

VC G(1)	MEANING
1	IAM (SZ)
2	SAM 1
3	SAM 2
4	SAM 3
5	SA RESPONSE
6	START SENDING
7	SAM 4
8	SAM 5
9	SAM 6
10	SAM 7
11	SAM 8
12	NO. REC'D
13	SET-UP SWITCH RESPONSE
14	RELEASE
15	RELEASE STARTED
16	CALL RELEASED
17	O/G ROUTE SEIZE
18	NO. REC'D
19	SET-UP COMPLETE
20	ANSWER
21	RELEASE
22	CLEAR-PATH RESPONSE
23	CCT FREE
24	
25	

A servicing activity may be a simple or a compound one. A simple servicing activity is typically represented by the holding or delaying of a process instance for the equivalent of the actual activity time such as the servicing activity resulting from the arrival of an 'IAM'or 'SAM1' task from SIS as in figure (7.5). A compound mission, on the other hand, may entail the sending of one or more tasks to other process instances, stopping the timing of a response time of interest and/or searching for particular incoming tasks using the SEEK(N) call to the process allocator such as in the mission initiated by the 'Start Sending' message arrival in Figure (7. 5).

If an outgoing task from a CPS instance is destined for another SIS instance, the identity of the called instance has got to be established, that is its process index, and the proper outgoing task index has to be written in the general register G(σ). In other words an identical procedure followed by an SIS instance and described in 7.2.2 is followed. The global procedure used here to establish the replication number of the outgoing SIS instance is the integer procedure SIS(CCTNUM), Line (224). 'CCTNUM', passed as an actual parameter, is the incoming or outgoing circuit number as appropriate. As the SIS process instance of highest priority is equivalent to an outgoing task index of 3, the outgoing task index of any other SIS instance stored in G(1) will have the value

$$G(1) = 3 + SISREP(CCTNUM)$$

Certain CPS service activities are of particular interest. The service activity initiated by the arrival of a 'SAM 3' task from an SIS process instance, sends a part-file read request task to the storage allocator, (2801 - 2807) for route translation. The details of the task are as follows:

G(0) = 1

G(1) = PI

G(2) = 8

G(3)-G(7) = not used

G(1) contains the identity or the process index of the CPS instance sending the request. This information is used by the storage allocator, later, to route back the response task to the appropriate CPS instance. The value of G(2) is an indication to the storage allocator of the service requested that is a part-file read. The storage allocator response task is usually processed fast by the CPS replicate because it is of the highest priority and hence is always inserted at the top of the input queue. The CPS instance responds to the storage allocator response task by sending the task 'Outgoing Route Seize' to the CPS instance in charge of the outgoing route (2809 - 2818). When this task is encountered in the input queue, the called CPS instance responds by selecting randomly a free outgoing circuit. It, also, creates a call record process instance for this outgoing circuit. A 'Start Sending' task is also sent to the CPS instance in charge of the incoming circuit communicating the identity of the selected free outgoing circuit.

Until the message 'Set Up Complete' is generated the CPS instance responsible for the incoming circuit has the control of the telephone call. From the 'Set Up Complete' message until the 'Call Released' message the CPS instance in charge of the outgoing circuit has the control of the call. After 'Call Released', the call as such has ended; all that remains is for the circuits to be fully released that is for the call record, the incoming SIS hardware and the outgoing SIS hardware instances, related to the released call, to terminate. These three process instances are described in section 7.2.5.

7.2.4 The DSS Simulation Model:

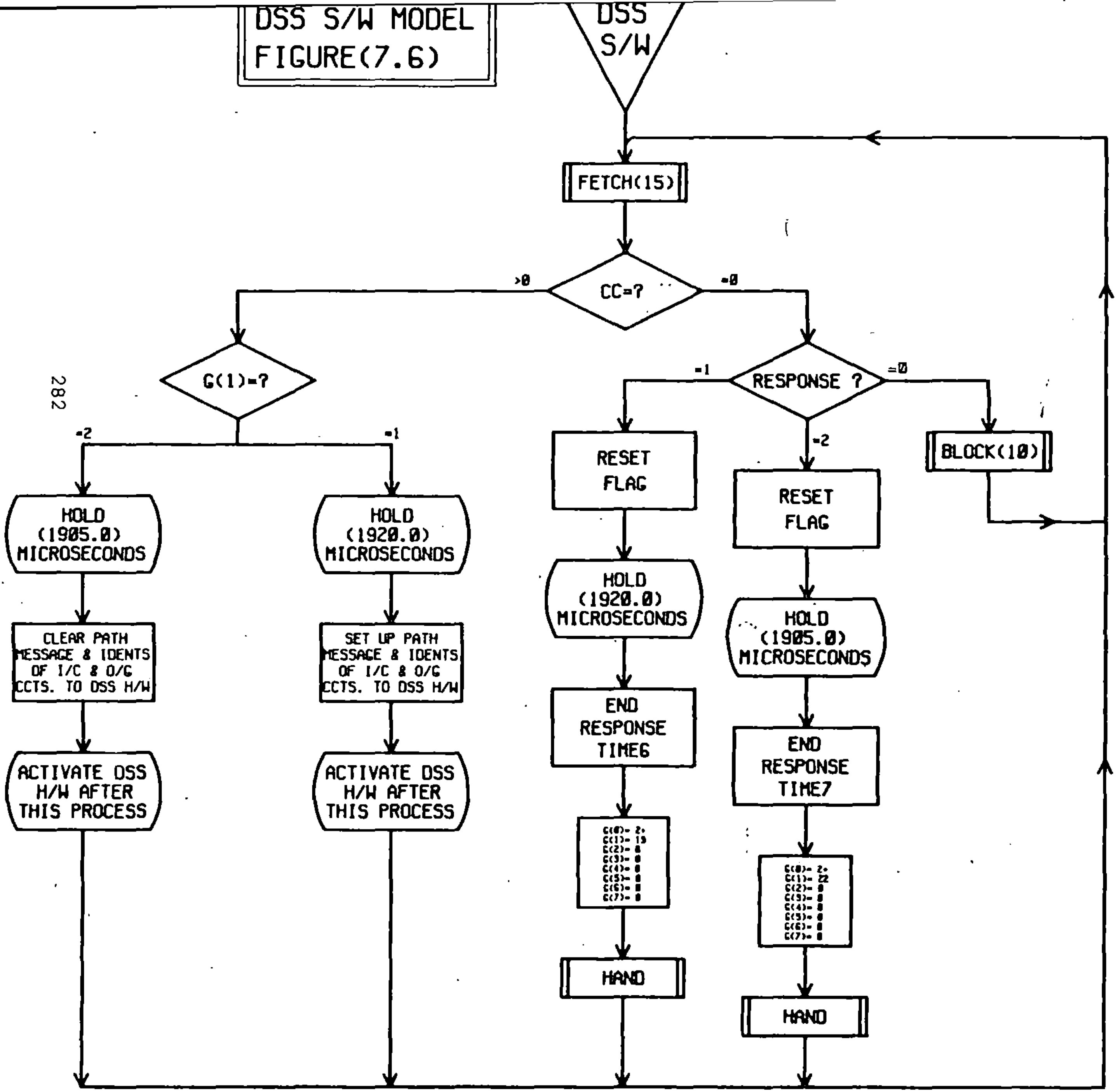
This sub-system is modelled by two simulation processes. DSSSW (3078-3161) and DSSHW (3165-3230) modelling the DSS handler and the DSS hardware respectively.

The DSSSW process is a periodic process activated once every 10 msec. Figure (7.6) depicts the process model flow chart and its task index table. When the process is activated periodically, it picks up any incoming tasks in the input queue by a FETCH(15) call to the operating system which is interpreted as a request to fetch any task of any priority.

The DSSSW process expects two types of tasks only and both from CPS process instances. The two tasks are 'Set Up Switch Path' and 'Clear Switch Path'. In each case the identities of the incoming and outgoing circuits in question are sent in G(3) and G(4) respectively, as usual.

If DSSSW detects the presence of an incoming task, it checks the value of G(1) to establish whether the request task is for setting up or clearing down a switch connection. A value of G(1) = 1 indicates the former, whereas G(1) = 2 indicates the latter. In both cases, the process holds for the appropriate activity time sends a hardware message to set up or clear the path, together with the identities of the incoming and outgoing circuits to DSSHW process and schedules DSSHW to run immediately after itself.

If no incoming task is detected in the input queue, DSSSW checks for any hardware messages from DSSHW as a response to previous messages from itself for setting or clearing switch connections. The hardware messages from DSSHW are indicated by DSSHW setting one or both of the indicators RESPONSE 1 and RESPONSE 2. If the presence of a hardware



DSS S/W MODEL
FIGURE(7.6)

TIT

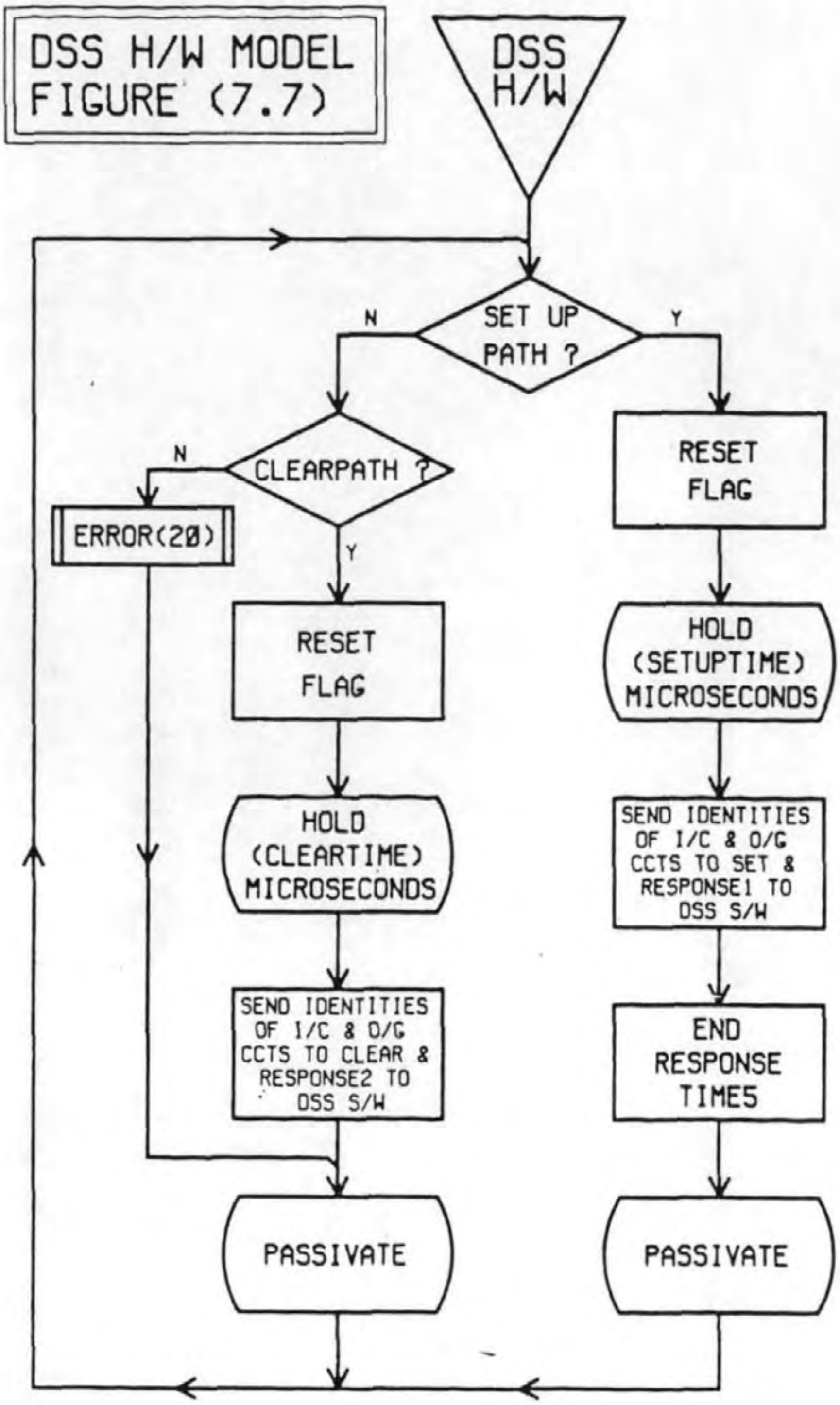
OCTI			
1			
2	49	11	
			1Ø
3	5Ø	11	
			1Ø
4	51	11	
			1Ø
5	52	11	
			1Ø

message is detected, the appropriate indicator is reset so that identical hardware messages may be sent. DSSSW then holds for the activity time, stops the appropriate time measurement and sends a response task to the appropriate CPS instance. If the task to be sent is a response task to a 'Set Up Switch Pass' request, the identity of the CPS replication is derived from the identity of the incoming circuit as this is the process instance which sends the request in the first place. Similarly, the identity of the CPS replication to which the response task to a 'Clear Switch Path' is to be sent is derived from the identity of the outgoing circuit. Both the incoming and outgoing circuits identities will have been sent by DSSHW along with the hardware message. The values will be written to INCCTS and OUTCCTS or INCCTC and OUTCCTC.

After servicing a hardware message from DSSHW, a fresh look is taken at the process input queue to see if any request tasks have arrived in the meantime. After servicing any request task which may have arrived, in the manner described, a check is made on the presence of hardware response messages from DSSHW. Any such request is also serviced in the manner described above. This sequence is repeated until all the software and hardware messages are serviced. The process then blocks awaiting the next periodic activation 10 msecs later by INTIM.

Figure (7.7) shows the flow chart of the DSSHW process (3165-3230) which simulates the digital switch. DSSHW is activated by DSSSW after sending a hardware request. The identities of the incoming and outgoing circuits are sent usually by DSSSW as part of the hardware message. When activated, DSSHW first checks for a 'Set Up Path' request. If it finds one, it first resets this

DSS H/W MODEL
FIGURE (7.7)


DSS H/W
SET UP PATH ?
N
Y
CLEARPATH ?
N
Y
ERROR(20)
RESET FLAG
RESET FLAG
HOLD (CLEARTIME) MICROSECONDS
HOLD (SETUPTIME) MICROSECONDS
SEND IDENTITIES OF I/C & O/G CCTS TO CLEAR & RESPONSE2 TO DSS S/W
SEND IDENTITIES OF I/C & O/G CCTS TO SET & RESPONSE1 TO DSS S/W
END RESPONSE TIME5
PASSIVATE
PASSIVATE

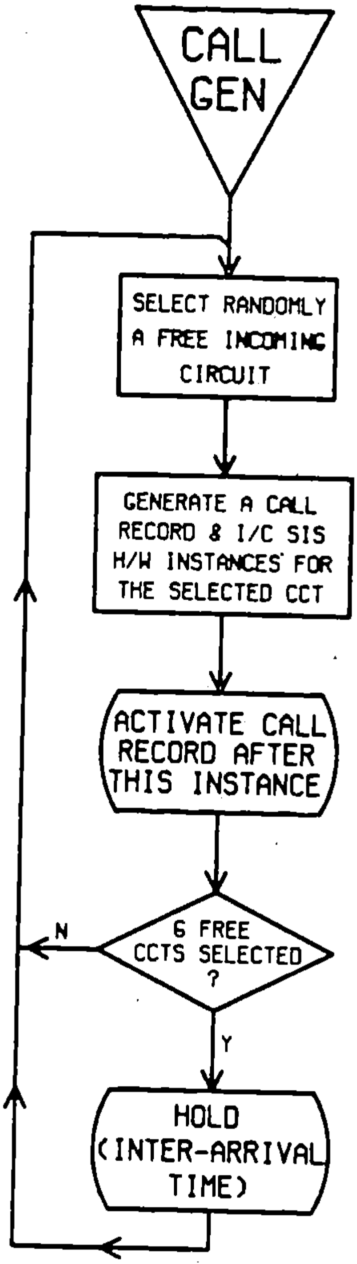
hardware message indicator, which is a boolean variable. It then holds for the set up time, derived from a uniform distribution of maximum value of 10 msec, the maximum time required by the digital switch to switch through a path between two terminations. Next it sends the identities of the terminations connected to DSSSW process and the appropriate hardware message, that is setting RESPONSE 1. It also, stops the timing measurement of the 'Switching through time (Hardware)' before passivating.

A similar action is taken if the hardware request message is to clear an existing connection. In this case the hardware response message to the DSSSW process will be indicated by setting RESPONSE 2. If when DSSHW is activated it finds no hardware message request pending, it outputs an error message.

7.2.5 Telephone Calls Generation Simulation

The strategy adopted in the generation of telephone calls for the DMNSC model aims at the maximum flexibility in the generation process and call progression through the exchange. For each call generated, three processes instances are created and exist in the model throughout the lifetime of a call. These are the call record process instance, CALLRECORD(CIRUITNUM) (2029-2114), the call incoming SIS hardware, INCSISHW(CCTNUM) (2166-2333), and the call outgoing SIS hardware, OUTGSISHW(CCTNUM) (3234-3379). These three processes instances interact and co-operate among themselves to produce the desired realistic modelling of a call progression through the DMNSC. The following is a detailed explanation of this.

The call generation is controlled centrally by a single call generator process instance, CALLGENERATOR (2118-2162) , depicted in Figure (7.8).


CALL GEN
SELECT RANDOMLY A FREE INCOMING CIRCUIT
GENERATE A CALL RECORD & I/C SIS H/W INSTANCES FOR THE SELECTED CCT
ACTIVATE CALL RECORD AFTER THIS INSTANCE
6 FREE CCTS SELECTED ?
N
Y
HOLD (INTER-ARRIVAL TIME)

CALL GENERATOR MODEL FLOWCHART
FIGURE (7.8)

The call generator creates instances of CALLRECORD and INCSISHW processes at exponentially distributed time intervals to represent the arrival of telephone calls. The inter-arrival time is determined by the telephone traffic offered to the exchange in erlangs, A, say. Suppose that

C = the number of calls originating during a long period T

t = the average duration of a call.

and A = The average number of simultaneous calls in progress during the period T that is

A = 'traffic flow' or 'traffic intensity' in erlangs,

Now, the volume of traffic may be defined simultaneously as:

$$\text{Volume of Traffic} = C\text{x}t$$

$$= A\text{x}T$$

$$\therefore \quad A\text{x}T = C\text{x}t$$

$$\text{or } A = \frac{Ct}{T}$$

Since C calls originate during the period T, and the average holding time as a fraction of T is $\bar{t}$, then the average number of calls which originate during t = $C\bar{t}$ = A = traffic intensity in erlangs.

Taking a traffic intensity of 600 erlangs, an absolute value of t of 3 minutes and T as being 1 hour then

$$600 = C \times \frac{3}{60}$$

$$\therefore C = \text{calls generated in 1 hour} = 12{,}000$$

$$\therefore \quad \text{Average call inter-arrival time} = \frac{3600}{12000} = 0.3 \text{ sec.}$$

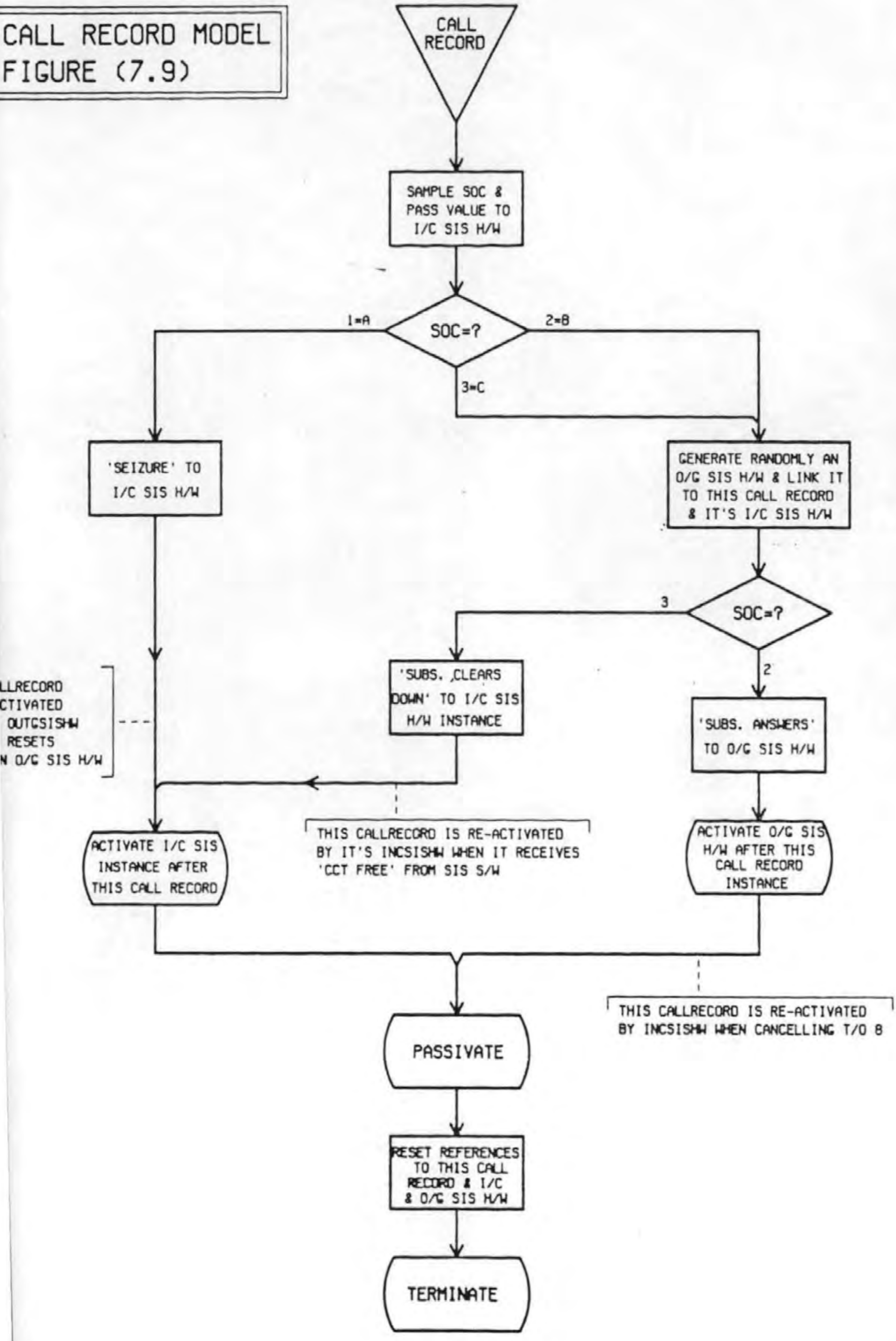
Since the traffic is considered as originating from an infinite number of sources, that is a Poisson call arrival, the inter-arrival distribution is sampled from a negative exponential distribution with the mean $= \frac{1}{0.3}$ sec^{-1} = 3.33×10^{-6} μsec^{-1}.

When a telephone call is due to be generated, CALLGENERATOR will select, randomly, a free incoming circuit. An incoming circuit is free if no CALLRECORD process instance with the identity of the circuit exists in the model. When a free circuit is selected, CALLGENERATOR creates a CALLRECORD instance and an INCSISHW instance for the call, passing the circuit identity selected to both instances as the call identity. It also, inter links the two instances together using reference variables local to each process instance.

The call generator generates a few calls at the start of the simulation run in order to reduce the transient period. Other measures to reduce this transient period, or 'warm-up' period, are discussed in connection with the CALLRECORD process. The call generator then holds for an inter arrival time sampled from a negative exponential distribution whose mean is determined by the traffic value before going through the above described process of a call generation.

When a CALLRECORD process instance is created, (Figure (7..9)) it is scheduled to be active after the CALLGENERATOR. To reduce further the transient period, the concept of the 'state of Call' is introduced in CALLRECORD (2029-2114). A newly generated call may be in one of three states marked as A, B and C in Figure (7.1). The particular state is randomly selected by the CALLRECORD process instance as its first action when activated by the CALLGENERATOR process. The concept of the state of call is introduced to reduce the length of the transient period in the simulator. This is very useful in view of the fact that if all calls injected into the system are generated starting with a 'Seizure' hardware message arrival, that is at state A, then a minimum of three minutes of simulated time is required to obtain a reasonable mix of calls to justify the

CALL RECORD MODEL
FIGURE (7.9)


CALL RECORD
SAMPLE SOC & PASS VALUE TO I/C SIS H/W
SOC=?
1=A
2=B
3=C
'SEIZURE' TO I/C SIS H/W
GENERATE RANDOMLY AN O/C SIS H/W & LINK IT TO THIS CALL RECORD & IT'S I/C SIS H/W
SOC=?
3
2
'SUBS. CLEARS DOWN' TO I/C SIS H/W INSTANCE
'SUBS. ANSWERS' TO O/C SIS H/W
LLRECORD
CTIVATED
OUTGSISHW
RESETS
N O/C SIS H/W
THIS CALLRECORD IS RE-ACTIVATED BY IT'S INCSISHW WHEN IT RECEIVES 'CCT FREE' FROM SIS S/W
ACTIVATE I/C SIS INSTANCE AFTER THIS CALL RECORD
ACTIVATE O/C SIS H/W AFTER THIS CALL RECORD INSTANCE
THIS CALLRECORD IS RE-ACTIVATED BY INCSISHW WHEN CANCELLING T/O 8
PASSIVATE
RESET REFERENCES TO THIS CALL RECORD & I/C & O/C SIS H/W
TERMINATE

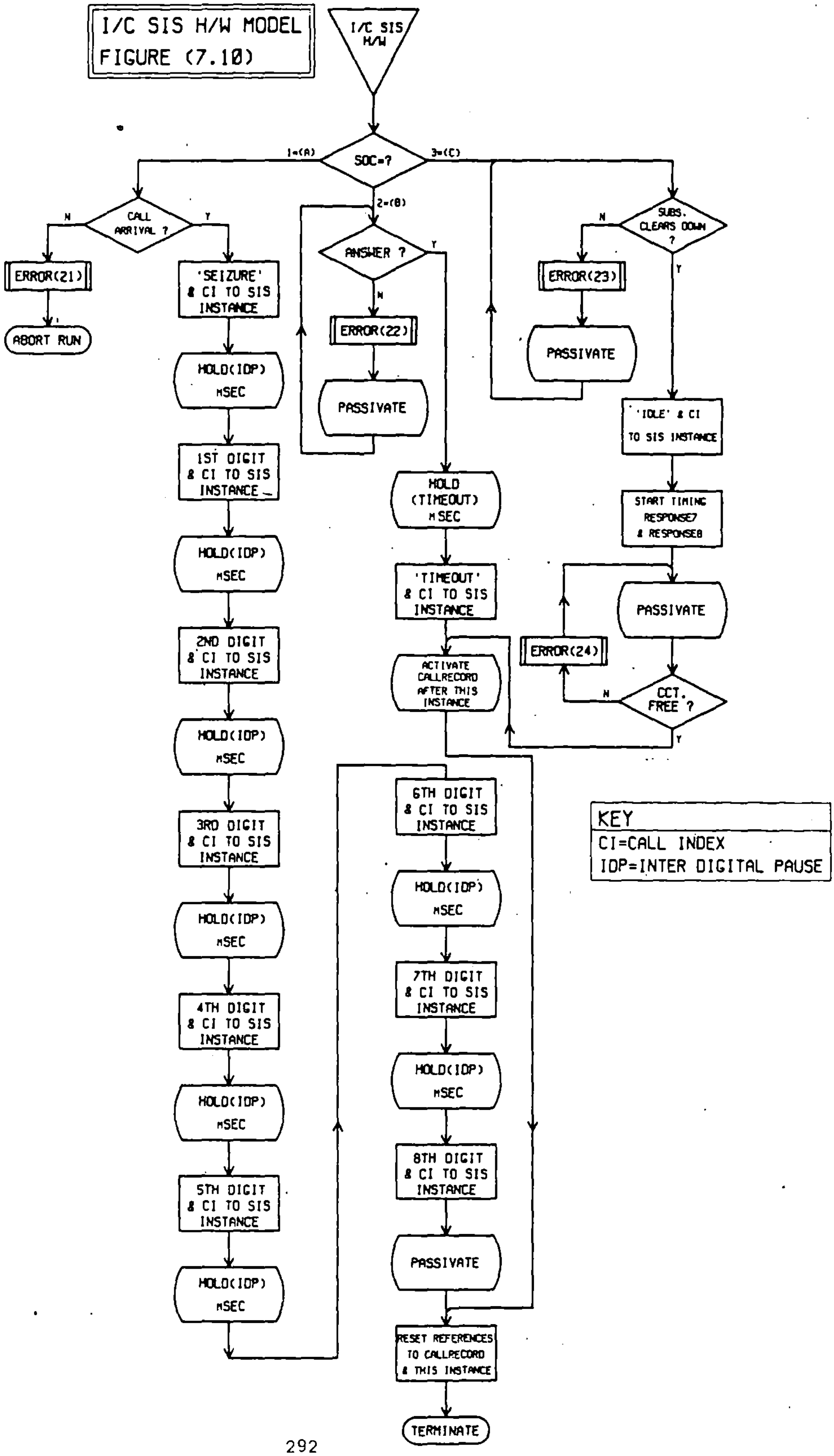
collection of statistical data. The reason for this is that the post dialling delay takes six seconds on average and the speech phase three minutes on average, which results in a longer run because of the fine time grain of the Mark II BL System simulator and the fact that it is not doing much useful work, apart from housekeeping, while holding for those respective times. By generating the individual calls to start at A, B or C that is with a 'Seizure' or 'Subscriber Answers' or 'Subscriber Clears Down' event, the post dialling delay and the speech phase are eliminated. To conserve the flow of traffic through the system, a call generated at state A has got to terminate before the start of state B. Likewise, a call generated at state B has got to terminate before the start of state C. The simulation processes modelling the telephone calls that is CALLRECORD, INCSISHW and OUTGSISHW have been designed and synchronized to ensure this. The net result of this strategy of call generation is that we are able to get a relatively fast simulation for the DMNSC with simulated time to real time ratio of about 7 to 1 on average.

When activated, a CALLRECORD instance determines randomly the state of its call and passes that information to its INCSISHW instance. If the state of call sampled is state A, the CALLRECORD instance will indicate a 'Seizure' to the calling subscriber line circuit that is the call INCSISHW instance. It will then schedule the INCSISHW instance and passivates. On the other hand, if the state of call sampled is state B or C, then judging by Figure (7.1), an OUTGSISHW process instance must be created for the call to proceed. The CALLRECORD instance selects randomly a free outgoing circuit number, creates an instance of OUTGSISHW with that circuit identity and links it both ways to its INCSISHW and CALLRECORD instances. If the state of call generated is state B the CALLRECORD instance will indicate to the OUTGSISHW that the called subscriber has answered before scheduling the OUTGSISHW instance to run after this CALLRECORD instance when it passivates. If the state of call is that of state C, the the INCSISHW instance is notified that the calling subscriber has cleared down before scheduling it to run when this CALLRECORD passivates. For a type A call this CALLRECORD instance is re-scheduled by the OUTGSISHW instance when resetting the timeout after the last digit. If it is of type B or C then the CALLRECORD instance will be re-scheduled by the INCSISHW instance when cancelling the timeout after the arrival of 'Answer' or 'Circuit Free' message from SISSW instance respectively as shown in Figure (7.1). When re-activated, the CALLRECORD instance resets the references to itself, the INCSISHW and OUTGSISHW instances to NONE and terminates.

The INCSISHW process flow chart is shown in Figure (7.10) and the listing on (2166-2333). For each call generated an instance of this process is created at the time of creation of a CALLRECORD instance by the CALLGENERATOR process. This process instance is first activated by its CALLRECORD process instance if the state of call is A or C and by the respective SIS instance for a type B call.

If it is a state A, then a 'Seizure' message will have been sent to this instance by its CALLRECORD. The first action for INCSISHW instance is to check for the presence of this message. If it is not there, an error message is printed and the simulation stopped. This is a serious error which will throw the model out of synchronism and distort the measurements if the run is allowed to continue. Assuming that the 'Seizure' message is found, it will be passed together with the incoming circuit identity to the SIS replication. INCSISHW instance then alternately holds for an interdigital pause then sends

I/C SIS H/W MODEL
FIGURE (7.10)


I/C SIS H/W
SOC=?
1=(A)
2=(B)
3=(C)
CALL ARRIVAL ?
N
Y
ERROR(21)
ABORT RUN
'SEIZURE' & CI TO SIS INSTANCE
HOLD(IDP) mSEC
1ST DIGIT & CI TO SIS INSTANCE
HOLD(IDP) mSEC
2ND DIGIT & CI TO SIS INSTANCE
HOLD(IDP) mSEC
3RD DIGIT & CI TO SIS INSTANCE
HOLD(IDP) mSEC
4TH DIGIT & CI TO SIS INSTANCE
HOLD(IDP) mSEC
5TH DIGIT & CI TO SIS INSTANCE
HOLD(IDP) mSEC
ANSWER ?
Y
N
ERROR(22)
PASSIVATE
HOLD (TIMEOUT) mSEC
'TIMEOUT' & CI TO SIS INSTANCE
ACTIVATE CALLRECORD AFTER THIS INSTANCE
6TH DIGIT & CI TO SIS INSTANCE
HOLD(IDP) mSEC
7TH DIGIT & CI TO SIS INSTANCE
HOLD(IDP) mSEC
8TH DIGIT & CI TO SIS INSTANCE
PASSIVATE
RESET REFERENCES TO CALLRECORD & THIS INSTANCE
TERMINATE
SUBS. CLEARS DOWN ?
N
Y
ERROR(23)
PASSIVATE
'IDLE' & CI TO SIS INSTANCE
START TIMING RESPONSE7 & RESPONSE8
PASSIVATE
ERROR(24)
CCT. FREE ?
N
Y
KEY
CI=CALL INDEX
IDP=INTER DIGITAL PAUSE

a digit and the circuit identity to SIS replication until the eighth digit is sent after which the instance passivates. When reactivated later by its CPS replication on receiving the set-up switch response from DSS, it resets its reference variables to NONE and terminates.

For a state B call, INCSISHW instance first checks for the presence of an 'Answer' message from the SIS replication and prints an error message if the message is not found. It resets a covering time out started by its SIS replication after a delay sampled from a uniform distribution. It schedules its CALLRECORD instance, resets the references to NONE and terminates. When the CALLRECORD instance is activated, it resets all its references to NONE and terminates. Special care has been taken in the DMNSC model to reset the reference variables to transient processes to NONE when they terminate to reduce the time used for garbage collection in SIMULA and consequently increase the run efficiency of the simulator.

For a state C call, INCSISHW instance checks for the presence of a 'Subscriber Clears Down' hardware message from its CALLRECORD instance and outputs a error message if not present. It sends a hardware message 'Idle' and the call identity to the SIS replication, starts 'Release Time' and 'Clear Forward Transfer Time' time measurements and passivates. When reactivated by the SIS replication it checks for the presence of a 'Circuit Free' message, schedules its CALLRECORD, sets the reference variables to NONE and terminates marking the end of its telephone call.

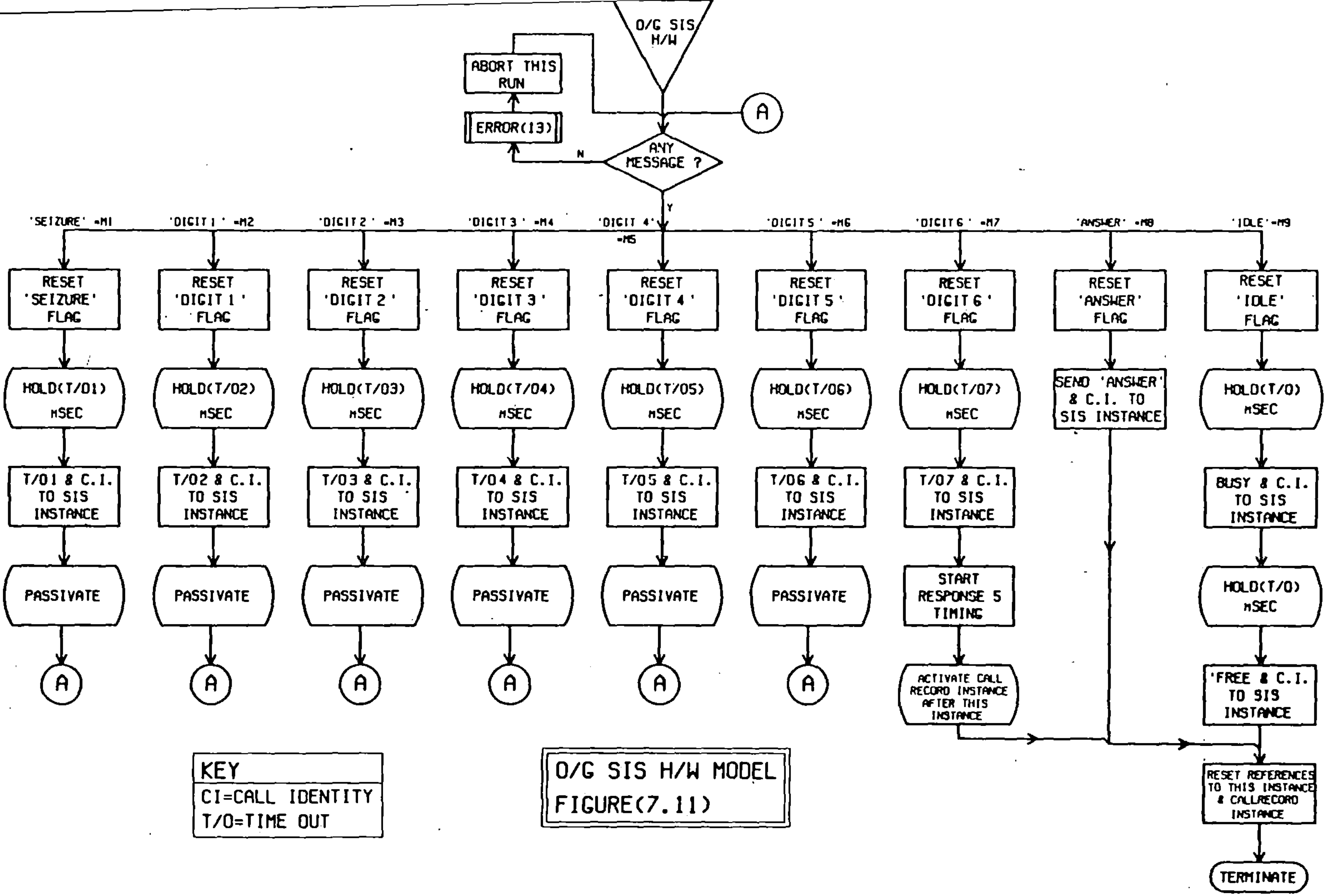
A third process, that together with CALLRECORD and INCSISHW complete the modelling of a telephone call is the OUTGSISHW process (3234-3379). It models the call scenario at the outgoing end of the exchange by modelling the hardware messages exhanged with SIS replications and indicating the states through which a call passes at the outgoing end as shown in Figure (7.11).

If the state of call generated is A, that is a call starting at the beginning of its life scenario, then the responsbility of the selection of a free outgoing circuit is vested on the appropriate CPS replication process. This selection is made sometime later after the call generation. When a free outgoing circuit is selected, an OUTGSISHW process instance is created with the outgoing circuit identity passed as an actual parameter. It is linked both ways to its corresponding CALLRECORD and INCSISHW instances.

On the other hand, if the state of call sampled is B or C then the OUTGSISHW process instance is created by the CALLRECORD instance. and inter-linked at the time of sampling of the state of call.

Whenever an OUTGSISHW instance is activated, it always checks for the arrival of a hardware message as in Figure (7.11) and if no such message is pending it outputs an error message and terminates, otherwise, if a message is found it is serviced and then the process passivates. The process instance is always activated by its SIS replication or its CALLRECORD instance after sending a hardware message. For the messages sent by the SIS replication, a covering time-out is started in SIS and the OUTGSISHW has to send back a timeout reset signal within the specified timeout limits. The reset signals are sent after a delay sampled from uniform distributions (TIMEOUT1 to TIMEOUT7). After resetting the covering timeout for the 6th digit, OUTGSISHW starts the timing measurement of the 'Switch Through Time (Hardware)' response time, schedules its CALLRECORD, resets its reference to NONE and terminates.

If the call is generated to start at label B in Figure (7.1) that is state of call B, then after servicing the 'Subscriber Answer' message



KEY
CI=CALL IDENTITY
T/O=TIME OUT

O/G SIS H/W MODEL
FIGURE(7.11)

from its CALLRECORD, OUTGSISHW immediately resets it reference variables to NONE and terminates.

For calls with state of call C, the OUTGSISHW instance services the 'Idle' message by sending back 'Busy' and 'Free' messages to its SIS instance, resets its references to NONE and terminates.

7.2.6 Quantification of the Delays in the DMNSC

To validate the design of the DMNSC it is necessary to quantify particular delays between certain instances in the call processing sequence in the DMNSC. Such information cannot be obtained from the DMNSC prototype and the only feasible way is through simulation. With reference to Figure (7.1), the encircled numbers indicate the start and finish of response times in which there is particular interest, as follows:-

1) Outgoing circuit selection time (hardware to hardware)
2) " " " " (task space to task space)
3) Signal transfer time (hardware to hardware)
4) " " " (task space to task space)
5) Switch through time (hardware)
6) Switch through time (software)
7) Release time
8) Clear forward transfer time.

The DMNSC configuration tested consists of 1061 circuits, 2 CPUs, 2 instances of SIS, 4 instances of CPS, 1 instance of DSS and a traffic of 0.6 erlang per circuit. For each quantity of interest (1) to (8) the average value, the standard deviation, the minimum and the maximum are calculated. These values are tabulated in Table (7.6).

Measure Delay of Interest	Average Value (μs)	Standard Deviation (μs)	Minimum Value (μs)	Maximum Value (μs)
O/G CCT. Selection Time (H/W to H/W)	37120.0	2642.2	34176.1	40591.0
O/G CCT. Selection Time (Task space to Task space)	24925.7	66.6	24868.0	25024.0
Signal Transfer Time (H/W to H/W)	16055.5	2630.0	13091.2	19478.0
Signal Transfer Time (Task space to Task space)	6114.8	10.3	6100.0	6121.0
Switch through time (H/W)	25088.0	2093.2	23602.0	27482.0
Switch through time (S/W)	27720.3	1162.3	26735.0	29002.0
Release Time	33527.5	11928.8	21619.3	46519.0
Clear forward Transfer Time	18451.6	7770.1	11749.0	27349.0

TABLE (7. 6.) : QUANTIFICATION OF THE DELAYS OF INTEREST IN THE DMNSC

The values are found to be within the DMNSC specifications. Experiments for larger sizes of the DMNSC handling up to 20,000 erlangs of traffic may be conducted if required to validate the DMNSC design. As a by-product of this experiment, the lockouts occupancies in the DMNSC, namely, FREELO, SUSPLO and INTLO are examined. As expected they are found to be extremely low ranging from 1.36% to 2.47%.

CHAPTER 8

CONCLUSIONS

8.1 ACHIEVEMENTS

In this thesis we have developed a philosophy for simulating stored program control switching systems and have presented a methodology for a computer-aided design of such systems through simulation. The multi-level process simulation proposed and presented eliminates the limitations of the real-time environment simulation and flat-level simulation methodologies used at present. A real-time environment simulator is costly to develop and is limited in its application to the testing and debugging of the application software. There is no means of evaluating the performance of the real-time operating system and/or proposed future modifications and extensions. On the other hand, flat-level software simulators are monolithic in nature, difficult to verify and validate and costly to run. The resulting lack of credibility manifests itself in the suspicion with which the simulator results are treated.

We have developed an alternative simulation methodology of SPC systems where the complex SPC system is divided into its natural sub-systems. The hardware and software sub-systems in a system are transformed in a 1 to 1 transformation process to their corresponding models. The sub-system models are placed at different levels in a hierarchy according to the role they play in the real system. For example,the model of the real-time operating system upon which the other sub-systems depend for their functioning is placed at the bottom level of the simulator, with the applications models one level higher. The adoption of the philosophy of a multi-level process-based simulation enabled us to develop a simulator in which the interfaces and driving tables of the real system are preserved. The resulting simulator looks very similar to the high-level structure of the real system. Moreover, the detailed trace incorporated in the simulator traces the interaction of the sub-systems models in a manner readily recognizable to the software design engineers. These factors helped to sell the simulator to the community of users of software design engineers.

The importance of verification and validation in the simulation process has been stressed in Chapter 6. This aspect of model building has sometimes been ignored by some simulation analysts with a consequent lack of credibility and of confidence in the simulator. In contrast to the flat-level simulator of System X developed by the British Telecommunications (formerly the British Post Office), our simulator has been thoroughly verified and validated. We feel that, for a complex and highly-interactive system such as the Mark II BL multi-processor system, the verification of the logic of individual modules in the simulator as well as the sequences of interactions between the modules is extremely important. Indeed, getting the quasi-parallel serial execution of SIMULA to reproduce faithfully the parallel interactions of the process allocators, the interrupt triplicates Process and the individual processes running on different CPUs proved to be the most time-consuming phase in the development of the simulator. For example, on one occasion it was noted from the trace that the local sequence control of a process changes its position and skips a statement when a process was re-activated after being interrupted and pre-empted. No obvious reason was apparent and the proper sequence was obtained by a re-arrangement of the sequencing statements in the process allocator and the interrupt triplicates. On another occasion during the validation phase, the storage allocator was not activated, because when it is loaded and had a higher priority, the CPU in which it was running was still referred to as LPA by INTRIP (Chapter 6) which was waiting in a queue for its process allocator to finish executing its present instruction. LPA references the process allocator on LCPU, the CPU running the lowest priority process. The process allocator on which the storage allocator is to run, senses that it is LPA and that INTIM is waiting for it. Therefore, instead of activating the storage allocator, it activates INTIM. When activated, INTIM comes out of its waiting queue and updates LPA as

LPA :- LCPU.MYPA

and hence points to a different CPU. INTRIP will then instruct LPA to re-schedule and the storage allocator is never activated again. To cope with such situations additional pre-cautionary logic is incorporated in INTRIP thus whenever INTRIP is activated to service an interrupt, it checks whether LPA is changed since it was last activated and whether the suspended queue interrupt servicing is still required. If not, as in this case, then the interrupt is not serviced and the appropriate process such as the storage allocator here is re-activated. Other similar situations are corrected by taking appropriate actions until the simulator is able to cope with expected and unexpected combinations of events. The effort spent on the verification and validation proved to be worthwhile, judging by the validation experiment results.

The hierarchical multi-level process simulator proved to be capable of meeting its objectives which were; firstly, to provide a tool for the designers of the real-time operating system of the Mark II BL multiprocessor system which could be used for the performance evaluation of the prototype design and the evaluation of possible modifications and extensions; secondly, to provide a tool for the performance evaluations of the members of System X family of exchanges. The ability of the simulator in this respect is demonstrated by the simulation of the Digital Main Network Switching Centre (DMNSC). Valuable information to the processor utility and call processing sub-systems designers was obtained using the simulator as described in Chapter 7. Such information is not possible to obtain in this detail otherwise.

By adopting our philosophy of hierarchical multi-level process simulation for SPC systems, we were able to develop a simulator package which is open-ended, flexible in its level of detail of individual modules, adaptable, relatively fast and smaller in size than the corresponding flat-level simulator developed for System X before. Our simulator is about 3.5K statements and runs at a speed of 7:1 to 10:1 relative to real-time, depending on the configuration being modelled and the traffic intensity.

Aside from the original objectives and owing to the credibility demonstrated by the simulator, it is now being used to guide the analytical modelling process of the System X family of exchanges. The simulator is being used to test the simplifying assumptions, such as the assumption of nodal independence, necessary for the approximate analytical modelling of System X. It will also be used for validating the analytical model when fully developed. Another use of the simulator is the evaluation of the proposed processor sub-system architecture for System X Release 2. To reduce the store contention in a tightly-coupled system such as the Mark II BL multiprocessor system, it is now proposed that clusters of a few CPUs each will be more appropriate. The simulator will be adapted to evaluate the performance of clusters of CPUs relative to the existing Mark II BL multiprocessor system architecture.

The simulator developed fits nicely with the existing emulator, which emulates a single CPU, to form a powerful integrated computer-aided design package for SPC systems design. The emulator is used for the verification of applications software and the simulator for the performance evaluation of SPC exchanges.

8.2 SUGGESTIONS FOR FURTHER WORK

Design modifications to the Overload Control Sub-system (OCS) have been proposed recently in an attempt to optimize its performance (GREE 80). To evaluate these modifications a fairly detailed simulation of the Overload Control Sub-system has been requested. The model of the overload control sub-system will be incorporated in the DMNSC model, It will then be possible to optimize the performance of the Overload Control Sub-system through conducting experiments on the DMNSC model.

The problems of determining the length of a transient period in a run and the sample size to achieve a specific confidence level in the results have been discussed by many simulation analysts before (FISH 67, KLEI 74, MIHR 72). These problems arise when the data is correlated such as in queuing systems. An original approach, the regenerative approach, has been proposed recently (CHAN 78, IGLE 78). This approach is based on the philosophy that many stochastic systems have the property of 'starting afresh' probabilistically from time to time. This enables the observation of independent and identically distributed blocks of data in the course of the simulation.

It will be nicer to incorporate the regenerative process feature in the simulator such that a PROCESS CLASS REGENERATIVE will determine the run length according to the required confidence level passed as a parameter at run time.

Interactive simulation packages are flexible and more user-oriented. Features such as the ability to stop the simulator during a run, interrogate and/or change the variable values and assemble a configuration to be run by a questionnaire type of conversation between the user and the package at the start of a session will make the simulator more attractive.

APPENDIX A

MK II BL MULTIPROCESSOR SYSTEM SIMULATOR IN CSL

TABLE (A-1) : PERIODIC PROCESSES TASK TABLE (PPTSK)

PROCESS INDEX	INCOMING TASK INDEX	TASK PRIORITY
1	3	6
2	2	10
3	6	7
4	4	4
5	3	9
6	2	8
7	1	7
8	6	6
9	4	4
10	3	5

TABLE (A-2) PERIODIC PROCESS TABLES

ABLE A : PROCESS 2

PROCESS TI	CALLED PI	CALLED PROCESS TI	TASK PRIORITY
1	7	0	6
2	8	3	4
3	4	0	8

ABLE B : PROCESS 3

PROCESS TI	CALLED PI	CALLED PROCESS TI	TASK PRIORITY
1	2	3	6
2	8	1	3
5	9	0	13
6	5	0	12

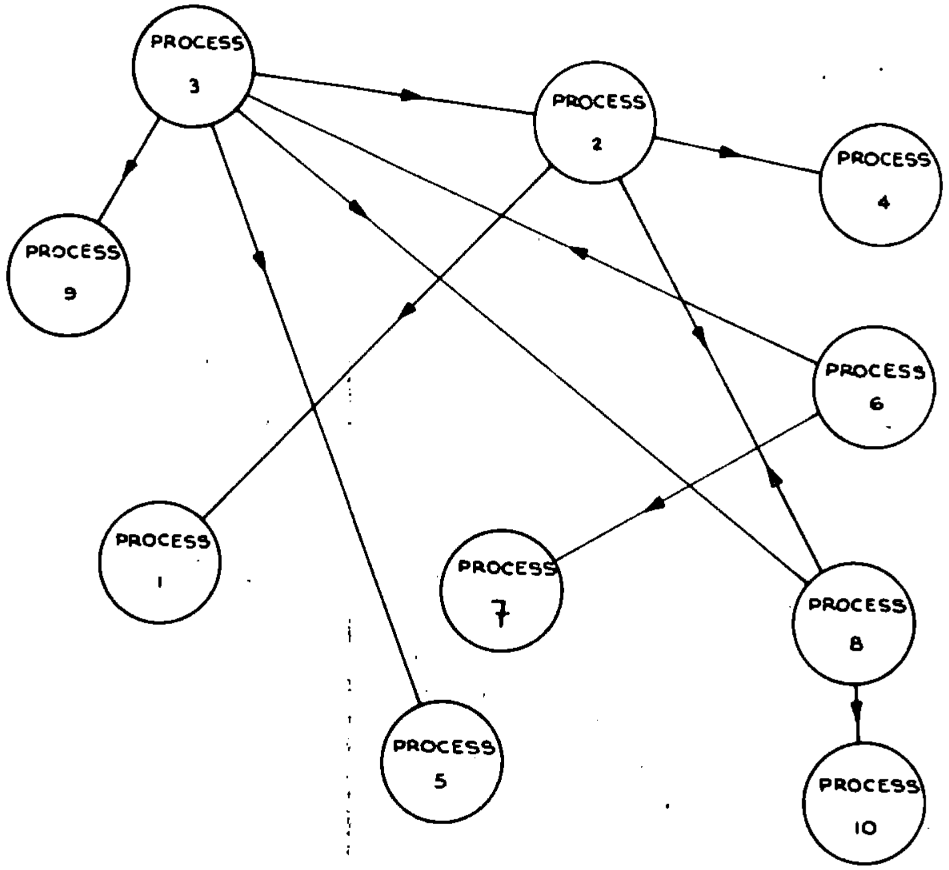
ABLE C : PROCESS 6

PROCESS TI	CALLED PI	CALLED PROCESS TI	TASK PRIORITY
2	1	0	2
3	3	4	10

BLE D : PROCESS 8

PROCESS TI	CALLED PI	CALLED PROCESS TI	TASK PRIORITY
1	2	3	3
3	10	0	5

APPENDIX (A-1) : COMMUNICATION BETWEEN PERIODIC APPLICATION PROCESSES

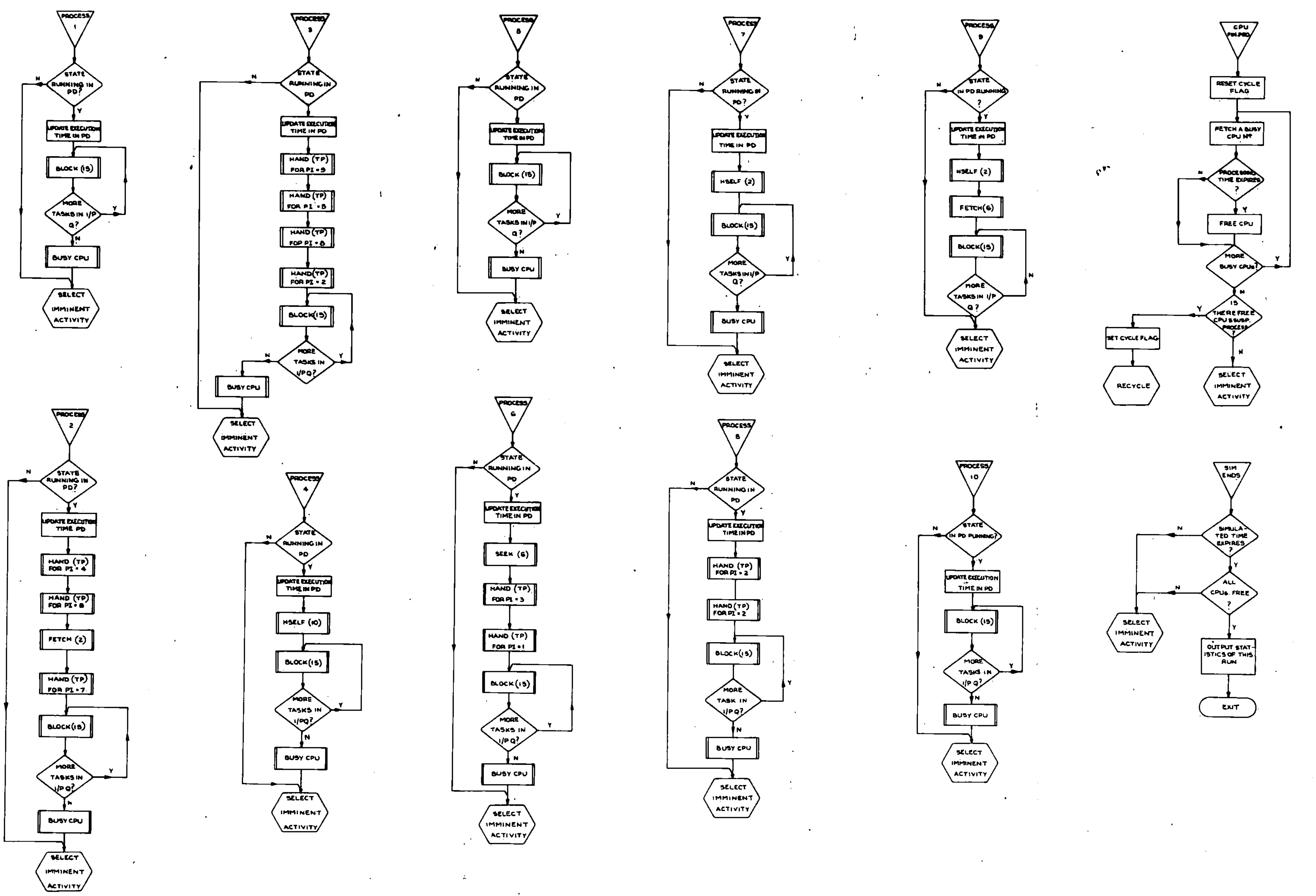


PERIODIC TASK BIT MAP (PTBM)

PROCESS Nº	10	9	8	7	6	5	4	3	2	1	CLOCK INT Nº
	0	0	1	0	1	0	0	1	1	0	1
	0	1	0	1	0	0	1	0	0	1	2
	1	0	1	0	1	1	0	1	1	0	3
	0	0	0	0	0	0	0	0	0	1	4

POINTER

APPENDIX (A-2) MODEL ACTIVITIES


ACTIVITY PROCESS 1

ACTIVITY PROCESS 2

ACTIVITY PROCESS 3

ACTIVITY PROCESS 4

ACTIVITY PROCESS 5

ACTIVITY PROCESS 6

ACTIVITY PROCESS 7

ACTIVITY PROCESS 8

ACTIVITY PROCESS 9

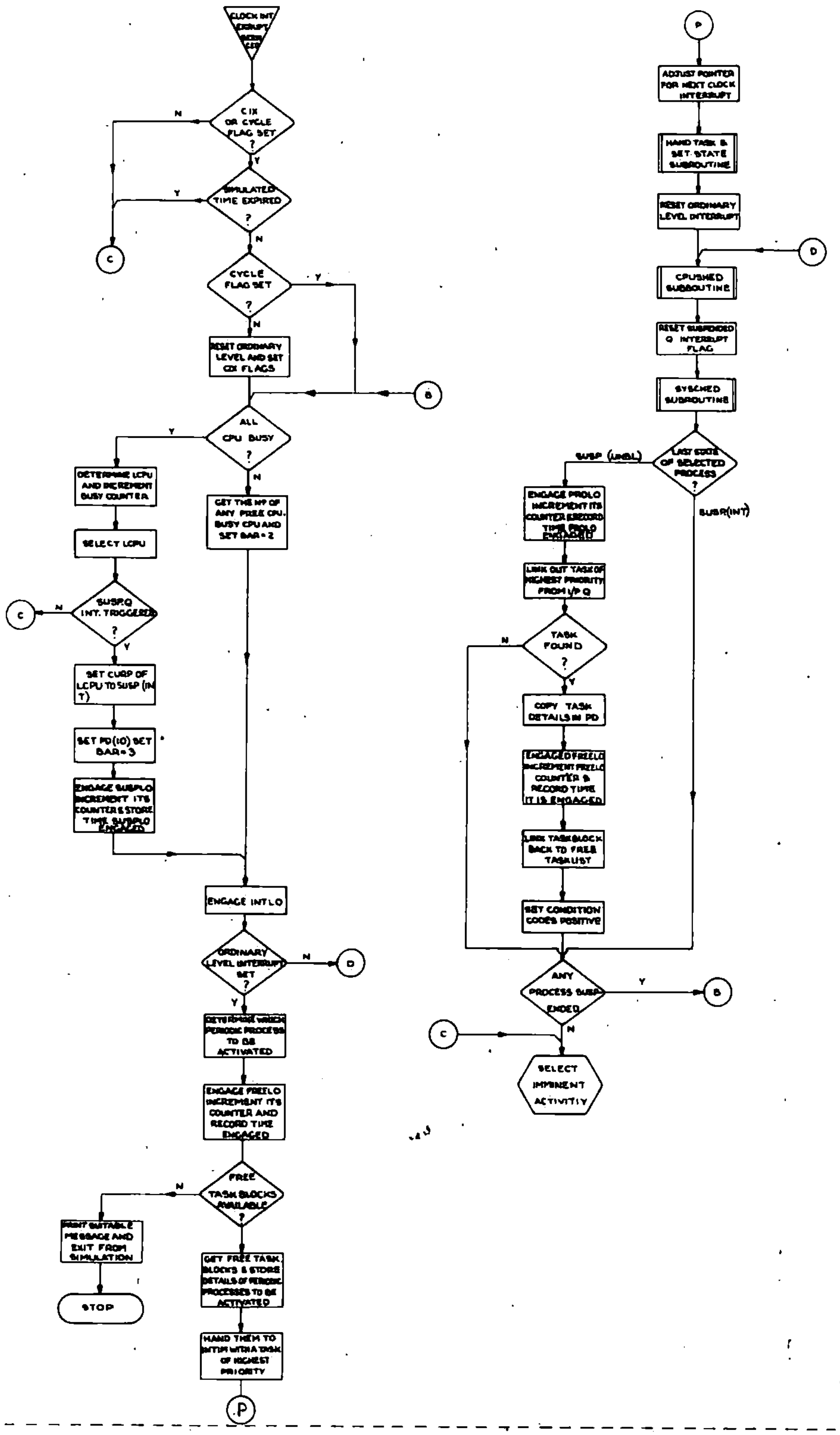
ACTIVITY PROCESS 10

ACTIVITY CPU FINISHES PROCESSING

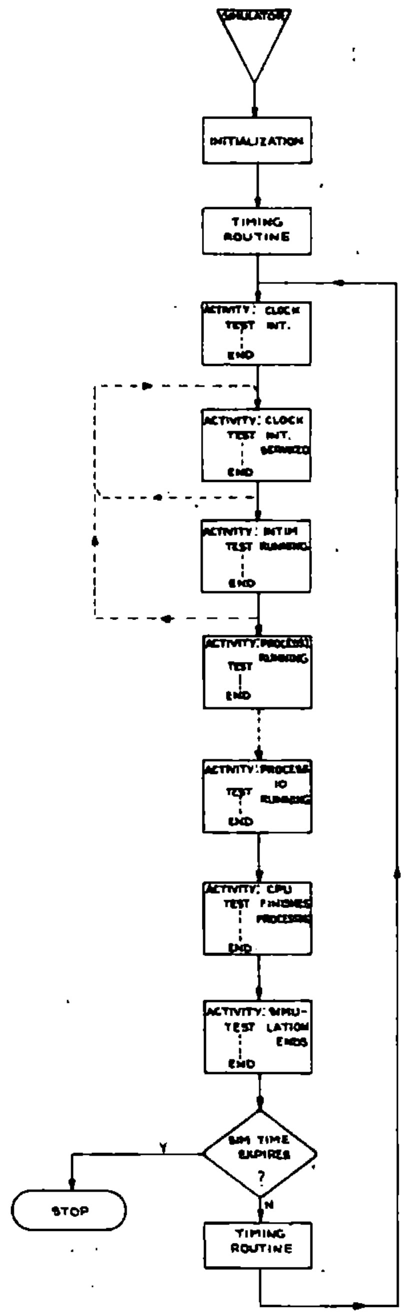
ACTIVITY SIMULATION ENDS

APPENDIX (A3): MODEL ACTIVITIES. AND INITIALIZATION.

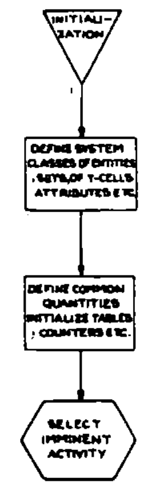
ACTIVITY CLOCK INTERRUPT SERVICED

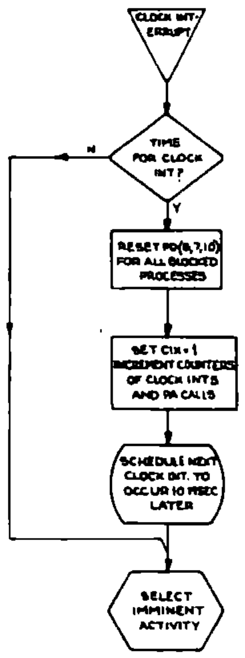

CLOCK INT ERRUPT SERVI CED
CIX OR CYCLE FLAG SET ?
N
Y
SIMULATED TIME EXPIRED ?
Y
N
C
CYCLE FLAG SET ?
Y
N
RESET ORDINARY LEVEL AND SET CIX FLAGS
B
ALL CPU BUSY ?
Y
N
DETERMINE LCPU AND INCREMENT BUSY COUNTER
SELECT LCPU
SUSP.Q INT. TRIGGERED ?
N
C
Y
SET CURP OF LCPU TO SUSP (INT)
SET PD(10) SET BAR=3
ENGAGE SUSPLO INCREMENT ITS COUNTER & STORE TIME SUSPLO ENGAGED
GET THE Nº OF ANY FREE CPU. BUSY CPU AND SET BAR=2
ENGAGE INTLO
ORDINARY LEVEL INTERRUPT SET ?
N
D
Y
DETERMINE WHICH PERIODIC PROCESS TO BE ACTIVATED
ENGAGE FREELO INCREMENT ITS COUNTER AND RECORD TIME ENGAGED
FREE TASK BLOCKS AVAILABLE ?
N
PRINT SUITABLE MESSAGE AND EXIT FROM SIMULATION
STOP
GET FREE TASK BLOCKS & STORE DETAILS OF PERIODIC PROCESSES TO BE ACTIVATED
HAND THEM TO INTIM WITH A TASK OF HIGHEST PRIORITY
P
P
ADJUST POINTER FOR NEXT CLOCK INTERRUPT
HAND TASK & SET STATE SUBROUTINE
RESET ORDINARY LEVEL INTERRUPT
D
CPUSHED SUBROUTINE
RESET SUSPENDED Q INTERRUPT FLAG
SYSCHED SUBROUTINE
LAST STATE OF SELECTED PROCESS ?
SUSP (UNBL)
SUSP(INT)
ENGAGE PROLO INCREMENT ITS COUNTER & RECORD TIME PROLO ENGAGED
LINK OUT TASK OF HIGHEST PRIORITY FROM I/P Q
TASK FOUND ?
N
Y
COPY TASK DETAILS IN PD
ENGAGED FREELO INCREMENT FREELO COUNTER & RECORD TIME IT IS ENGAGED
LINK TASKBLOCK BACK TO FREE TASKLIST
SET CONDITION CODES POSITIVE
ANY PROCESS SUSP-ENDED
Y
B
N
C
SELECT IMMINENT ACTIVITIY

SIMULATOR PROGRAM LAYOUT



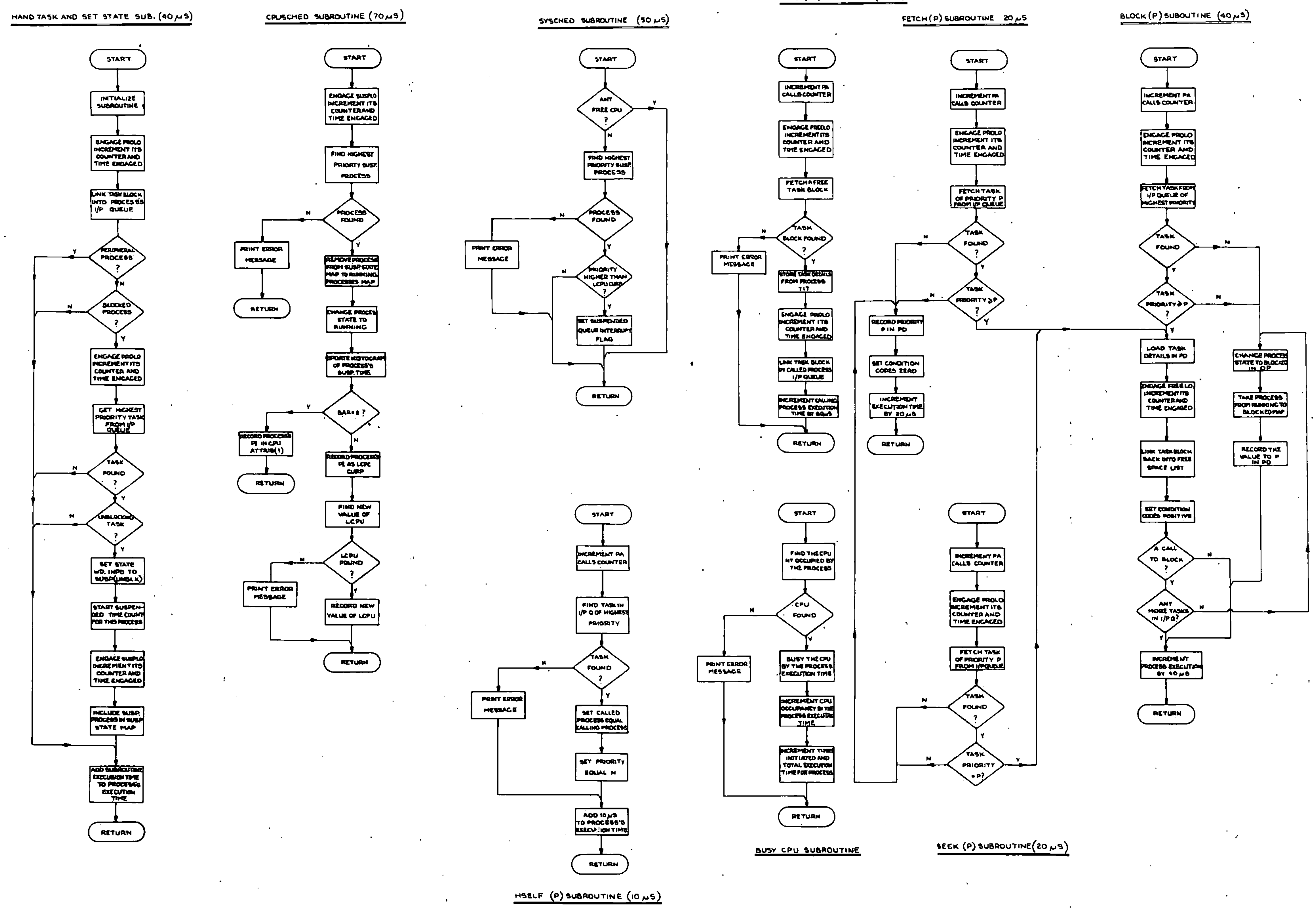
INITIALIZATION.


INITIALI-ZATION
DEFINE SYSTEM CLASSES OF ENTITIES, SETS OF T-CELLS ATTRIBUTES ETC.
DEFINE COMMON QUANTITIES INITIALIZE TABLES, COUNTERS ETC.
SELECT IMMINENT ACTIVITY


CLOCK INT-ERRUPT
TIME FOR CLOCK INT ?
N
Y
RESET PD(6,7,10) FOR ALL BLOCKED PROCESSES
SET CIX=1 INCREMENT COUNTERS OF CLOCK INTS AND PA CALLS
SCHEDULE NEXT CLOCK INT. TO OCCUR 10 MSEC LATER
SELECT IMMINENT ACTIVITY

ACTIVITY CLOCK INTERRUPT.

APPENDIX (A4) : SUBROUTINES FOR CSL MODEL


HAND TASK AND SET STATE SUB. (40 μS)
START
INITIALIZE SUBROUTINE
ENGAGE PROLO INCREMENT ITS COUNTER AND TIME ENGAGED
LINK TASK BLOCK INTO PROCESS'S I/P QUEUE
PERIPHERAL PROCESS ?
BLOCKED PROCESS ?
ENGAGE PROLO INCREMENT ITS COUNTER AND TIME ENGAGED
GET HIGHEST PRIORITY TASK FROM I/P QUEUE
TASK FOUND ?
UNBLOCKING TASK ?
SET STATE WD. INPD TO SUSP(UNBLK)
START SUSPENDED TIME COUNT FOR THIS PROCESS
ENGAGE SUSPLO INCREMENT ITS COUNTER AND TIME ENGAGED
INCLUDE SUSP. PROCESS IN SUSP. STATE MAP
ADD SUBROUTINE EXECUSION TIME TO PROCESS'S EXECUTION TIME
RETURN
Y
N
CPUSCHED SUBROUTINE (70 μS)
START
ENGAGE SUSPLO INCREMENT ITS COUNTER AND TIME ENGAGED
FIND HIGHEST PRIORTY SUSP. PROCESS
PROCESS FOUND
PRINT ERROR MESSAGE
RETURN
REMOVE PROCESS FROM SUSP. STATE MAP TO RUNNING PROCESSES MAP
CHANGE PROCESS STATE TO RUNNING
UPDATE HISTOGRAM OF PROCESS'S SUSP. TIME
BAR=2 ?
RECORD PROCESS'S PI IN CPU ATTRIB(1)
RETURN
RECORD PROCESS'S PI AS LCPC CURP
FIND NEW VALUE OF LCPU
LCPU FOUND ?
PRINT ERROR MESSAGE
RECORD NEW VALUE OF LCPU
RETURN
SYSCHED SUBROUTINE (50 μS)
START
ANY FREE CPU ?
FIND HIGHEST PRIORITY SUSP. PROCESS
PROCESS FOUND
PRINT ERROR MESSAGE
PRIORITY HIGHER THAN LCPU CURP ?
SET SUSPENDED QUEUE INTERRUPT FLAG
RETURN
HAND (TP) SUBROUTINE (60 μS)
START
INCREMENT PA CALLS COUNTER
ENGAGE FREELO INCREMENT ITS COUNTER AND TIME ENGAGED
FETCH A FREE TASK BLOCK
TASK BLOCK FOUND ?
PRINT ERROR MESSAGE
STORE TASK DETAILS FROM PROCESS TIT
ENGAGE PROLO INCREMENT ITS COUNTER AND TIME ENGAGED
LINK TASK BLOCK IN CALLED PROCESS I/P QUEUE
INCREMENT CALLING PROCESS EXECUTION TIME BY 60 μS
RETURN
FETCH (P) SUBROUTINE 20 μS
START
INCREMENT PA CALLS COUNTER
ENGAGE PROLO INCREMENT ITS COUNTER AND TIME ENGAGED
FETCH TASK OF PRIORITY P FROM I/P QUEUE
TASK FOUND ?
TASK PRIORITY ≥ P ?
RECORD PRIORITY P IN PD
SET CONDITION CODES ZERO
INCREMENT EXECUTION TIME BY 20 μS
RETURN
BLOCK (P) SUBOUTINE (40 μS)
START
INCREMENT PA CALLS COUNTER
ENGAGE PROLO INCREMENT ITS COUNTER AND TIME ENGAGED
FETCH TASK FROM I/P QUEUE OF HIGHEST PRIORITY
TASK FOUND
TASK PRIORITY ≥ P ?
LOAD TASK DETAILS IN PD
ENGAGE FREELO INCREMENT ITS COUNTER AND TIME ENGAGED
LINK TASK BLOCK BACK INTO FREE SPACE LIST
SET CONDITION CODES POSITIVE
A CALL TO BLOCK ?
ANY MORE TASKS IN I/P Q?
INCREMENT PROCESS EXECUTION BY 40 μS
RETURN
CHANGE PROCESS STATE TO BLOCKED IN DP
TAKE PROCESS FROM RUNNING TO BLOCKED MAP
RECORD THE VALUE TO P IN PD
START
INCREMENT PA CALLS COUNTER
FIND TASK IN I/P Q OF HIGHEST PRIORITY
TASK FOUND ?
PRINT ERROR MESSAGE
SET CALLED PROCESS EQUAL CALLING PROCESS
SET PRIORITY EQUAL N
ADD 10 μS TO PROCESS'S EXECUTION TIME
RETURN
HSELF (P) SUBROUTINE (10 μS)
START
FIND THE CPU NT OCCUPIED BY THE PROCESS
CPU FOUND
PRINT ERROR MESSAGE
BUSY THE CPU BY THE PROCESS EXECUTION TIME
INCREMENT CPU OCCUPANCY BY THE PROCESS EXECUTION TIME
INCREMENT TIME INITIATED AND TOTAL EXECUTION TIME FOR PROCESS
RETURN
BUSY CPU SUBROUTINE
START
INCREMENT PA CALLS COUNTER
ENGAGE PROLO INCREMENT ITS COUNTER AND TIME ENGAGED
FETCH TASK OF PRIORITY P FROM I/P QUEUE
TASK FOUND ?
TASK PRIORITY = P?
SEEK (P) SUBROUTINE (20 μS)

```
TRANSLATION BY #XDC5 MK 6    DATE   22/07/77


          LISTING(1)
          SOURCE(CR,S1)
          OBJECT(ED,COMMON FILEA,ABCD)
    C
          MASTER PASM
          CLASS       LOCKOUT,3(2)
          CLASS TIME PROCESS,11(10)  SET SUSPENDED,INQUEUE,11,RUNNING,BLOCKE
         /D
          CLASS TIME CPU.1(3)  SET BUSY,FREE
          CLASS       TASK.200(5)  SET FRTSKLIST
          FLOAT OCCUPANCY,SIM
          ARRAY PTBM(4,10),PPTSK(10,3),PROC(11,2)
    C     PTBM IS THE PERIODIC TASK BIT MAP
    C     PPTSK IS PERIODIC PROCESS TASK TABLE,STORES FOR EACH ACTIVATED
    C     PROCESS,I/C TASK INDEX AND ITS PRIORITY
    C     A,B,C,D STORES I/C TIX,CALLED PI,OALLED PROCESS TIX &TASK PRIORITY
    C     FOR EACH CORRESPONDING PROCESS
          HIST  SUSPWAIT.11(20,40,60)
    C     ARRAY PROC(I,J) STORES TIMES INIATED AND EXECUSION TIMES/PROCESS
    C
          COMMON/AA/LOCKOUT/BB/PROCESS,SUSPENDED,RUNNING,BLOCKED
          COMMON/CC/CPU,BUSY,FREE
          COMMON/DD/TASK,FRTSKLIST
          COMMON/EE/PROC
          COMMON/FF/BAR,J,Y,INTRIP,SUSPQINT,N,K,M,PA,TX     /GG/SUSPWAIT
          COMMON/HH/T.PROCESS,T,CPU,T.TIME,T.CLINT
          COMMON/QQ/INQUEUE
    C
    C     SUSPWAIT HISTOGRAM MEASURES THE TIME FROM A PROCESS BIENG SUSPENDE
    C     D  UNTIL IT IS SELECTED TO RUN AGAIN
    C
    C     INIALIZATION
    C     ============
          Z=1
    C     Z IS THE NO. OF CPUS IN SYSTEM
          CYCLE=0
          INTRIP=0
          LOAD FREE,FRTSKLIST,BLOCKED
          FOR I = FRTSKLIST
            FOR L =1,5
              TASK.I(L)=0
            DUMMY
          FOR I =BLOCKED
            FOR L =1,10
              PROCESS.I(L)=0
            DUMMY
          FOR I =FREE
            T.CPU.I=-1
            FOR L =1,3
              CPU.I(L)=0
            DUMMY
          FOR W = BLOCKED
            PROCESS.W(8) =4
          DUMMY
    C
          DATA PTBM/0,1,0,1,1,0,1,0,1,0,1,0,0,1,4*0,1,0,1,0,1,2*0,1,2*0,1,0,
         K1,2*0,1,4*0,1,0/
          DATA PPTSK/1,2,3,4,5,6,7,8,9,10,10*0,6,10,7,4,9,8,7,6,4,5/
          ORLEV=0
    C     ORDINARY LEVELS INTERRUPT
```

```
      EXTKLV=0
C     EXTERNAL TASK LEVELS INT.
      SIMTIME=44000
      SIM=44000.0
C     STORES SIMULATED TIME IN SECS
      INT=0
      CIX=0
      POINTER=1
      WRITE(2,6)
    6 FORMAT(1H1,20X,38HSIMULATION OF MK 2BL PROCESS ALLOCATOR/1H0,10X,3
     /8H======================================///)
      WRITE(2,7)SIMTIME,Z
    7 FORMAT(1H0,16HSIMULATED TIME =,I8,2X,12H(MICRO-SECS)///1H0,13HNO.
     /OF CPUS =,I4,2X///1H0,51HLANGUAGE USED:CONTROL AND SIMULATION LANG
     /UAGE:[CSL]////)
C
      TRIP=0
C     FLAG TO INDICATE STATE OF INT. TRIPLICATES
      T.TIME=SIMTIME
C     SIMULATED TIME
      T.CLINT=10000
C     FIRST CLOCK INTERRUPT AFTER 10 MSECS
      NOCPU=0
      OCCUPANCY=0.00
C     COUNTS UNSUCCESSFUL ATTEMPTS TO FIND A FREE CPU
      WRITE(2,4000)
 4000 FORMAT(1H0,3HOK1)
      FOR S =1,4
        FOR I =1,10
          CHECK PTBM(S,I)
      PRISET(2) FREE,FRTSKLIST,BLOCKED
      WRITE(2,5000)
 5000 FORMAT(1H0,3HOK2)
C
      ACTIVITIES
C     ==========
C
      BEGIN CLOCK INTERRUPT
C     >>>>>>>>>><<<<<<<<<<
      T.TIME GT 0
      T.CLINT EQ 0
      FOR I =1,11
        PROCESS.I(6)=0
        PROCESS.I(7)=0
        PROCESS.I(8)=4
        PROCESS.I(10)=0
      DUMMY
C     RESET ATTRIBUTES FOR NEXT CLOOK INT
      CIX=1
      INT+1
      T.CLINT=10000
      PA+1
C
      BEGIN CLOCK INT. SERVED
C     >>>>>>>>>>>>>><<<<<<<<<<<
      CIX EQ 1  6140613
  613 CYCLE EQ 1  6140
  614 T.TIME GT 0
      CYCLE EQ 1   30615
  615 ORLEV=1
      ORLEV=1
C     SET ORDINARY LEVEL INTERRUPT
      WRITE(2,6000)
 6000 FORMAT(1H0,3HOK3)
      CIX=0
    3 FREE EMPTY   92
```

```
      FIND I BUSY MIN(CPU.I(1))
      INTRIP=I
      GO TO 15
    2 FIND Y FREE FIRST
      CPU.Y FROM FREE
      CPU.Y INTO BUSY
      BAR=2
C     TO SHOW THAT AFREE CPU IS SEIZED
      GO TO 13
   15 NOCPU+1
      SUSPOINT EQ 1
      X=CPU.INTRIP(1)
C     SELECT LCPU STORED IN INTRIP ,CURP OF LCPU IN CPU.X(1)
      PROCESS.X(3)=2
C     SET CURP OF LCPU SUSP(INT)
      PROCESS.X(10) = 1
C     TO INDICATE LATOR THAT PROCESS WAS SUSP(INT)
      BAR = 3
C     TO SHOW THAT A BUSY CPU IS INTERRUPTED
      LOCKOUT.2(1)+1
C     ENGAUGE SUSPLO
      LOCKOUT.2(2)+12
C     STORE TIME SUSPLO WAS USED
      PROCESS.X FROM RUNNING
      PROCESS.X INTO SUSPENDED
   13 TRIP =1
      PRISET(2) SUSPENDED
C     ENGAGE INT. TRIPLICATES L.O.
      ORLEV EQ 1    @35
C     TEST IF ORDINAY LEVELS ARE TRIGERRED
      FOR I=1,10
        PTBM(POINTER,I) EQ 1  @30
C     CHECK IF THE PERIODIC PROCESS IS TO BE ACTIVATED
        LOCKOUT.1(1)+1
        LOCKOUT.1(2)+12
        FIND L FRTSKLIST  FIRST     @30
        FOR X =1,3
          TASK.L(X) =PPTSK(I,X)
C     COPY TASK DETAILS OF ACTIVATED PROCESS FROM PPTSK
        TASK.L FROM FRTSKLIST
        TASK.L INTO INQUEUE,11
C     INSERT TASK IN INTIM QUEUE
   30   DUMMY
      FOR I =1,10
        CHECK PTBM(POINTER,I)
      PRISET(2) INQUEUE,11,FRTSKLIST
      POINTER EQ 4   @35
      POINTER = 1
      GO TO 40
   35 POINTER+1
C     SET POINTER FOR NEXT CLOCK INTERRUPT
   40 FIND J FRTSKLIST FIRST   @1000
      LOCKOUT.1(1)+1
      LOCKOUT.1(2)+12
      TASK.J(1) = 11
      TASK.J(3) =0
C     THIS IS THE TASK FOR INTIM WITH HIGHEST PRIORITY
      GO TO  133
 1000 WRITE(2,50)
   50 FORMAT(1H0,14H NO FREE TASKS)
      EXIT
C     WHEN FRTSKLIST EXPIRES PROGRAM HALTS
  133 DUMMY
      M=11
C    TASK HANDED TO INTIM
      TASK.J FROM FRTSKLIST
      FOR L =1,11
        CHECK PROCESS.L(8)
      CALL EHNDTSK
      PRISET(2) SUSPENDED
C     CALL HAND TASK AND SET STATE SUBROUTINE
      ORLEV=0
    5 CALL ECPUSCHED
      SUSPQINT = 0
      CALL ESYSCHED
      IF(PROCESS.N(1)) 130,130,135
C     PREVIOUS STATE SUSP(UNBL) OR SUSP(INT)?
  130 LOCKOUT.3(1)+1
      LOCKOUT.3(2)+20
      FIND I INQUEUE.N MIN(TASK,I(3))    @135
      FOR L=1,5
        PROCESS.N(L)=TASK,I(L)
C     COPY TASK DETAILS INTO P.D.
        TASK,I(L)=0
C     CLEAR TASK BLOCK
      LOCKOUT.1(1)+1
      LOCKOUT.1(2)+13
      TASK.I FROM INQUEUE.N
      TASK.I INTO FRTSKLIST
      PROCESS.N(7)=1
C     SET CONDITION CODES POSITIVE
  135 SUSPENDED EMPTY   @3
      DUMMY
C
      BEGIN INITIN RUNNING
C     >>>>>>>>><<<<<<<<<
C
      PRISET(2) SUSPENDED,INQUEUE.11
      PROCESS.11(8) EQ 1
      CYCLE=0
      M=11
      PROCESS.M(6)=40
CRECORD TIME SPENT IN MAIN PROGRAM
      PROC(11,1)+1
      PROC(11,2)+40
      LOCKOUT.3(1)+1
      LOCKOUT.3(2)+20
  159 FIND J INQUEUE.11  MIN(TASK.J(3))
      TASK.J FROM INQUEUE.11
      CALL EHNDTSK
      INQUEUE.11 EMPTY  @159
C    ANY MORE TASKS? IF SO CALL EHNDTSK
      N=15
C     SET PARAMETERS FOR EBLOCK(P)
      CALL EBLOCK(P)
      PROC(11,2)+PROCESS,M(6)
      PRISET(2) SUSPENDED
      FIND I BUSY FIRST 163@165
        CPU.I(1) EQ 11
  163 CPU.I(2)+PROCESS.M(6)
      T.CPU.I=PROCESS.M(6)+70
      GO TO 166
  165 WRITE(2,400)
  400 FORMAT(1H0,18HBUSY CPU NOT FOUND)
  166 SUSPENDED EMPTY 300@167
C     ANY PROCESS SUSPENDED?
  167 FREE EMPTY  300@168
  168 CYCLE=1
      RECYCLE CLOCK INT. SERVED
  300 DUMMY
C
      BEGIN PROCESS1 RUNNING
```

```
C     >>>>>>>>>>>>>>>>>>>>>>
      PRISET(2) RUNNING
      FOR I = RUNNING
        CHECK PROCESS.I(8)
      PROCESS.1(8) EQ 1
      PROCESS.1(6)=100
      M=1
      N=15
  171 CALL EBLOCK(P)
      INQUEUE.1 EMPTY  @171
      CALL EBUSYCPU
      DUMMY
C
C
      BEGIN PROCESS2 RUNNING.
C     >>>>>>>>>>><<<<<<<<<<<
      PROCESS.2(3) EQ 1
      WRITE(2,7000)
 7000 FORMAT(1H0,3HOK4)
      PROCESS.2(6)=300
C     THIS IS PROCESSING TIME OF PROCESS2
      TX=3
      M=2
      CALL EHAND(TP)
      TX=2
      CALL EHAND(TP)
      N=6
      CALL EFETCH(P)
      TX=1
      CALL EHAND(TP)
      N=15
  190 CALL EBLOCK(P)
      INQUEUE.2 EMPTY  @190
      CALL EBUSYCPU
      DUMMY
C
      BEGIN PROCESS3 RUNNING
      PROCESS.3(3) EQ 1
      PROCESS.3(6)=300
      TX=3
      M=3
      CALL EHAND(TP)
      TX=4
      CALL EHAND(TP)
      TX=2
      CALL EHAND(TP)
      TX=1
      CALL EHAND(TP)
      N=15
  200 CALL EBLOCK(P)
      INQUEUE.3  EMPTY   @200
      CALL EBUSYCPU
      DUMMY
C
      BEGIN PROCESS6 RUNNING
C     >>>>>>>>>>><<<<<<<<<<<
      PROCESS.6(3) EQ 1
      PROCESS.6(6)=350
      M=6
      N=6
      CALL ESEEK(P)
      TX=2
      CALL EHAND(TP)
      TX=1
      CALL EHAND(TP)
      N=15
```

```
    202 CALL £BLOCK(P)
        INQUEUE.6 EMPTY  @202
        CALL £BUSYCPU
        DUMMY
C
        BEGIN PROCESS4 RUNNING
C       >>>>>>>>>>>>>>>>>>>>
        PROCESS.4(3) EQ 1
        PROCESS.4(6)=100
        M=4
        N=10
        CALL £HSELF(P)
        N=15
    205 CALL £BLOCK(P)
        INQUEUE.4 EMPTY  @205
        CALL £BUSYCPU
        DUMMY
C
C
        BEGIN PROCESS5 RUNNING
C       >>>>>>>>>><<<<<<<<<<
        PROCESS.5(3) EQ 1
        PROCESS.5(6)=200
        M=5
        N=15
    310 CALL £BLOCK(P)
        INQUEUE.5 EMPTY  @310
        CALL £BUSYCPU
        DUMMY
C
        BEGIN PROCESS7 RUNNING
C       >>>>>>>>>><<<<<<<<<<
        PROCESS.7(3) EQ 1
        PROCESS.7(6)=200
        M=7
        N=2
        CALL £HSELF(P)
        N=15
    330 CALL £BLOCK(P)
        INQUEUE.7 EMPTY  @330
        CALL £BUSYCPU
        DUMMY
C
        BEGIN PROCESS8 RUNNING
C       >>>>>>>>>><<<<<<<<<<
        PROCESS.8(3) EQ 1
        PROCESS.8(6)=150
        TX=2
        M=8
        CALL £HAND(TP)
        TX=1
        CALL £HAND(TP)
        N=15
        CHECK PROCESS.3(8)
        PRISET(2) INQUEUE.8
    350 CALL £BLOCK(P)
        INQUEUE.8 EMPTY  @350
        CHECK PROCESS.3(8),PROCESS.8(9)
        PRISET(2) INQUEUE.8
        CALL  £BUSYCPU
        DUMMY
C
        BEGIN PROCESS9 RUNNING
C      >>>>>>>>>><<<<<<<<<<
        PROCESS.9(8) EQ 1
        PROCESS.9(6)=200
```

```
      M=9
      N=2
      CALL EHSELF(P)
      N=6
      CALL EFETCH(P)
      N=15
  370 CALL EBLOCK(P)
      INQUEUE.9 EMPTY  @370
      CALL EBUSYCPU
      DUMMY
C
      BEGIN PROCESS10 RUNNING
C     >>>>>>>>>>>><<<<<<<<<<
      PROCESS.10(3) EQ 1
      PROCESS.10(6)=500
      M=10
      N=15
  380 CALL EBLOCK(P)
      INQUEUE.10 EMPTY  @380
      CALL EBUSYCPU
       FOR I =1,11
         T.PROCESS.I EQ 0  381@382
  381 DUMMY
  382 DUMMY
C
      BEGIN CPU FINISHES PROCESSING
C     >>>>>>>>>>>>>>><<<<<<<<<<<<<<
      PRISET(2) BUSY,BLOCKED,FREE
      CYCLE=0
      FOR I = BUSY
        CHECK T.CPU.I,CPU.I(1),CPU.I(2)
      FOR I=BUSY
        T.CPU.I EQ 0  @390
        CPU.I FROM BUSY
        CPU.I HEAD FREE
  390   DUMMY
      PRISET(2) BUSY,FREE
      FREE EMPTY  391@392
  392 SUSPENDED EMPTY  391@393
  393 CYCLE=1
      RECYCLE CLOCK INT. SERVED
  391 DUMMY
C
      BEGIN SIMULATION ENDS
C     >>>>>>>>>>><<<<<<<<<<
      T.TIME EQ 0     @500
      WRITE(2,8000)
 8000 FORMAT(1H0,3HOK5)
      WRITE(2,714)
  714 FORMAT(1H1,20X,38HSIMULATION OF MK 2AL PROCFSS ALLOCATOR/1H0,10X,3
     /8H======================================///)
      WRITE(2,715)SIMTIME,Z
  715 FORMAT(1H0,16HSIMULATED TIME =,I8,2X,12H(MICRO-SECS)///1H0,13HNO.
     /OF CPUS =,I4,2X///1H0,51HLANGUAGE USED:CONTROL AND SIMULATION LANG
     /UAGE:[CSL]////)
      WRITE(2,598)INT
  598 FORMAT(1H0,25HNO. OF CLOCK INTERRUPTS =,I8///)
      WRITE(2,600)
  600 FORMAT(1H0,16HPERIODIC PROCESS,5X,15HTIMES INITIATED,5X,21HEXECUSI
     /ON(MICRO-SECS))
      FOR I=1,11
        WRITE(2,602)I,PROC(I,1),PROO(I,2)
  602 FORMAT(1H0,5X,I4,10X,I8,10X,I8)
      WRITE(2,603)PA
  603 FORMAT(1H0,32HNO. OF PROCESS ALLOCATOR CALLS =,I8///)
      WRITE(2,604)LOCKOUT.1(1),LOCKOUT.1(2)
  604 FORMAT(1H0,12HLOCKOUT NAME,5X,13HTIMES ENGAGED,5X,21HEXECUSION(MIC
     /RO-SECS)/1H0,3X,6HFREELO,9X,I8,7X,I8)
C
      WRITE(2,605)LOCKOUT.2(1),LOCKOUT.2(2)
  605 FORMAT(1H0,3X,6HSUSPLO,9X,I8,7X,I8)
C
      WRITE(2,606)LOCKOUT.3(1),LOCKOUT.3(2)
  606 FORMAT(1H0,3X,5HPROLO,10X,I8,7X,I8///)
C
      WRITE(2,607)
  607 FORMAT(1H0,7HCPU NO.,7X,25HTIME OOCUPIED(MICRO-SECS),7X,21HTIME ID
     /LE(MICRO-SECS),7X,10H%OCCUPANOY)
      FOR I = 1,Z
         IDLE=SIMTIME-CPU.I(2)
         OCCUPANCY=CPU.I(2)/SIM*100.0
         WRITE(2,608)I,CPU.I(2),IDLE,OCCUPANCY
  608 FORMAT(1H0,3X,I4,15X,I8,10X,I8,10X,F8.4,1H%)
C
      WRITE(2,609)
  609 FORMAT(1H0,53HWAITING-TIME HISTOGRAM FOR PROCESS IN SUSPENDED STAT
     /E///)
      FOR I=1,11
         WRITE(2,717)I
         OUTPUT SUSPWAIT.I
  717 FORMAT(1H0,10X,14HPROCESS NUMBER,I4//)
      WRITE(2,611)
  611 FORMAT(1H0,14HTHIS IS A TRUE)
      EXIT
  500 DUMMY
      END
```

```
C
      BLOCK DATA
C     >>>>><<<<<          POSITION BEFORE MAIN PROGRAM
C     THIS SEGMENT GIVES INITIAL VALUES TO ITEMS IN COMMON BLOCKS
      ARRAY PROC(11,2),A(3,4),B(4,4),C(2,4),D(2,4)
C
      COMMON/FF/BAR,J,Y,INTRIP,SUSPQINT,N,K,M,PA,TX
      COMMON/EE/PROC,A,B,C,D
      DATA PROC/22*0/
      DATA A/1,2,3,7,8,4,0,3,0,6,4,8/
      DATA B/1,2,5,6,2,8,9,5,3,1,0,0,6,3,13,12/
      DATA C/2,3,1,3,0,4,2,10/
      DATA D/1,3,2,10,3,0,3,5/
      DATA BAR,J,Y,INTRIP,SUSPQINT,N,K,M,PA,TX/10*0/
      END
```

```
C
      SUBROUTINE £CPUSCHED
C     >>>>>>>>>><<<<<<<<<<
C     TAKES APPROX. 70 MICRO-SECS
      CLASS      LOCKOUT,3(2)
      CLASS TIME PROCESS,11(10) SET SUSPENDED,INQUEUE,11,RUNNING
      CLASS TIME CPU,1(3)  SET BUSY,FREE
      HIST  SUSPWAIT,11(20,40,60)
C
      COMMON/AA/LOCKOUT/BB/PROCESS,SUSPENDED,RUNNING
      COMMON/CC/CPU,BUSY,FREE
      COMMON/FF/BAR,J,Y,INTRIP,SUSPQINT,N,K,M/GG/SUSPWAIT
      COMMON/HH/T.PROCESS,T.CPU,T.TIME,T.CLINT
      COMMON/QQ/INQUEUE
C
      LOCKOUT.2(1)+1
      LOCKOUT.2(2)+57
      FIND L SUSPENDED MIN(11-L)    @50
C     SEARCH FOR THE HIGHEST PRIORITY SUSPENDED PROCESS
      PROCESS.L FROM SUSPENDED
      PROCESS.L INTO RUNNING
      PROCESS.L(3)=1
C     SET STATE WORD IN P.D. TO RUNNING
      ADD -T.PROCESS.L,SUSPWAIT.L
C     ADD TIME PROCESS SUSPENDED TO HIST SUSPWAIT
      N=L
      BAR EQ 2  90@95
   90 CPU.Y(1)=L
      T.CPU.Y    EQ 0  91@92
   91 DUMMY
   92 DUMMY
      GO TO 100
   95 CPU.INTRIP(1)=L
      FIND I BUSY MIN(CPU.I(1))   40@70
C     FIND NEW VALUE OF LCPU
   70 WRITE(2,80)
   80 FORMAT(1H0,27HNEW VALUE OF LCPU NOT FOUND)
      GO TO 100
   40 INTRIP=I
      GO TO 100
C     SEND NEW VALUE OF LCPU TO INT. TRIP. LO
   50 WRITE(2,60)
   60 FORMAT(1H0,33HNO SUSP PROCESS FOUND IN CPUSCHED)
      T.TIME  GE  0   @100
      T.CLINT  GE  0  120@130
  120 DUMMY
  130 DUMMY
  100 WRITE(2,140)
  140 FORMAT(1H0,17H£CPUSCHED ENTERED)
      RETURN
C  TIME SPENT IN £CPUSCHED IS ADDED IN £BUSYCPU
      END
```

```
C
C       SUBROUTINE SYSCHED:TAKES APPROX. 50 MICRO=SECS
C       >>>>>>>>>>>>>>>>>>
        SUBROUTINE ESYSCHED
        CLASS  PROCESS.11(10)  SET SUSPENDED
        CLASS       CPU.1(3)  SET BUSY,FREE
C
        COMMON/BB/PROCESS,SUSPENDED/CC/CPU,BUSY,FREE/FF/BAR,J,INTRIP,
       /SUSPOINT
C
        FREE EMPTY    @50
        FIND I SUSPENDED MIN(11-I)     @40
C       SEARCH FOR HIGHEST PRIORITY SUSPENDED PROCESS
        GO TO  70
     40 WRITE(2,60)
     60 FORMAT(1H0,32HNO SUSP PROCESS FOUND IN SYSCHED)
        GO TO 50
C
     70 I GT  CPU.INTRIP(1)     @50
C
C       SUSPENDED PROCESS HIGHER THAN LCPU CURP?
C
        SUSPOINT=1
C       SET THE SUSPENDED QUEUE INTERRUPT
     50 WRITE(2,80)
     80 FORMAT(1H0,16HESYSCHED ENTERED)
        RETURN
        END
```

```
C
C
          SUBROUTINE EHAND(TP)
C         IT HANDS ATASK TO THE CALLED PROCESS
C         TAKES APPROX. 60 MICRO-SECS
          CLASS  LOCKOUT.3(2)
          CLASS  TASK.200(5)  SET FRTSKLIST
          CLASS  PROCESS.11(10)  SET SUSPENBED,INQUEUE.11,RUNNING
          ARRAY PROC(11,2),A(3,4),B(4,4),C(2,4),D(2,4)
C
          COMMON/AA/LOCKOUT/BB/PROCESS,SUSPENDED,RUNNING
          COMMON/EE/PROC,A,B,C,D
          COMMON/FF/BAR,J,Y,INTRIP,SUSPQINT,N,K,M,PA,TX
          COMMON/DD/TASK,FRTSKLIST
          COMMON/QQ/INQUEUE
C
          PA+1
C         RECORD CALLING OF PA
          LOCKOUT.1(1)+1
C         ENGAGE  FREELO
          LOCKOUT.1(2)+13
C         FREELO TAKES 13 MICRO-SECS
          FIND I FRTSKLIST FIRST     @600
          TASK.I FROM FRTSKLIST
C         FETCH AFREE TASK BLOCK
          M EQ 2   @ 100
          FOR L=2,4
            TASK.I(L-1) = A(TX,L)
      100 M EQ 3   @ 200
C         IS CALLING PROCESS INDEX =3?
          FOR L=2,4
            TASK.I(L-1)= B(TX,L)
      200 M EQ 6   @ 300
          FOR L=2,4
            TASK.I(L-1)=C(TX,L)
      300 M EQ 8   @400
          FOR L=2,4
            TASK.I(L-1)=D(TX,L)
      400 DUMMY
C
C         STORE TASK DETAILS FROM APPROPRIATE PROCESS ARRAY
          P=TASK.I(1)
C         TX IS CALLING PROCESS TASK INDEX
C         M IS CALLING PROCESS INDEX
          TASK.I INTO INQUEUE.P
C         LINK TASK BLOCK IN PROCESS INPUT Q
          LOCKOUT.3(1)+1
          LOCKOUT.3(2)+25
          PROCESS.M(6)+60
C         RECORD TIME SPENT
          GO TO 700
      600 WRITE(2,7)
C
        7 FORMAT(1H0,17HNO FREE TASK LEFT)
      700 WRITE(2,800)
      800 FORMAT(1H0,17HEHAND(TP) ENTERED)
          RETURN
          END
```

```
C
C
      SUBROUTINE £BLOCK(P)
C     >>>>>>>>>>>>>>>>>>>>
C     TAKES APPROX 40 MICRO-SECS
C
      CLASS LOCKOUT.3(2)
      CLASS PROCESS.11(10) SET SUSPENDED,INQUEUE,11,RUNNING,BLOCKED
      CLASS TASK.200(5)  SET FRTSKLIST
C
      COMMON/AA/LOCKOUT/BB/PROCESS,SUSPENDED,RUNNING,BLOCKED
      COMMON/FF/BAR,J,Y,INTRIP,SUSPQINT,N,K,M,PA
      COMMON/DD/TASK,FRTSKLIST
      COMMON/QQ/INQUEUE
C
      PA+1
      LOCKOUT.3(1)+1
      LOCKOUT.3(2)+20
      FIND I INQUEUE.M  MIN(TASK.I(3))      @ 20
C     GET FIRST TASK IN Q  OF HIGHEST PRIORITY
      TASK.I(3) LE N   @ 20
C     TASK IN I/P Q WITH PRIORITY > OR = N?
C     M IS PROCESS INDEX CALLING £BLOCK(N)
C     N IS PRIORITY OF REQUIRED TASK
      FOR X=1,3
        PROCESS.M(X)= TASK.I(X)
C     LOAD TASK IN PROCESS DESCRIPTOR
      LOCKOUT.1(1)+1
      LOCKOUT.1(2)+13
      TASK.I FROM INQUEUE.M
      TASK.I INTO FRTSKLIST
C     LINK TSK BLOCK BACK INTO FREE SPACE LIST
C
      PROCESS.M(7) = 1
C     SET CONDITION CODES POSITIVE
      INQUEUE.M  EMPTY    20@30
   20 PROCESS.M(3)= 4
C     SET STATE TO BLOCKED IN P.D.
      PROCESS.M  FROM RUNNING
      PROCESS.M  INTO BLOCKED
      PROCESS.M(9)= N
C     RECORD VALUE OF N IN BLOCKED(N) IN P.D.
   30 PROCESS.M(6)+40
C     STORE TIME SPENT IN THE CALL
      WRITE(2,40)
   40 FORMAT(1H0,17H£BLOCK(P) ENTERED)
      RETURN
      END
```

```
C
C
      SUBROUTINE £FETCH(P)
C     >>>>>>>>>>>>>>>>>
C     TAKES APPROX 20 MICRO-SECS
      CLASS LOCKOUT,3(2)
      CLASS PROCESS,11(10) SET  SUSPENDED,INQUEUE,11
      CLASS TASK,200(5)  SET  FRTSKLIST
C
      COMMON/AA/LOCKOUT/BB/PROCESS,SUSPENDED
      COMMON/DD/TASK,FRTSKLIST
      COMMON/FF/BAR,J,Y,INTRIP,SUSPQINT,N,K,M,PA
      COMMON/QQ/INQUEUE
C
C
      PA+1
      LOCKOUT.3(1)+1
      LOCKOUT.3(2)+20
      FIND I INQUEUE.M  MIN(TASK.I(3))      @ 20
C     GET FIRST TASK IN Q  OF HIGHEST PRIORITY
      TASK.I(3) LE N   @ 20
C     TASK IN I/P Q  WITH PRIORITY <OR= N?
C     M IS THE PROCESS INDEX CALLING £BLOCK(N)
C     N IS STATED BEFORE THE CALL IN THE CALLING PROCESS
      FOR X=1,3
        PROCESS.M(X)=TASK.I(X)
C     LOAD TASK IN P.D.
      LOCKOUT.1(1)+1
      LOCKOUT.1(2)+13
      TASK.I FROM INQUEUE.M
      TASK.I INTO FRTSKLIST
C     LINK TASK BLOCK BACK INTO FREE SPACE LIST
      PROCESS.M(7)=1
C     SET CONDITION CODES POSITIVE
      GO TO 30
   20 PROCESS.M(9)=N
C     STORES VALUE OF REQUIRED PRIORITY
      PROCESS.M(7) = 0
C     SET CONDITION CODES ZERO
   30 PROCESS.M(6)+20
      WRITE(2,40)
   40 FORMAT(1H0,17HEFETCH(P) ENTERED)
      RETURN
      END
```

```
C
C
      SUBROUTINE ESEEK(P)
C     >>>>>>>>>>>>>>>>>>>
C     TAKES APPROX. 20 MICRO-SECS
      CLASS LOCKOUT.3(2)
      CLASS PROCESS.11(10) SET  SUSPENDED,INQUEUE.11
      CLASS TASK.200(5)  SET  FRTSKLIST
C
      COMMON/AA/LOCKOUT/BB/PROCESS,SUSPENDED
      COMMON/DD/TASK,FRTSKLIST
      COMMON/FF/BAR,J,Y,INTRIP,SUSPQINT,N,K,M,PA
      COMMON/QQ/INQUEUE
C
      PA+1
      LOCKOUT.3(1)+1
      LOCKOUT.3(2)+20
      FIND I INQUEUE.M  MIN(TASK.I(3))      @ 40
C     GET FIRST TASK IN Q  OF HIGHEST PRIORITY
      TASK.I(3) EQ N   @ 20
C     TASK IN I/P Q WITH PRIORITY EQUAL N?
C     M IS THE PROCESS INDEX CALLING ESEEK(N)
C     N IS STATED BEFORE THE CALL IN THE CALLING PROCESS
      FOR X=1,3
        PROCESS.I(X)=TASK.I(X)
C     LOAD TASK IN P.D.
      LOCKOUT.1(1)+1
      LOCKOUT.1(2)+13
      TASK.I FROM INQUEUE.M
      TASK.I INTO  FRTSKLIST
C     LINK TASK BLOCK BACK TO FRTSKLIST
      PROCESS.M(7)=1
C     SET CONDITION CODES POSITIVE
      GO TO 30
   20 PROCESS.M(9) = N
C     STORE VALUE OF N IN SEEK(N)
      PROCESS.M(7) = 0
C     SET CONDITION CODES ZERO
   40 WRITE(2,50)
   50 FORMAT(1H0,16HNO TASK IN I/P Q)
   30 PROCESS.M(5)+20
      WRITE(2,60)
   60 FORMAT(1H0,16HESEEK(P) ENTERED)
      RETURN
      END
```

```
C
C
      SUBROUTINE EHSELF(P)
C     >>>>>>>>>>>>>>>>>>>>   TAKES APPROX. 10 MICRO-SECS
      CLASS PROCESS.11(10)  SET  SUSPENDED,INQUEUE .,11
      CLASS TASK.200(5)
C
      COMMON/BB/PROCESS,SUSPENDED
      COMMON/DD/TASK,FRTSKLIST
      COMMON/FF/BAR,J,Y,INTRIP,SUSPQINT,N,K,M,PA
      COMMON/QQ/INQUEUE
C
      PA+1
      FIND I INQUEUE.M  MIN(TASK.I(3))      @ 40
C     GET FIRST TASK IN Q  OF HIGHEST PRIORITY
      TASK.I(1) = M
C     SET CALLED PROCESS EQUAL CALLING PROCESS
      TASK.I(3)= N
C     SET PRIORITY EQUAL N
      GO TO 60
   40 WRITE(2,50)
   50 FORMAT(1H0,16HNO TASK IN I/P Q)
   60 PROCESS.M(5)+10
C     RECORD TIME
      WRITE(2,70)
   70 FORMAT(1H0,17HEHSELF(P) ENTERED)
      RETURN
      END
      SUBROUTINE EHNDTSK
C     >>>>>>>>><<<<<<<<<
C     SUBROUTINE HANDTASK ANDSET STATE,TAKES APPROX. 40 MICRO-SECS
      CLASS        LOCKOUT,3(2)
      CLASS TIME PROCESS,11(10) SET SUSPENDED,INQUEUE,11,RUNNING,BLOCKED
      CLASS TIME CPU,1(3)  SET BUSY,FREE
      CLASS        TASK,200(5) SET FRTSKLIST
C
      COMMON/AA/LOCKOUT/BB/PROCESS,SUSPENDED,RUNNING,BLOCKED
      COMMON/CC/CPU,BUSY,FREE
      COMMON/DD/TASK,FRTSKLIST
      COMMON/FF/BAR,J,Y,INTRIP,SUSPQINT,N,K,M,PA,TX
      COMMON/HH/T.PROCESS,T,CPU,T.TIME,T.CLINT
      COMMON/QQ/INQUEUE
C
      LOCKOUT.3(1)+1
      LOCKOUT.3(2)+25
      W=TASK.J(1)
      TASK,J INTO INQUEUE.W
C     LINK TASK BLOCK IN PROCESS I/P Q
      PRISET(2)  INQUEUE,11
      TASK,J(1) LT 64    @60
C
C
C     TO TEST FOR APEREPHRAL PROCESS,TEST WHETHER PI>64
      FOR L =1,11
        CHECK PROCESS.L(8)
      PROCESS.W(8)  EQ  4    @100
C     CALLED PROCESS STATE BLOCKED(N) FOR SOME N?
      LOCKOUT.3(1)+1
      LOCKOUT.3(2)+25
      FIND I INQUEUE.W  MIN(TASK.I(3))    @120
      PROCESS.W(9)  EQ  0  40@50
   50 PROCESS.W(9)  GE  TASK,I(3)   @140
C     PRIORITY OF HANDED TASK AT LEAST N?
   40 PROCESS.W(3)=3
C     CHANGE STATE WORD IN P.D. TO SUSP(UNBL)
      T.PROCESS.W=0
C     START SUSPTIME COUNT
      LOCKOUT.2(1)+1
      LOCKOUT.2(2)+14
      PROCESS.W  FROM  BLOCKED
      PROCESS.W  INTO  SUSPENDED
C     INCLUDE PROCESS IN SUSPENDED STATE MAP
      GO TO 80
   60 WRITE(2,90)
   90 FORMAT(1H0,8HPI GE 64)
      GO TO 80
  100 WRITE(2,110)
  110 FORMAT(1H0,17HPROCESS,W(8) NE 4)
      GO TO 80
  120 WRITE(2,130)
  130 FORMAT(1H0,16HNO TASK IN INQ,W)
      GO TO 80
  140 WRITE(2,150)
  150 FORMAT(1H0,25HPROCESS,W(9) LT TASK.I(3))
      GO TO 80
   80 PROCESS.M(6)+40
C    RECORD TIME SPENT
      WRITE(2,70)
   70 FORMAT(1H0,15HEHNDTSK ENTERED)
      FOR I =FREE
        T.CPU.I EQ 0  160@170
  160 DUMMY
```

```
170 DUMMY
    T.TIME GE 0  @180
    T.CLINT  GE  0  180@190
180 DUMMY
190 DUMMY
    RETURN
    END
```

```
C
C
      SUBROUTINE ƐBUSYCPU
C     >>>>>>>>><<<<<<<<<<
C
C     IT BUSIES CPU ON WHICH PROCESS RUNNING
C     BY TOTAL RUNNING TIME OF THE PROCESS
C
      CLASS TIME PROCESS,11(10)
      CLASS TIME CPU,1(3)  SET BUSY,FREE
      ARRAY PROC(11,2)
C
      COMMON/BB/PROCESS/CC/CPU,BUSY,FREE
      COMMON/EE/PROC
      COMMON/FF/BAR,J,Y,INTRIP,SUSPQINT,N,K,M,PA,TX
      COMMON/HH/T.PROCESS,T,CPU,T.TIME,T.CLINT
C
      FIND  I  BUSY FIRST  2000 @ 2500
            CPU.I(1) EQ M
 2000 T.CPU.I=PROCESS.M(6)+70
C   70 IS THE TIME SPENT IN ƐCPUSCHED
      CPU.I(2)+PROCESS.M(6)
C     STORES TIME PROCESS OCCUPIED THIS CPU
C
      PROC(M,1)+1
      PROC(M,2)+PROCESS.M(6)
      GO TO 50
 2500 WRITE(2,3000)
 3000 FORMAT(1H0,18HBUSY CPU NOT FOUND)
      FOR L =1,11
        T.PROCESS.L EQ 0  60@70
   60 DUMMY
   70 DUMMY
      T.TIME  GE  0  @80
      T.CLINT  GE  0   90@80
   80 DUMMY
   90 DUMMY
   50 WRITE(2,100)
  100 FORMAT(1H0,16HƐBUSYCPU ENTERED)
      RETURN
      END
```

```
          SIMULATION OF MK 2BL PROCESS ALLOCATOR
          ======================================


SCHOOL OF ELECTRICAL ENGINEERING-PLYMOUTH POLYTECHNIC-A.M.SALIH-JULY,1977


SIMULATED TIME =  1000000  (MICRO-SECS)


NO. OF CPUS =   2


LANGUAGE USED:CONTROL AND SIMULATION LANGUAGE:[CSL]



NO. OF CLOCK INTERRUPTS =       99


                                         T
PERIODIC PROCESS     TIMES INITIATED     EXECUSION(MICRO-SECS)
       1                   49             6860
       2                   50            27000
       3                   50            29000
       4                   25             4710
       5                   25             6960
       6                   50            26500
       7                   25             7210
       8                   50            17460
       9                   25             6750
      10                   25            14460
      11                   99            26840
NO. OF PROCESS ALLOCATOR CALLS =     1467


                                    T
LOCKOUT NAME      TIMES ENGAGED     EXECUSION(MICRO-SECS)
   FREELO              2038         26021
   SUSPLO               946         33583
   PROLO               2811         63700
```

```
CPU NO.        TOTAL TIME        TIME OCCUPIED        TIME IDLE        %OCCUPANCY

    1            1000000             86020              913980           8.6020%

    2            1000000             83770              916230           8.3770%

WAITING-TIME HISTOGRAM FOR PROCESS IN SUSPENDED STATE


          PROCESS NUMBER    1

COUNT           RANGE
   24              TO       10
    0         11   TO      110
    0        111   TO      210
    0        211   TO      310
    0        311   TO      410
    0        411   TO      510
   25        511   TO      610
    0        611   TO      710
    0        711   TO      810
    0        811   TO      910
    0        911   TO     1010
    0       1011   TO     1110
    0       1111   TO     1210
    0       1211   TO     1310
    0       1311   TO     1410
    0       1411   TO     1510
    0       1511   TO     1610
    0       1611   TO     1710
    0       1711   TO     1810
    0       1811   TO

          PROCESS NUMBER    2

COUNT           RANGE
    0              TO       10
    0         11   TO      110
    0        111   TO      210
    0        211   TO      310
    0        311   TO      410
    0        411   TO      510
    0        511   TO      610
    0        611   TO      710
    0        711   TO      810
   25        811   TO      910
    0        911   TO     1010
    0       1011   TO     1110
    1       1111   TO     1210
   24       1211   TO     1310
    0       1311   TO     1410
    0       1411   TO     1510
    0       1511   TO     1610
    0       1611   TO     1710
    0       1711   TO     1810
    0       1811   TO
```

```
                    PROCESS NUMBER    3

COUNT                 RANGE
     0                  TO        10
     0            11    TO       110
     0           111    TO       210
     0           211    TO       310
     1           311    TO       410
    24           411    TO       510
     0           511    TO       610
     0           611    TO       710
     0           711    TO       810
     0           811    TO       910
     0           911    TO      1010
     0          1011    TO      1110
    25          1111    TO      1210
     0          1211    TO      1310
     0          1311    TO      1410
     0          1411    TO      1510
     0          1511    TO      1610
     0          1611    TO      1710
     0          1711    TO      1810
     0          1811    TO

                    PROCESS NUMBER    4

COUNT                 RANGE
     0                  TO        10
     0            11    TO       110
     0           111    TO       210
     0           211    TO       310
    25           311    TO       410
     0           411    TO       510
     0           511    TO       610
     0           611    TO       710
     0           711    TO       810
     0           811    TO       910
     0           911    TO      1010
     0          1011    TO      1110
     0          1111    TO      1210
     0          1211    TO      1310
     0          1311    TO      1410
     0          1411    TO      1510
     0          1511    TO      1610
     0          1611    TO      1710
     0          1711    TO      1810
     0          1811    TO
```

```
                  PROCESS NUMBER    5

COUNT                 RANGE
       0                TO        10
       0          11    TO       110
       0         111    TO       210
       0         211    TO       310
       0         311    TO       410
       0         411    TO       510
       0         511    TO       610
       0         611    TO       710
      25         711    TO       810
       0         811    TO       910
       0         911    TO      1010
       0        1011    TO      1110
       0        1111    TO      1210
       0        1211    TO      1310
       0        1311    TO      1410
       0        1411    TO      1510
       0        1511    TO      1610
       0        1611    TO      1710
       0        1711    TO      1810
       0        1811    TO

                  PROCESS NUMBER    6

COUNT                 RANGE
       0                TO        10
       0          11    TO       110
       0         111    TO       210
      25         211    TO       310
       0         311    TO       410
       0         411    TO       510
       1         511    TO       610
      24         611    TO       710
       0         711    TO       810
       0         811    TO       910
       0         911    TO      1010
       0        1011    TO      1110
       0        1111    TO      1210
       0        1211    TO      1310
       0        1311    TO      1410
       0        1411    TO      1510
       0        1511    TO      1610
       0        1611    TO      1710
       0        1711    TO      1810
       0        1811    TO
```

```
                PROCESS NUMBER    7

COUNT               RANGE
     0                TO        10
     0          11    TO       110
     0         111    TO       210
    25         211    TO       310
     0         311    TO       410
     0         411    TO       510
     0         511    TO       610
     0         611    TO       710
     0         711    TO       810
     0         811    TO       910
     0         911    TO      1010
     0        1011    TO      1110
     0        1111    TO      1210
     0        1211    TO      1310
     0        1311    TO      1410
     0        1411    TO      1510
     0        1511    TO      1610
     0        1611    TO      1710
     0        1711    TO      1810
     0        1811    TO

                PROCESS NUMBER    8

COUNT               RANGE
    25                TO        10
     0          11    TO       110
     0         111    TO       210
     0         211    TO       310
    25         311    TO       410
     0         411    TO       510
     0         511    TO       610
     0         611    TO       710
     0         711    TO       810
     0         811    TO       910
     0         911    TO      1010
     0        1011    TO      1110
     0        1111    TO      1210
     0        1211    TO      1310
     0        1311    TO      1410
     0        1411    TO      1510
     0        1511    TO      1610
     0        1611    TO      1710
     0        1711    TO      1810
     0        1811    TO
```

```
                 PROCESS NUMBER    9

COUNT               RANGE
   25                 TO         10
    0            11   TO        110
    0           111   TO        210
    0           211   TO        310
    0           311   TO        410
    0           411   TO        510
    0           511   TO        610
    0           611   TO        710
    0           711   TO        810
    0           811   TO        910
    0           911   TO       1010
    0          1011   TO       1110
    0          1111   TO       1210
    0          1211   TO       1310
    0          1311   TO       1410
    0          1411   TO       1510
    0          1511   TO       1610
    0          1611   TO       1710
    0          1711   TO       1810
    0          1811   TO

                 PROCESS NUMBER   10

COUNT               RANGE
   25                 TO         10
    0            11   TO        110
    0           111   TO        210
    0           211   TO        310
    0           311   TO        410
    0           411   TO        510
    0           511   TO        610
    0           611   TO        710
    0           711   TO        810
    0           811   TO        910
    0           911   TO       1010
    0          1011   TO       1110
    0          1111   TO       1210
    0          1211   TO       1310
    0          1311   TO       1410
    0          1411   TO       1510
    0          1511   TO       1610
    0          1611   TO       1710
    0          1711   TO       1810
    0          1811   TO
```

```
                    PROCESS NUMBER  11

COUNT                 RANGE
   99                  TO         10
    0          11      TO        110
    0         111      TO        210
    0         211      TO        310
    0         311      TO        410
    0         411      TO        510
    0         511      TO        610
    0         611      TO        710
    0         711      TO        810
    0         811      TO        910
    0         911      TO       1010
    0        1011      TO       1110
    0        1111      TO       1210
    0        1211      TO       1310
    0        1311      TO       1410
    0        1411      TO       1510
    0        1511      TO       1610
    0        1611      TO       1710
    0        1711      TO       1810
    0        1811      TO
```

APPENDIX B

SIMULA LISTING OF SYSTEM X SIMULATOR PACKAGE AND CROSS REFERENCE

```
BEGIN
    COMMENT********* SYSTEM X SIMULATOR PACKAGE ***********
    *                ===========================
    *
    * DESCRIPTION:
    *           COMPRISES SIMULATION MODULES OF THE
    *           PROCESS ALLOCATOR,INTIM,STORAGE ALL-
    *           OCATOR,INTERRUPT TRIPLICATES,CPUS,
    *           BACKGROUND PROCESSES,CLOCK INTERRUPTS,
    *           RASH, WITHFBLCALL, NOFBLCALL, INITIATOR
    *           PROCESS, PROCESSES FOR THE HYPOTHETICAL
    *           DIGITAL MAIN NETWORK SWITCHING CENTRE
    *           I.E. TR, LCH1, ICS, ILCR, IRSP, NR, SH,
    *           OCS, OLCR, ORSP, AND LCH2.MODELS ARE ALSO
    *           ENCLUDED OF CALL GENERATOR, CALL RECORD,
    *           INCOMING SIGNAL INTERWORKING SUBSYSTEM HARDWARE,
    *           OUTGOING SIGNAL INTERWORKING SUBSYSTEM HARD-
    *           WARE, SIGNAL INTERWORKING SUBSYSTEM, CALL
    *           PROCESSING SUBSYSTEM, DIGITAL SWITCHING SUB-
    *           SYSTEM SOFTWARE, DIGITAL SWITCHING SUBSYSTEM
    *           HARDWARE.
    *           A PARTICULAR SYSTEM X EXCHANGE SIMULATION IS
    *           OBTAINED BY INITIALISING THE APPROPRIATE PRO-
    *           CESSES INSTANCES AND ELEMENTS OF THE INPUT DATA
    *           FILE.
    *
    **********************GLOBAL VARIABLES ************
    *
    *TOTALCCTS: TOTAL NO. OF CCTS IN EXCHANGE
    *INCOMCCTS: NO. OF INCOMMING CIRCUITS
    *SIMPERIOD: SIMULATED PERIOD
    *MININCCT: THE I/C CIRCUIT WITH THE MIN. NO.
    *MINOUTCCT: " O/G    "     "    " MAX.  "
    *MAXINCCT:  " I/C    "     "    " MIN.  "
    *MAXOUTCCT: " O/G    "     "    " MAX.  "
    *SD1-SD20 : SEEDS FOR STATISTICAL DISRIBUTIONS
    * NUM: THE NUMBER OF CPUS IN THE SYSTEM
    *POINTER: TO INDEX DOWN PERIODIC PROCESSES TABLE(PPTABLE)
    *FRENG: NC. OF PAS WAITING FOR FREELO
    *FMIN: MIN NO. OF PAS WAITING FOR FREELO
    *FMAX: MAX  "   "  "    "      "    "
    *FRWAIT: TOTAL TIME SPENT BY ALL PAS WAITING FOR FREELO
    *FNUM: NO. OF TIMES FREELO ENGAGED
    *SUSENG: NO. OF PAS WAITING FOR SUSPLO
    *SMIN: MIN NO. OF PAS WAITING FOR SUSPLO
    *SMAX: MAX  "   "  "    "      "    "
    *SUSWAIT: TOTAL TIME SPENT BY ALL PAS WAITING FOR SUSPLO
    *SNUM: TIMES SUSPLO ENGAGED
    *INENG: NC. OF PAS WAITING FOR INTLO
    *IMIN: MIN NO. OF PAS WAITING FOR INTLO
    *IMAX: MAX "   "  "    "      "   "
    *INWAIT: TOTAL TIME SPENT BY ALL PAS WAITING FOR INTLO
    *INUM: TIMES INTLO ENGAGED
    *NUMOFPROCESSES: NO. OF NOFBLCALL OR WITHFBLCALL
    *                INSTANCES IN A RUN
    *HPCPS: HIGHEST PRIORITY CPS INSTANCE
    *FLAG: TO START AND STOP OUTPUTTING PROGRAM TRACE
    *FREELO: TRUE WHEN FREELO IS ENGAGED
    *SUSPLO:  "    "  SUSPLO "    "
    *INTLO:   "    "  INTLO  "    "
    *SUSPMAP: SUSPENDED PROCESSES STATE MAP
    *FREELOQ: A QUEUE WHERE PA WAITS FOR FREELO TO BE RELEASED
    *SUSPLOQ: "   "     "    "   "    "  SUSPLO "  "     "
    *INTLOQ:  "   "     "    "   "    "  INTLO  "  "     "
    *FREETASKLIST: A QUEUE OF FREE TASK BLOCKS
    *LPAQ: A QUEUE WHERE INTERRUPT TRIPLICATES  WAIT FOR
    *       PA ON LCPU TO FINISH SERVICING A PROCESS CALL
    *       BEFORE INTERRUPTING LCPU
    *INTRIP: A REF  TO INTERRUPT TRIPLICATES
    *CLINT:  "  "    "  CLOCK INTERRUPT
    *P(0:127)  REF ARRAY TO PROCESSES IN SYSTEM
    *DSSHANDLER: A REF TO DSS HANDLER
    *DIGITALSWITCH: A REF TO THE DIGITAL SWITCH H/W
    *CALLSGEN: A REF TO THE CENTRAL CALL GENERATOR PROCESS
    *STARTER: A REF TO THE INITIATOR PROCESS
    *B(1: NUM): A REF ARRAY TO BACKGROUND PROCESSES
    *C(1: NUM): A REF ARRAY TO THE CPUS
    *CALL(MININCCT: MAXOUTCCT) A REF ARRAY TO CALLS
    *                IN PROGRESS
    *SISHWINC(MININCCT: MAXOUTCCT): A REF ARRAY TO
    *                    I/C SIS H/W LINES
    *SISHWOUT(MININCCT: MAXOUTCCT): A REF ARRAY TO
    *                    O/G SIS H/W LINES
    *RESPONSE(1: 10): A REF ARRAY TO DELAYSTAT INSTANCES
    *
    ******************GLOBAL PROCEDURES***********
    *OUTTV(T,V): OUTPUTS TEXT T AND INTEGER V
    *OUTTVR(T,V):    "      "  "  "  REAL    "
    *OUTLINE(T):     "      "  " ONLY
    *ERROR(NO):      "  AN ERROR NO.
    *WRITE(T,V,I):     "    TEXT T ,TEXT V AND INTEGER I
    *OR2: AN EFFICIENT OR PROCEDURE
    *AND2:     "       AND   "
    *SISREP(CCTNUM): RETURNS WITH SIS REP NO. OF CIRCUIT WHOSE
    *                NO. IS CCTNUM
    *CPSREP(CCTNUM): RETURNS WITH CPS REP NO. OF CIRCUIT WHOSE
    *                 NO. IS CCTNUM
```

```
     *
     ************************END GLOBAL PARAMETERS******;


                             COMMENT
 *
 *    ABDUL SALIH: DEC. 1979
 *
 *             GEC MARK IIBL MULTIPROCESSOR SUBSYSTEM AND
 *             CIHER SYSTEM X SUBSYSTEMS SIMULATOR STRUCTURE
 *



                                 PROCESS
                                    I
                                    I
                  ---------------------------------
                  I                 I              I
                  I                 I              I
                 CPU               AP              OS

       ------------------------------------------------
        I            I             I          I         I
        I            I             I          I         I
      RASH          SIS           DSS        CPS  ...   SA

                        --------------------------
                        I                        I
                        I                        I
                      DMNSC                LOCAL EXCHANGE
;

INTEGER NUM,  TCTALCCTS, INCOMCCTS, MININCCT, NUMOFPROCESSES,
 MAXINCCT, MINCUTCCT, MAXOUTCCT, HPCPS, SD1, SD2, SD3, SD4, SD5, SD6,
SD7, SD8, SD9, SD10, SD11, SD12, SD13, SD14, SD15, SD16, SD17, SD18,
SD19, SD20;
REAL SIMPERIOD;
TOTALCCTS:=ININT;
INCOMCCTS:=ININT;
MININCCT := ININT;
MAXINCCT := ININT;
MINOUTCCT:= ININT;
MAXOUTCCT:= TCTALCCTS;
NUM:=ININT;
HPCPS    := 49;
SIMPERIOD := INREAL;

SIMULATION
BEGIN

  INTEGER I,J,PCINTER ,FRENG,FMIN,FMAX,SUSENG,SMIN,SMAX,FNUM,SNUM,
  INENG,IMIN,IMAX,INUM,FRWAIT,SUSWAIT,INWAIT,K;
  BOOLEAN FREEIC,SUSPLO,FLAG;
  BOOLEAN ARRAY SUSPMAP(0:127);
  REF(HEAD)FREELOQ,SUSPLOQ,FREETASKLIST,INTLOQ,LPAQ;
  REF(INTRIPLICATES) INTRIP;
  REF(INITIATOR) STARTER;
  REF(CPU) ARRAY C(1:NUM);
  REF(BACKGROUND) ARRAY B(1:NUM);
  REF(AP) ARRAY P(0:127);
  REF(CALLRECORD) ARRAY CALL(MININCCT:MAXOUTCCT);
  REF(INCSISHW) ARRAY SISHWINC(MININCCT:MAXOUTCCT);
  REF(OUTGSISHW) ARRAY SISHWOUT(MININCCT:MAXOUTCCT);
  REF(CLOCKINTERRUPT) CLINT;
  REF(DSSSW)DSSHANDLER;
  REF(CALLGENEBATOR)CALLSGEN;
  REF(DSSHW)DIGITALSWITCH;
  REF(DELAYSTAT) ARRAY RESPONSE(1:10);


  PROCEDURE OUTTV(T,V);
  COMMENT---------------;
  VALUE T;   TEXT T; INTEGER V;
  BEGIN
    OUTTEXT(T);
    OUTINT(V,10);
    OUTIMAGE;
  END*****OUTTV*****;


  PROCEDURE OUTTVR(T,V);VALUE T;TEXT T;REAL V;
  COMMENT---------------;
  BEGIN
    OUTTEXT(T);
    OUTFIX(V,2,12);
    OUTIMAGE;
  END**OUTTVR***;


  PROCEDURE OUTLINE(T);VALUE T;TEXT T;
  COMMENT------------------;
  BEGIN OUTTEXT(T);
    OUTIMAGE;
  END****OUTLINE*****;
```

```
PROCEDURE ERRCR(NO);
COMMENT---------------;
INTEGER NO;
BEGIN
  OUTTEXT("*****ERROR NO. ");
  OUTINT(NO,3);
END****ERRCR PROCEDURE****;


PROCEDURE WRITE(T,V,I);
VALUE T,V; TEXT T,V; INTEGER I;
BEGIN
  OUTTEXT(T);
  OUTTEXT(V);
  OUTINT(I,6);
  OUTIMAGE;
END****OF WRITE***;


BOOLEAN PROCEDURE AND2(A,B); BOOLEAN A,B;
AND2 := IF A THEN B ELSE FALSE;


BOOLEAN PROCEDURE OR2(A,B); BOOLEAN A,B;
OR2 := IF A THEN TRUE ELSE B;


INTEGER PROCEDURE SISREP(CCTNUM); INTEGER CCTNUM;
COMMENT**DETERMINES SIS REPLICATE ACCORDING TO CCT. NO.*****;
SISREP := IF CR2( AND2(CCTNUM >= 1, CCTNUM <= 58 ),
AND2( CCTNUM >= 102 , CCTNUM <= 581 ))  THEN 0
ELSE 1;


INTEGER PROCEDURE CPSREP(CCTNUM); INTEGER CCTNUM;
COMMENT***DETERMINES CPS REPLICATE ACCORDING TO CCTNUM***;
BEGIN
  CPSREP := IF AND2(CCTNUM >= 1 , CCTNUM <= 65 ) THEN 0
  ELSE
  IF AND2(CCTNUM >= 66 , CCTNUM <= 101 ) THEN 1
  ELSE
  IF  AND2(CCTNUM >=102 , CCTNUM <= 581 ) THEN 2
  ELSE
  IF AND2(CCTNUM >=582 , CCTNUM <=1061 ) THEN 3
  ELSE
  999;
END****CPSREP****;



     COMMENT*************AP PROCESS******************
     *
     * DESCRIPTION:
     *            CONTAINS DATA STRUCTURES AND INTERFACING
     *            PROCEDURES COMMON TO MKIIBL PROCESSES
     * FUNCTICN:
     *         WHEN A PROCESS IS PREFIXED BY AP(PI) IT
     *         WILL HAVE ALL THE BELOW LISTED ATTRIBUTES
     *
     *
     * VARIABLES:
     *
     * PI :        PROCESS INDEX - INDICATIVE OF ITS PRIORITY
     * CIX:        PROCESS ALLOCATOR CALL INDEX
     * PRSTART:    SIMULATED TIME VALUE AT THE START OF A PROLO WAIT
     * PERIODIC:   BOOLEAN, TRUE FOR A PERIODIC PROCESS
     * REMAININGACTIME:
     *             TIME TO COMPLETE AN ACTIVITY WHEN A PROCESS IS
     *             INTERRUPTED
     * INIT:       TIMES PROCESS INITIATED
     * MAX:        MAX NO OF TASKS IN I/P QUEUE
     * MIN:        MIN "  "  "  "  "  "
     * QLEN:       LENGTH OF I/P QUEUE
     * PMIN:       MIN NO OF PROCESS ALLOCATORS WAITING FOR PROLO
     * PMAX:       MAX "  "     "      "         "       "    "
     * PAQ :          "  "      "      "         "       "    "
     * PNUM:       TIMES PROLO ENGAGED
     * PRWAIT:     TOTAL TIME PROCESS ALLOCATORS  "       "    "
     * PROLO:      BOOLEAN,TRUE WHEN PROLO IS ENGAGED
     * PD:         PROCESS DESCRIPTOR
     * TIT:        TASK INDEX TABLE
     * PROLOQ:     QUEUF WHERE PROCESS ALLOCATORS WAIT FOR PROLO
     * INPUTC:     PROCESS INPUT QUEUE
     * RELEVANTCPU:REFERS TO CPU IN WHICH PROCESS IS RUNNING
     * PA:         REFERS TO PROCESS ALLOCATOR OF CPU WHERE
     *             THE PROCESS IS RUNNING
     * T:          A TEXT REF HOLDING THE NAME OF THE PROCESS
     * PROCEDURES:
     * ----------
     * BLOCK(N): DETERMINES CIX,PASSIVATES PROCESS,ACTIVATES PA
     * FETCH(N):     "      "     "        "      "     "
     * SEEK(N) :     "      "     "        "      "     "
     * SELF(N):      "      "     "        "      "     "
     * FBLOCK(P):    "      "     "        "      "     "
     * CC:       RETURNS WITH THE VALUE OF CONDITION CODES
```

```
             ********************** COMMENT ENDS*************;

        PROCESS CLASS AP(PI); INTEGER PI;
        COMMENT***HAS AS ATTRIBUTES PD,TIT AND THE
         CALLS PROCEDURES TO THE PA***;
        BEGIN
          INTEGER I,CIX,H,M,S,INIT ,D,X,MAX,MIN,QLEN,PAQ,
          PMIN,PMAX,FNUM,PRWAIT,PRSTART,W;
          BOOLEAN PRCLO,PERIODIC;
          SHORT INTEGER ARRAY PD(0:31),G(0:7),TIT(1:52,1:5);
          REF(CPU)RELEVANTCPU;
          REF(PROCESSALLOCATOR)PA;
          REAL  REMAININGACTIME;
          TEXT T;



          REF(HEAD)PRCLOQ,INPUTQ;
          PROCEDURE HAND;
          BEGIN
            IF T NE "INTIM" THEN FOR I:=0 STEP 1 UNTIL 7 DO
            RELEVANTCPU.G(I):=G(I);
            CIX:=2;
            ACTIVATE PA AFTER CURRENT;
            PASSIVATE;
          END**HAND***;


          PROCEDURE BLOCK(N); INTEGER N;
          BEGIN
            CIX:=5;
            S:=N;
            RELEVANTCPU.CCS:=0;
            ACTIVATE PA AFTER CURRENT;
            PASSIVATE;
          END**BLOCK**;


          PROCEDURE FETCH(N);INTEGER N;
          BEGIN
            CIX:=4;
            RELEVANTCPU.CCS:=0;
            H:=N;
            ACTIVATE PA AFTER CURRENT;
            PASSIVATE;
          END**FETCH**;


          PROCEDURE SEEK(N); INTEGER N;
          BEGIN
            CIX:=3;
            RELEVANTCPU.CCS:=0;
            M:=N;
            ACTIVATE PA AFTER CURRENT;
            PASSIVATE;
          END**SEEKN**;


          PROCEDURE SELF(N); INTEGER N;
          BEGIN
            CIX:=1;
            FOR I:=0 STEP 1 UNTIL 7 DO
            RELEVANTCPU.G(I):=G(I);
            D:=N;
            ACTIVATE PA AFTER CURRENT;
            PASSIVATE;
          END***SELFN***;


          PROCEDURE FBLOCK(P); INTEGER P;
          BEGIN
            CIX:=6;
            W:=P;
            RELEVANTCPU.CCS:=0;
            ACTIVATE PA AFTER CURRENT;
            PASSIVATE;
          END** FBLOCK ****;


          INTEGER PROCEDURE CC;
          BEGIN
            CC:=RELEVANTCPU.CCS;
          END***CC**;
        END**CLASS AP**PI**;



             COMMENT*********** CPU PROCESS ******************
             *
             *DESCRIPTION: SIMULATES THE RELEVANT CHARACTARISTICS OF A CPU
             *FUNCTION   : CREATES A PROCESS ALLOCATOR INSTANCE FOR
             *               THIS CPU.RETAIN REFRENCE TO ITS CURP.
             *
             *VARIABLES  :
             * NUMBER  THIS CPU NUMBER
             * CCS:    CONDITION CODES
             * CURP:   AREF TO THE CURRENT RUNNING PROCESS
            * BKGTIME:TIME SPENT ON BACKGROUND
            * STARTTIME: OF THIS CPU STARTING PROCESS EXECUSION
            * MYPA:      REFERS TO PROCESS ALLOCATOR ON THIS CPU
            * G(0-15):  16 GENERAL REGISTERS
            *
            * INPUT TO: MYPA
            * OUTPUT FROM: NONE
            * ACTIVATES: NONE
            * ACTIVATED BY: NONE
            *
            ****************COMMENT ENDS **************;

      PROCESS CLASS CPU(NUMBER);INTEGER NUMBER;
      BEGIN                                                    B17
        INTEGER CCS,I;
        REF(AP)CURP;
        REAL BKGTIME,STARTTIME;
        REF(PROCESSALLOCATOR)MYPA;
        INTEGER ARRAY G(0:15);
        MYPA:-NEW PROCESSALLOCATOR;
        MYPA.CALLINGPROCESS:-CURP;
        MYPA.MYCPU:-THIS CPU;
      END***CPU***;                                            E17



      LINK CLASS TASKBLOCK;
      BEGIN                                                    B18
        COMMENT*********************************
        IN A FREETASKBLOCK
        TASK(1) WORD STORES THE CALLING PROCESS INDEX,
        TASK(2) WORD STORES THE TASK PRIORITY,
        TASK(3) WORD STORES THE ICTI I.E. OGTI FOR CALLED PROCESS ,
        TASK(4)-TASK(10) WORDS STORE THE TASK DETAILS ,
        ***********COMMENT ENDS****************;
        INTEGER ARRAY TASK(1:10);
      END**TASKBLOCK**;                                        E18



      COMMENT*********STORAGE ALLOCATOR**************
      *
      *DESCRIPTION: RESPONSIBLE FOR STORE MANAGEMENT
      *
      *FUNCTION: MODELS ONLY SERVICING OF THE FOLLOWING
      *          REQUESTS FROM OTHER PROCESSES: OPEN FILE
      *           ,CLOSE FILE AND PART-FILE READ.WHEN REQUEST
      *          HONOURED SA RETURNS TASK TO REQUESTING PROCESS.
      *
      *VARIABLES:
      *    OPENFILE  LABEL,OPEN FILE REQUEST TO SA
      *    CLOSEFILE  LABEL,CLOSE FILE REQUEST TO SA
      *    PARTFILE  LABEL,PART-FILE READ REQUEST TO SA
      *INPUT TO:REQUESTING PROCESS
      *OUTPUT FROM:REQUESTING PROCESS
      *ACTIVATES: NONE
      *ACTIVATED BY: NONE
      *
      ***********************************************;

              AP CLASS STORAGEALLOCATOR;
      COMMENT---------------------------;
      BEGIN                                                    B19

        A: IF FLAG THEN
        OUTTV("RELEVANTCPU,S G(2)=",RELEVANTCPU.G(2));
        IF RELEVANTCPU.G(2) = 8 THEN GO TO PARTFILE
        ELSE
        IF RELEVANTCPU.G(2) = 4 THEN GO TO OPENFILE
        ELSE
        IF RELEVANTCPU.G(2) = 5 THEN GO TO CLOSEFILE
        ELSE
        OUTLINE("ILLEGAL ENTRY TO STORAGE ALLOCATOR*****");
        OPENFILE: HOLD(4050.00);
        COMMENT***TIME TO OPEN FILE********;
        IF FLAG THEN
        OUTLINE("OPEN FILE REQUEST TO SA");
        G(0):=RELEVANTCPU.G(0);
        G(1):=RELEVANTCPU.G(1);
        G(2):=RELEVANTCPU.G(2);
        HAND;
        COMMENT***TASK RETURNED TO CALLING PROCESS WITHOUT CHANGE
        ,ICTI IN G(0) NOW USED AS OGTI TO TRANSLATE TO CALLING
        PROCESS**********;
        BLOCK(10);
        GO TO A;
        CLOSEFILE: HOLD(4300.00);
        COMMENT***TIME TO CLOSE FILE****;
        IF FLAG THEN
        OUTLINE("CLOSE FILE REQUEST TO SA");
        G(0):=RELEVANTCPU.G(0);
        G(1):=RELEVANTCPU.G(1);
        G(2):=RELEVANTCPU.G(2);
        HAND;
        COMMENT*****SEE COMMENT AFTER HAND ABOVE*****;
        BLOCK(10);
        GO TO A;
```

```
  PARTFILE: HOLD(5000.0);                                                 00486000
  COMMENT***TIME TO PART-FILE READ MAY LATER BE DRAWN FROM AN IMPIRICAL   00487000
  DISTRIBUTION ACCORDING TO NO. OF CPUS IN THE SYSTEM**;                  00488000
  G(0):=RELEVANTCPU.G(0)+(RELEVANTCPU.G(1)-HPCPS);                        00489000
  G(1):= 5;                                                               00490000
  G(2):=RELEVANTCPU.G(1);                                                 00491000
  G(3):=RELEVANTCPU.G(2);                                                 00492000
  HAND;                                                                   00493000
  IF FLAG THEN OUTLINE("PART-FILE READ TO SA**");                         00494000
  COMMENT****SEE COMMENT AFTER OPENFILE****;                              00495000
  BLOCK(10);                                                              00496000
  GO TO A;                                                                00497000
END***OF STORAGE ALLOCATOR****;                                          00498000 E19
                                                                         00499000
                                                                         00500000
                                                                         00501000
COMMENT****************RASH PROCESS****************                      00502000
*                                                                        00503000
*DESCRIPTION:                                                            00504000
*     A SUITE OF PROCESSES TO AID IN THE DEBUGGING AND                   00505000
*    COMMISSIONING OF MK IIBL OPERATING SYSTEM.                          00506000
*                                                                        00507000
*FUNCTION:                                                               00508000
*     IN THIS EXPT. RASH PROCESSES EXERCISE CALLS TO                     00509000
*     THE STORAGE ALLOCATOR TO OPEN AND CLOSE FILES ONLY                 00510000
*VARIABLES:                                                              00511000
*     COUNTER  TO RECORD NO. OF TIMES RASH LOOP TRAVERSED                00512000
*INPUT TO:    SA                                                         00513000
*OUTPUT FROM:   SA,INTIM                                                 00514000
*ACTIVATES: NONE                                                         00515000
*ACTIVATED BY: NONE                                                      00516000
*                                                                        00517000
*****************************************************;                  00518000
                                                                         00519000
      AP CLASS RASH;                                                     00520000
COMMENT-------------------;                                              00521000
BEGIN                                                                    00522000 B20
  INTEGER COUNTER;                                                        00523000
  LOOP: HOLD(579.2);                                                      00524000
  COMMENT***SET UP TASK TO CLOSE FILE TIME****;                           00525000
  G(0):=1;                                                                00526000
  G(1):=PI;                                                               00527000
  G(2):=5;                                                                00528000
  HAND;                                                                   00529000
  COMMENT**CLOSE FILE REQUEST RO SA***;                                   00530000
  BLOCK(3);                                                               00531000
  COMMENT***WAITING FOR TASK FROM SA WHOSE PRIORITY=3***;                 00532000
  HOLD(459.20);                                                           00533000
  COMMENT****SET UP TASK TO OPEN FILE*****;                               00534000
  G(0):=1;                                                                00535000
  G(1):=PI;                                                               00536000
  G(2):=4;                                                                00537000
  HAND;                                                                   00538000
  COMMENT***OPEN FILE REQUEST TO SA***;                                   00539000
  BLOCK(3);                                                               00540000
  COMMENT***WAITING FOR TASK FROM SA WHOSE PRIORITY=3**;                  00541000
  HOLD(109.20);                                                           00542000
  COMMENT****HOUSE KEEPING*********;                                      00543000
  FETCH(15);                                                              00544000
  COMMENT***LOOKING FOR INTIM TASK WHICH OCCURS EVERY 100MSEC*;           00545000
  IF CC>0 THEN                                                            00546000
  BEGIN                                                                   00547000 B21
    HOLD(600.00);                                                          00548000
    COMMENT***DEAL WITH INTIM TASK****;                                    00549000
    COUNTER:=COUNTER+1;                                                    00550000
  END***RESPONSE TO INTIM TASK*** ELSE                                    00551000 E21
  COUNTER:=COUNTER+1;                                                     00552000
  IF FLAG THEN                                                            00553000
  OUTTV("COUNTER OF RASH=",COUNTER);                                      00554000
  HOLD(109.20);                                                           00555000
  COMMENT****HOUSEKEEPING**********;                                      00556000
  GO TO LOOP;                                                             00557000
END****OF RASH PROCESS******;                                            00558000 E20
                                                                         00559000
                                                                         00560000
                                                                         00561000
     COMMENT*************INTIM PROCESS*************                      00562000
     *                                                                   00563000
     * DESCRIPTION:- THE INTERRUPT AND TIMING PROCESS                    00564000
     * FUNCTION: HANDS UNBLOCKING  TASKS TO PERIODIC                     00565000
     *           PROCESS AND TIMING TASKS                                00566000
     * VARIABLES:                                                        00567000
     *    X     INTEGER                                                  00568000
     * PTR     A POINTER                                                 00569000
     * INPUT TO: NONE                                                    00570000
     * OUTPUE FROM: NONE                                                 00571000
     * ACTIVATES: NONE                                                   00572000
     * ACTIVATED BY: NONE                                                00573000
     ****************COMMENT ENDS**************;                         00574000
                                                                         00575000
AP CLASS INTIM;                                                          00576000
COMMENT*** PI=16**;                                                      00577000
BEGIN                                                                    00578000 B22
  INTEGER X,PTR;                                                          00579000
  START: X:=0;                                                            00580000
  IF FLAG THEN OUTLINE("INTIM CORRECTLY ENTERED");                        00581000
  FOR X:=1 STEP 1 UNTIL 30 DO                                             00582000
  BEGIN                                                          00583000 B23
    IF INTRIP.PPTABLE(PTR,X) NE 0 THEN                             00584000
    BEGIN                                                          00585000 B24
      TIT(1,2):=INTRIP.PPTABLE(PTR,X);                               00586000
      COMMENT*******G1-G7 STORES PIS OF APS TO                       00587000
      BE ACTIVATED BY INTIM.COPY PI IN INTIM'S                       00588000
      TIT 1,2******;                                                 00589000
      TIT(1,3):=0;                                                   00590000
      TIT(1,4):=5;                                                   00591000
      G(0):=1;                                                       00592000
      COMMENT***THIS IS TRANSLATED BY PA TO ICTI=0                   00593000
      AND PRICRIY=5 WHICH IS UNBLOCKING                              00594000
      FOR AP****;                                                    00595000
      HAND;                                                          00596000
    END**G X  NE 0**;                                              00597000 E24
    IF FLAG THEN                                                   00598000
    BEGIN                                                          00599000 B25
      OUTTV("VALUE OF VARIABLE X NOW IS",X);                         00600000
      OUTTV("PI STORED IN PPTABLE =",INTRIP.PPTABLE(PTR,X));         00601000
      OUTIMAGE;                                                      00602000
    END;                                                           00603000 E25
  END;                                                           00604000 E23
  HOLD(800.0);                                                   00605000
  COMMENT***TIME TAKEN BY INTIM*****;                            00606000
  BLOCK(15);                                                     00607000
  GOTO START;                                                    00608000
END**INTIM***;                                                  00609000 E22
                                                                00610000
                                                                00611000
                                                                00612000
     COMMENT*********** INTERRUPT TRIPLICATES ********           00613000
     *                                                           00614000
     *DESCRIPTION:                                               00615000
     *           THIS MODULE MODELS THE ACTIVITES OF             00616000
     *           THE INTERRUPT TRIPLICATES UNITS                 00617000
     *FUNCTION:                                                  00618000
     *         RETAINS REF TO PA ON LCPU.WHEN SUSPENDED          00619000
     *         QUEUE IS TRIGGERED OR A CLOCK INTERRUPT           00620000
     *         ARRIVES IT INTERRUPTS LCPU                        00621000
     *VARIABLES:                                                 00622000
     * INTLO: INTERRUPT TRIPLICATES LOCK OUT                     00623000
     * SUSPOINT:TRUE WHEN SUSPENDED QUEUE IS TRIGGERED           00624000
     * CLOCKINT:TRUE WHEN A CLOCK INTERRUPT OCCURS               00625000
     * PPTABLE: TABLE OF PERIODIC PROCESSES                      00626000
     * LCPU:    CPU RUNNING LOWEST PRIORITY PROCESS              00627000
     * CUSP:    CURRENT SUSPENDED PROCESS                        00628000
     * CUCP:    CURRENT CALLED PROCESS                           00629000
     * LPA:     PROCESS ALLOCATOR ON LCPU                        00630000
     *                                                           00631000
     *INPUT TO: PA                                               00632000
     *OUTPUT FROM: PA,CLOCKINT                                   00633000
     *ACTIVATES: LPA                                             00634000
     *ACTIVATED BY: PA,CLOCKINT                                  00635000
     ****************** COMMENT ENDS ***************;            00636000
                                                                00637000
PROCESS CLASS INTRIPLICATES;                                    00638000
BEGIN                                                           00639000 B26
  BOOLEAN INTLO,SUSPOINT,CLOCKINT;                               00640000
  SHORT INTEGER ARRAY PPTABLE(1:10,1:30);                        00641000
  REF(CPU)LCPU;                                                  00642000
  REF(AP) CUSP,CUCP;                                             00643000
  REF(PROCESSALLOCATOR)LPA;                                      00644000
                                                                 00645000
  START: LPA:-LCPU.MYPA;                                         00646000
  CUSP:-LPA.CALLINGPROCESS;                                      00647000
  IF FLAG THEN                                                   00648000
  OUTLINE("INTRIP ENTERED NOW");                                 00649000
  IF NOT(CLOCKINT OR SUSPOINT) THEN                              00650000
  BEGIN                                                          00651000 B27
    ERROR(5);                                                      00652000
    OUTIMAGE;                                                      00653000
    OUTTVR("AT TIME=",TIME);                                       00654000
    OUTTV("LCPU IS NO.",LCPU.NUMBER);                              00655000
    WRITE("AND LCPU CURP IS   ",LCPU.CURP.T,LCPU.CURP.PI);         00656000
    OUT;                                                           00657000
    GO TO ESCAPE;                                                  00658000
  END                                                            00659000 E27
  ELSE                                                           00660000
  BEGIN                                                          00661000 B28
    IF SUSPOINT THEN                                               00662000
    BEGIN                                                          00663000 B29
      SUSPOINT:=FALSE;                                               00664000
      COMMENT***STEER INTERRUPT TO LCPU***;                          00665000
      WHILE NOT LPA.IDLE DO WAIT(LPAQ);                              00666000
      OUT;                                                           00667000
      I:= 0;                                                         00668000
      WHILE (AND2(NOT SUSPMAP(I) ,I LE 126 ) ) DO                    00669000
      I := I + 1;                                                    00670000
      IF I GE LPA.CALLINGPROCESS.PI THEN                             00671000
      BEGIN                                                          00672000 B30
        IF(CUCP=/=NONE AND CUCP.IDLE) THEN ACTIVATE CUCP               00673000
        AFTER CURRENT;                                                 00674000
        IF LPA.CALLINGPROCESS.IDLE THEN ACTIVATE LPA.CALLINGPROCESS;   00675000
        IF FLAG THEN                                                   00676000
        OUTLINE("SUSPOINT NOT SERVED"); GO TO ESCAPE;                  00677000
      END;                                                           00678000 E30
      IF FLAG THEN                                                   00679000
      OUTLINE("INTRIP ENTERED BECAUSE SUSPOINT TRIGGERED");                  00680000
      LPA.INTERRUPT := TRUE;                                                 00681000
      IF NCT CUSP.IDLE THEN                                                  00682000
      BEGIN                                                                  00683000 B31
        CUSP.REMAININGACTIME := CUSP.EVTIME - TIME;                          00684000
        CANCEL(CUSP);                                                        00685000
        COMMENT*** SUSPEND LCPU CURP ***;                                    00686000
      END;                                                                   00687000 E31
      ACTIVATE LPA AFTER CURRENT;                                            00688000
    END***SUSPOINT***;                                                       00689000 E29
    IF CLOCKINT THEN                                                         00690000
    BEGIN                                                                    00691000 B32
      WHILE NCT LPA.IDLE DO WAIT(LPAO);                                      00692000
      IF FLAG THEN OUTTV("INT. TRIPLICATES ENTERED FOR CLOCKINT",TIME);      00693000
      CUT;                                                                   00694000
      LPA:-LCFU.MYPA;                                                        00695000
      WHILE NCT LPA.IDLE DO WAIT(LPAO);                                      00696000
      OUT;                                                                   00697000
      LPA.CLCCK:=TRUE;                                                       00698000
      CUSP:-LPA.CALLINGPROCESS;                                              00699000
      LPA.INTERRUPT := TRUE;                                                 00700000
      IF NCT CUSP.IDLE THEN                                                  00701000
      BEGIN                                                                  00702000 B33
        CUSP.REMAININGACTIME := CUSP.EVTIME - TIME;                          00703000
        CANCEL(CUSP);                                                        00704000
        COMMENT*** SUSPEND LCPU CURP ***;                                    00705000
      END;                                                                   00706000 E33
      ACTIVATE LPA AFTER CURRENT;                                            00707000
      IF FLAG THEN                                                           00708000
      OUTLINE("INTRIP HAS SERVED CLOCKINT");                                 00709000
      CLOCKINT:=FALSE;                                                       00710000
    END**CLOCKINT**;                                                         00711000 E32
  END**NOT CLCCKINT OR SUSPOINT **;                                          00712000 E28
  ESCAPE:   PASSIVATE;                                                       00713000
  GOTO START;                                                                00714000
END**INTRIPLICATES***;                                                       00715000 E26
                                                                             00716000
                                                                             00717000
                                                                             00718000
     COMMENT**************** PROCESS ALLOCATOR ***********                   00719000
     *                                                                       00720000
     *DESCRIPTION: THIS SIMULATION MODULE IS AT THE HEART                    00721000
     *             OF THE MKIIBL SIMULATION PACKAGE                          00722000
     *FUNCTION: SCHEDULES PROCESSES TO RUN ON DIFFERENT                      00723000
     *          CPUS ON PRIORITY BASIS.HANDLES COMMUNICATION                 00724000
     *          BETWEEN THE PROCESSES AND SERVICES HARDWARE                  00725000
     *          AND SOFTWARE INTERRUPTS                                      00726000
     *VARIABLES:                                                             00727000
     * LASTSTATE: INDICATES SUSPENDED PROCESS STATE                          00728000
     * OGTI: CUTGOING TASK INDEX                                             00729000
     * PAC : TIMES PROCESS ALLOCATOR CALLED                                   00730000
     * PAINT: "    "       "     INTERRUPTED                                 00731000
     * INTERRUPT: TRUE WHEN AN INTERRUPT OCCURS                              00732000
     * CLOCK: TRUE WHEN A CLOCK INTERRUPT OCCURS                              00733000
     * PAOVERHEAD: PROCESS ALLOCATOR OVERHEAD FOR                            00734000
     *             SERVICING CALLS AND INTERRUPTS                            00735000
     * FRSTART: TIME AT THE START OF A FREELO WAIT                           00736000
     * SUSTART:  "   "   "   "    " " SUSPLO "                               00737000
     * INSTART:  "   "   "   "    " " INTLO  "                               00738000
     * STARTTIMF:"   "   "   "    " "  PA SERVICING ACTIVITY                 00739000
     * PROVERHEAD: FETCH AND BLOCK CALLS PA OVERHEAD                         00740000
     * CALLINGPROCESS: REF TO PROCESS CALLING THE PA                         00741000
     * CALLEDPROCESS :  "  "      " CALLED BY CALLINGPROCESS                 00742000
     * UNSUSPENDED :    "  "      " SELECTED AND TAKEN OUT OF                00743000
     *               SUSPENDED STATE MAP                                     00744000
     * RUNNINGPROCESS: REFERS TO PROCESS RUNNING ON THIS CPU                 00745000
     * CLINDEX: A SWITCH TO BRANCH TO A CALL-SERVICING                       00746000
     *          SEQUENCE OF PA ACCORDING TO THE CALL INDEX                   00747000
     * MYCPU: REF TO CPU ON WHICH THIS PA RUNS                               00748000
     * TASKHANDED: REF TO TASK JUST HANDED                                   00749000
     * TASKFCUND:   "  "   "    "   FOUND                                    00750000
     * FREETASKBLOCK: REF TO FIRST TASK IN FREE TASK LIST                    00751000
     *                                                                       00752000
     * PROCEDURES:                                                           00753000
     * ----------                                                            00754000
     * PROLO1(PROC): ENGAGES PROLO LO OR WAITS UNTIL RELEASED                00755000
     * QLENGTH(JOB) : CALCULATES PROCESS MAX AND MIN Q LENGTH                00756000
     * FREELC2:ENGAGES FREELO AND FETCHES A FREE TASK BLOCK                  00757000
     * FREELO1:ENGAGES FREELO AND RETURNS TASK BLOCK TO FREE                 00758000
     *         SPACE LIST                                                    00759000
     * SUSPLC1(SUSPROC): INCLUDES A PROCESS IN SUSP. STATE MAP               00760000
     *                   ENGAGING SUSPLO                                     00761000
     * SUSPLC2: REMOVES A PROCESS FROM SUSPENDED STATE MAP.                  00762000
     *         ENGAGING SUSPLO                                               00763000
     * LOADTASK: LOADS TASK DETAILS INTO CPU G0-G7 AND PROCESS               00764000
     *           PD(17-24)                                                   00765000
     * SYSCHED: ENGAGES INTLO AND TRIGGERS SUSP.Q INTERRUPT IF               00766000
     *          ASUSPENDED PROCESS IS OF HIGHEST PRIORITY THAN               00767000
     *          THE PROCESS RUNNING ON LCPU                                  00768000
     * CPUSCHED: FINDS LCPU,INGAGES INTLO AND SEND NEW LCPU                  00769000
     *           VALUE TO INTERRUPT TRIPLICATES                              00770000
     * SECS: CALCULATES THE TIME TAKEN TO INSERT A TASK IN                   00771000
     *       PROCESS'S INPUT QUEUE                                           00772000
     * HANDTASKANDSETSTATE: HANDS TASK TO PROCESS AND SETS                   00773000
     *                      IT SUSPENDED IF TASK UNBLOCKING                  00774000
     * INPUTS TO: INTERRUPT TRIPLICATES                                      00775000
     * OUTPUT FROM: INTERRUPT TRIPLICATES,PROCESSES(CALLS)                   00776000
```

```
         * ACTIVATES: INTERRUPT TRIPLICATES,CALLINGPROCESS,CALLED          00777000
         *                PROCESS,OTHER PAS WAITING FOR LOCK OUTS AFTER    00778000
         *                BEING RELEASED BY THIS PA AND OTHER AP'S         00779000
                          REQUESTING SERVICE                               00780000
         * ACTIVATED BY: INTERRUPT TRIPLICATES                             00781000
         *********************** COMMENT ENDS *****************;           00782000
                                                                           00783000
PROCESS CLASS PROCESSALLOCATOR;                                            00784000
BEGIN                                                                      00785000 B34
  INTEGER Z,LASTSTATE,K,I,J,OCTI,Y,PAC,                                      00786000
  PAINT,FRSTART,SUSTART,INSTART;                                             00787000
  REAL PAOVERHEAD, STARTTIME, FBOVERHEAD;                                    00788000
  BOOLEAN INTERRUPT,CLOCK;                                                   00789000
  REF(AP)CALLINGPROCESS,CALLEDPROCESS,UNSUSPENDED                            00790000
  ,RUNNINGPROCESS;                                                           00791000
  SWITCH CLINDEX:=SELV,HND,SEEKING,FETCHING,BLOCKING,FBLOCKING;              00792000
  REF(CPU)MYCPU;                                                             00793000
  REF(TASKBLOCK)TASKHANDED,TASKFOUND,FREETASKBLOCK;                          00794000
                                                                             00795000
                                                                             00796000
  PROCEDURE PROLO1(PROC);                                                    00797000
  REF(AP)PROC;                                                               00798000
  COMMENT***ENGAGES PROLO LO OR WAITS UNTILL RELEASED***;                    00799000
  BEGIN                                                                      00800000 B35
    PROC.PRSTART := TIME;                                                      00801000
    WHILE PROC.PROLO DO WAIT(PROC.PROLOQ);                                     00802000
    INSPECT PROC DO                                                            00803000
    BEGIN                                                                      00804000 B36
      PAQ := PROLOQ.CARDINAL;                                                    00805000
      IF PAQ GT PMAX THEN PMAX := PAQ;                                           00806000
      IF PAQ LT PMIN THEN PMIN := PAQ;                                           00807000
      PRWAIT := PRWAIT + TIME - PRSTART;                                         00808000
      PRSTART := 0.0;                                                            00809000
      PNUM := PNUM + 1;                                                          00810000
    END;                                                                       00811000 E36
    OUT;                                                                       00812000
    PROC.PROLO := TRUE;                                                        00813000
  END**PROLO1***;                                                            00814000 E35
                                                                             00815000
                                                                             00816000
  PROCEDURE QLENGTH(JOB);REF(AP)JOB;                                         00817000
  BEGIN                                                                      00818000 B37
    INSPECT JOB DO                                                             00819000
    BEGIN                                                                      00820000 B38
      QLEN:=INPUTQ.CARDINAL;                                                     00821000
      IF QLEN GT MAX THEN MAX:=QLEN;                                             00822000
      IF QLEN LT MIN THEN MIN:=QLEN;                                             00823000
    END;                                                                       00824000 E38
  END**OF QLENGTH WHICH CALCULATES AP QLENGTHS**;                            00825000 E37
                                                                             00826000
                                                                             00827000
  PROCEDURE FREELO1;                                                         00828000
  BEGIN                                                                      00829000 B39
    FRSTART:=TIME;                                                             00830000
    WHILE FREELO DO WAIT(FREELOQ);                                             00831000
    FRENG:=FREELOQ.CARDINAL;                                                   00832000
    IF FRENG GT FMAX THEN FMAX:=FRENG;                                         00833000
    IF FRENG LT FMIN THEN FMIN:=FRENG;                                         00834000
    FRWAIT:=FRWAIT+TIME-FRSTART;                                               00835000
    FRSTART:=0;                                                                00836000
    FNUM:=FNUM+1;                                                              00837000
    OUT;                                                                       00838000
    FREELO:=TRUE;                                                              00839000
    HOLD(15.4);                                                                00840000
    COMMENT***FREELO ACTIVITY TIME***;                                         00841000
    TASKFOUND.INTO(FREETASKLIST);                                              00842000
    FREELO:=FALSE;                                                             00843000
    ACTIVATE FREELOQ.FIRST  ;                                                  00844000
  END**FREELO1:INSERTS TASKBLOCK INTO FREETASKLIST**;                        00845000 E39
                                                                             00846000
                                                                             00847000
  PROCEDURE FREELO2;                                                         00848000
  BEGIN                                                                      00849000 B40
    COMMENT***FETCHS AFREE TASKBLOCK**;                                        00850000
    FRSTART := TIME;                                                           00851000
    WHILE FREELO DO WAIT(FREELOQ);                                             00852000
    FRENG := FREELOQ.CARDINAL;                                                 00853000
    IF FRENG GT FMAX THEN FMAX := FRENG;                                       00854000
    IF FRENG LT FMIN THEN FMIN := FRENG;                                       00855000
    FRWAIT := FRWAIT + TIME - FRSTART;                                         00856000
    FRSTART := 0.0;                                                            00857000
    FNUM := FNUM + 1;                                                          00858000
    OUT;                                                                       00859000
    FREELO := TRUE;                                                            00860000
    HOLD(15.4);                                                                00861000
    COMMENT**FREELO ACTIVITY TIME****;                                         00862000
    TSKIN: IF FREETASKLIST.FIRST IS TASKBLOCK THEN                             00863000
    FREETASKBLOCK:-FREETASKLIST.FIRST QUA TASKBLOCK                            00864000
    ELSE                                                                       00865000
    BEGIN                                                                      00866000 B41
      NEW TASKBLOCK.INTO(FREETASKLIST);                                          00867000
      GO TO TSKIN;                                                               00868000
    END;                                                                       00869000 E41
    FREETASKBLOCK.OUT;                                                         00870000
    FREELO:=FALSE;                                                             00871000
    ACTIVATE FREELOQ.FIRST  ;                                                  00872000
  END**FREELO2***;                                                           00873000 E40
```

```
                                                                                 00874000
                                                                                 00875000
        PROCEDURE SUSPLO1(SUSPROC);                                              00876000
        REF(AP)SUSPROC;                                                          00877000
        BEGIN                                                                    00878000 B42
          COMMENT**INCLUDES SUSPENDED PROCESS IN SUSPMAP***;                       00879000
          SUSTART:=TIME;                                                           00880000
          WHILE SUSPLO DO WAIT(SUSPLOQ);                                           00881000
          SUSENG:=SUSPLOQ.CARDINAL;                                                00882000
          IF SUSENG GT SMAX THEN SMAX:=SUSENG;                                     00883000
          IF SUSENG LT SMIN THEN SMIN:=SUSENG;                                     00884000
          SUSWAIT:=SUSWAIT+TIME-SUSTART;                                           00885000
          SUSTART:=0;                                                              00886000
          SNUM:=SNUM+1;                                                            00887000
          OUT;                                                                     00888000
          SUSPLO:=TRUE;                                                            00889000
          HOLD(10.2);                                                              00890000
          COMMENT***THIS IS SUSPLO1 ACTIVITY TIME*****;                            00891000
          SUSPMAP(SUSPROC.PI):=TRUE;                                               00892000
          IF FLAG THEN                                                             00893000
          BEGIN                                                                    00894000 B43
            WRITE("PROCESS INTERED IN SUSPMAP IS    ",SUSPROC.T,SUSPROC.PI);         00895000
            OUTTV("AT RUNTIME =",TIME);                                              00896000
            OUTIMAGE;                                                                00897000
            OUTLINE("SUSPENDED PROCESSES IN SUSPMAP ARE:-");                         00898000
            FOR I:=0 STEP 1 UNTIL 127 DO                                             00899000
            IF SUSPMAP(I) THEN OUTTV(P(I).T,I);                                      00900000
            OUTIMAGE;                                                                00901000
          END;                                                                     00902000 E43
           COMMENT***SET PROCESS SUSPENDED IN SUSPMAP***;                          00903000
          SUSPLO:=FALSE;                                                           00904000
          ACTIVATE SUSPLOQ.FIRST  ;                                                00905000
        END**SUSPLO1***;                                                         00906000 E42
                                                                                 00907000
                                                                                 00908000
        PROCEDURE SUSPLO2;                                                       00909000
         COMMENT**ENGAGES SUSPLO AND REMOVES SUSP PROCESS                        00910000
        FROM SUSPMAP***;                                                         00911000
        BEGIN                                                                    00912000 B44
          INTEGER I;                                                               00913000
          SUSTART:=TIME;                                                           00914000
          WHILE SUSPLO DO WAIT(SUSPLOQ);                                           00915000
          SUSENG:=SUSPLOQ.CARDINAL;                                                00916000
          IF SUSENG GT SMAX THEN SMAX:=SUSENG;                                     00917000
          IF SUSENG LT SMIN THEN SMIN:=SUSENG;                                     00918000
          SUSWAIT:=SUSWAIT+TIME-SUSTART;                                           00919000
          SUSTART:=0;                                                              00920000
          SNUM:=SNUM+1;                                                            00921000
          OUT;                                                                     00922000
          SUSPLO:=TRUE;                                                            00923000
          I:=0;                                                                    00924000
          WHILE ((NOT SUSPMAP(I)) AND I<127) DO I:=I+1;                            00925000
          IF I=127 THEN                                                            00926000
          BEGIN                                                                    00927000 B45
            IF FLAG THEN                                                             00928000
            BEGIN                                                                    00929000 B46
              OUTLINE("NO SUSPENDED PROCESS FOUND");                                   00930000
              OUTTV("TO RUN ON CPU NO.",MYCPU.NUMBER);                                 00931000
              OUTTV("AT TIME=",TIME);                                                  00932000
              OUTIMAGE;                                                                00933000
            END;                                                                     00934000 E46
            ERROR(2);                                                                00935000
          END**NO SUSP PROCESS** ELSE                                              00936000 E45
          BEGIN                                                                    00937000 B47
            SUSPMAP(I):=FALSE;                                                       00938000
            UNSUSPENDED:-P(I);                                                       00939000
            UNSUSPENDED.RELEVANTCPU:-MYCPU;                                          00940000
            MYCPU.CURP:-UNSUSPENDED;                                                 00941000
            COMMENT**UPDATE THIS CPU CURP******;                                     00942000
            UNSUSPENDED.PA:-THIS PROCESSALLOCATOR;                                   00943000
            IF FLAG THEN                                                             00944000
            BEGIN                                                                    00945000 B48
              WRITE("CHOSEN PROCESS IS    ",P(I).T,I);                                 00946000
              OUTTV("AT TIME =",TIME);                                                 00947000
              OUTTV("TO RUN ON CPU NO.",MYCPU.NUMBER);                                 00948000
              OUTIMAGE;                                                                00949000
              OUTLINE("SUSPENDED PROCESSES NOW ARE:-");                                00950000
              FOR J:=0 STEP 1 UNTIL 127 DO                                             00951000
              IF SUSPMAP(J) THEN OUTTV(P(J).T,J);                                      00952000
              OUTIMAGE;                                                                00953000
            END;                                                                     00954000 E48
            HOLD(16.6+4.0*(I//16));                                                  00955000
            COMMENT***SUSPLO ACTIVITY TIME***;                                       00956000
            SUSPLO:=FALSE;                                                           00957000
            ACTIVATE SUSPLOQ.FIRST  ;                                                00958000
          END;                                                                     00959000 E47
        END**SUSPLO2***;                                                         00960000 E44
                                                                                 00961000
                                                                                 00962000
        PROCEDURE LOADTASK;                                                      00963000
         COMMENT***LOADS IC TASK DETAILS INTO CPU G(0)-G(7)                      00964000
         AND CALLINGPROCESS'S PD(17-24)***;                                      00965000
        BEGIN                                                                    00966000 B49
          INTEGER I,J;                                                             00967000
          FOR I:=3 STEP 1 UNTIL 10 DO                                              00968000
             CALLINGPROCESS.PD(I+14):=TASKFOUND.TASK(I);                           00969000
          FOR J:=0 STEP 1 UNTIL 7 DO                                               00970000
```

```
    COMMENT***MYCPU.G(0) STORES ICTI*****;
    MYCPU.G(J):=TASKFOUND.TASK(J+3);
  END*LOADTASK***;


  PROCEDURE SYSCHED;
  BEGIN
    INTEGER I;
    I:=0;
    WHILE ((NOT SUSPMAP(I)) AND I<127) DO I:=I+1;
    IF I=127 THEN
    BEGIN
      ERROR(2);
      OUTLINE("NO SUSPENDED PROCESS FOUND IN SUSPMAP");
    END
    ELSE
    BEGIN
      INSPECT INTRIP DO
      IF ( I < LCPU.CURP.PI AND I NE 127 )
      THEN
      BEGIN
        INSTART:=TIME;
        WHILE INTLO DO WAIT(INTLOQ);
        INENG:=INTLOQ.CARDINAL;
        IF INENG GT IMAX THEN IMAX:=INENG;
        IF INENG LT IMIN THEN IMIN:=INENG;
        INWAIT:=INWAIT+TIME-INSTART;
        INSTART:=0;
        INUM:=INUM+1;
        OUT;
        INTLO:=TRUE;
        SUSPCINT:=TRUE;
        IF FLAG THEN
        BEGIN
          OUTTV("SUSPOINT TRIGGERED AT TIME=",TIME);
          OUTTV("BY PROCESS ALLOCATOR ON CPU NO.",MYCPU.NUMBER);
          WRITE("BECAUSE HIGHEST PRIO. SUSP PROCESS IS     ",P(I).T,I);
          WRITE("AND LCPU CURP IS    ",LCPU.CURP.T,LCPU.CURP.PI);
          OUTTV("AND LCPU IS NO. ",LCPU.NUMBER);
          OUTIMAGE;
        END;
        HOLD(11.6);
        COMMENT*** INTLO ACTIVITY TIME***;
        IF(LPAQ.EMPTY AND INTRIP.IDLE) THEN ACTIVATE INTRIP;
        INTLO:=FALSE;
        ACTIVATE INTLOQ.FIRST;
      END;
    END;
  END**SYSCHED**;


  PROCEDURE CPUSCHED;
  BEGIN
    INTEGER I,MAX,K;
    SUSPLO2;
    MAX:=0;
    FOR I:=1 STEP 1  UNTIL  NUM DO
    IF CLINT.C(I).CURP.PI>MAX THEN MAX:=CLINT.C(I).CURP.PI;
     COMMENT***COMPARE ALL CPUS TO FIND LCPU***;
    INSTART:=TIME;
    WHILE INTRIP.INTLO DO WAIT(INTLOQ);
    INENG:=INTLOQ.CARDINAL;
    IF INENG GT IMAX THEN IMAX:=INENG;
    IF INENG LT IMIN THEN IMIN:=INENG;
    INWAIT:=INWAIT+TIME-INSTART;
    INSTART:=0;
    INUM:=INUM+1;
    OUT;
    INTRIP.INTLO:=TRUE;
    INTRIP.LCPU:-P(MAX).RELEVANTCPU;
     COMMENT***SEND NEW VALUE OF LCPU TO INTRIPLICATES***;
    IF FLAG THEN
    OUTTV("LCPU UPDATED ,NOW IS NO.",INTRIP.LCPU.NUMBER);
    HOLD(11.6);
    COMMENT***INTLO ACTIVITY TIME****;
    INTRIP.INTLO:=FALSE;
    ACTIVATE INTLOQ.FIRST;
  END***CPUSCHED***;


  REAL PROCEDURE SECS;
  BEGIN
    REAL L,X,B,Y;
    REF(HEAD)QUEUE;
    QUEUE:-CALLEDPROCESS.INPUTQ;
    IF TASKHANDED==QUEUE.LAST THEN X:=0.0 ELSE X:=1.0;
    IF NOT QUEUE.EMPTY THEN L:=QUEUE.CARDINAL
    ELSE L:=0.0;
    IF CALLEDPROCESS.PD(8)=4 THEN B:=1.0 ELSE B:=0.0;
    IF CALLEDPROCESS.PD(8)=4 AND TASKHANDED.TASK(2)<=
    CALLEDPROCESS.PD(9) THEN Y:=1.0 ELSE Y:=0.0;
    IF CALLINGPROCESS.CIX=2 THEN
    SECS:=28.6+10.8*L+1.0*B+7.4*X+16.2*Y ELSE
    SECS:=16.8+10.8*L+7.4*X;
    COMMENT**WHEN CIX=2 SECS IS PROLO ACTIVITY TIME IN
    <HAND> CALL WHEREAS OTHER SECS IS PROLO ACTIVITY
    TIME IN <SELF> CALL TO PA*******;
```

```
END**OF**SECS*****;                                                       01068000 E56
                                                                          01069000
                                                                          01070000
PROCEDURE HANDTASKANDSETSTATE;                                            01071000
 COMMENT***HANDS THE TASK TO THE CALLED PROCESS AND                       01072000
 SETS ITS STATE**;                                                        01073000
BEGIN                                                                     01074000 B57
  REF(HEAD)Q;                                                               01075000
  REF(TASKBLOCK)X;                                                          01076000
  REF(LINKAGE)Y;                                                            01077000
                                                                            01078000
  PROCEDURE SUSPROC;                                                        01079000
  BEGIN                                                                     01080000 B58
    COMMENT*** SUSPENDS PROCESS AND INSERTS IN SUSP MAP ***;                  01081000
    CALLEDPROCESS.PD(8) := 3;                                                 01082000
    HOLD(SECS);                                                               01083000
    CALLEDPROCESS.PROLO := FALSE;                                             01084000
    SUSPLC1(CALLEDPROCESS);                                                   01085000
    QLENGTH(CALLEDPROCESS);                                                   01086000
    ACTIVATE CALLEDPROCESS.PROLOQ.FIRST;                                      01087000
  END ***SUSPROC***;                                                        01088000 E58
                                                                            01089000
  PROCEDURE NOSUSPROC;                                                      01090000
  BEGIN                                                                     01091000 B59
    WRITE("PROCESS HANDED TASK BUT NOT SUSPENDED IS",CALLEDPROCESS.T,         01092000
    CALLEDPROCESS.PI);                                                        01093000
    HOLD(SECS);                                                               01094000
    CALLEDPROCESS.PROLO := FALSE;                                             01095000
    ACTIVATE CALLEDPROCESS.PROLOQ.FIRST;                                      01096000
  END*** NOSUSPROC***;                                                      01097000 E59
  PROLO1(CALLEDPROCESS);                                                    01098000
  Q:-CALLEDPROCESS.INPUTQ;                                                  01099000
  Y:-Q;                                                                     01100000
  IF NOT Q.EMPTY THEN                                                       01101000
  BEGIN                                                                     01102000 B60
    IF Q.LAST QUA TASKBLOCK.TASK(2) LE TASKHANDED.TASK(2)                     01103000
    THEN Y:-Q.LAST ELSE                                                       01104000
    FOR X:-Y.SUC QUA TASKBLOCK WHILE X.TASK(2) LE TASKHANDED.TASK(2) DO       01105000
        Y:-X;                                                                 01106000
  END**INPUTQ NOT EMPTY***;                                                 01107000 E60
  TASKHANDED.FOLLOW(Y);                                                     01108000
  COMMENT***END**OF INSERTING TASK****;                                     01109000
  IF FLAG THEN                                                              01110000
  BEGIN                                                                     01111000 B61
    WRITE("PROCESS HANDED TASK IS    ",CALLEDPROCESS.T,CALLEDPROCESS.PI);     01112000
    OUTTV("AT RUNTIME =",TIME);                                               01113000
    OUTTV("NO. OF TASKS IN INPUTQ =",Q.CARDINAL);                             01114000
    IF CALLINGPROCESS.T NE "BACKGROUND" THEN                                  01115000
    WRITE("HANDING PROCESS IS    ",CALLINGPROCESS.T,CALLINGPROCESS.PI);       01116000
    OUTIMAGE;                                                                 01117000
    OUTLINE("INPUTQ TASKS PRIORITIES ARE:-");                                 01118000
    Y:-Q;                                                                     01119000
    FOR Y:-Y.SUC WHILE Y=/=NONE DO                                            01120000
    OUTINT(Y QUA TASKBLOCK.TASK(2),4);                                        01121000
    OUTIMAGE;                                                                 01122000
  END;                                                                      01123000 E61
  IF CALLEDPROCESS.PERIODIC THEN                                            01124000
  BEGIN                                                                     01125000 B62
    IF AND2(CALLEDPROCESS.PD(8) = 4,TASKHANDED.TASK(2) = 5) THEN              01126000
    SUSPROC ELSE NOSUSPROC                                                    01127000
  END***OF A PERIODIC PROCESS****                                           01128000 E62
  ELSE                                                                      01129000
  BEGIN                                                                     01130000 B63
    IF AND2(CALLEDPROCESS.PD(8) = 4 , TASKHANDED.TASK(2) LE                   01131000
    CALLEDPROCESS.PD(9) ) THEN SUSPROC ELSE NOSUSPROC;                        01132000
  END*** OF AN APERIODIC PROCESS****;                                         01133000 E63
END**HANDTASKANDSETSTATE***;                                              01134000 E57
                                                                          01135000
                                                                          01136000
COMMENT************** START OF PROCESS ALLOCATOR ACTIONS *********;       01137000
                                                                          01138000
PASTART:                                                                  01139000
STARTTIME := TIME;                                                        01140000
IF INTERRUPT THEN GO TO INT ELSE                                          01141000
BEGIN                                                                     01142000 B64
  PAC:=PAC+1;                                                               01143000
  GO TO CLINDEX(CALLINGPROCESS.CIX);                                        01144000
END;                                                                      01145000 E64
INT: INTERRUPT:=FALSE;                                                    01146000
PAINT:=PAINT+1;                                                           01147000
COMMENT*****************;                                                 01148000
IF FLAG THEN                                                              01149000
BEGIN                                                                     01150000 B65
  OUTTV("CPU INTERRUPTED IS NO.",MYCPU.NUMBER);                             01151000
  OUTTV("AND ITS INTERRUPT NO.",PAINT);                                     01152000
  OUTTV("AT RUNTIME =",TIME);                                               01153000
  OUTIMAGE;                                                                 01154000
END;                                                                      01155000 E65
COMMENT******************************;                                     01156000
HOLD(122.0);                                                              01157000
COMMENT**RE-ENT. TIME TO SCANNING OF INT. LEVELS**;                       01158000
RUNNINGPROCESS:-CALLINGPROCESS;                                           01159000
FOR I:=0 STEP 1 UNTIL 7 DO                                                01160000
    RUNNINGPROCESS.PD(I+24):=MYCPU.G(I);                                  01161000
RUNNINGPROCESS.PD(23):=MYCPU.CCS;                                         01162000
COMMENT**NEST GRS IN PD(24-31)*****;                                      01163000
RUNNINGPROCESS.PD(8):=2;                                                  01164000
```

```
SUSPLO1(RUNNINGPROCESS);                                                      01165000
IF RUNNINGPROCESS.T EQ "BACKGROUND" THEN                                      01166000
MYCPU.BKGTIME:=MYCPU.BKGTIME+TIME-MYCPU.STARTTIME;                            01167000
INSTART:=TIME;                                                                01168000
WHILE INTRIP.INTLO DO WAIT(INTLOQ);                                           01169000
INENG:=INTLOQ.CARDINAL;                                                       01170000
IF INENG GT IMAX THEN IMAX:=INENG;                                            01171000
IF INENG LT IMIN THEN IMIN:=INENG;                                            01172000
INWAIT:=INWAIT+TIME-INSTART;                                                  01173000
INSTART:=0;                                                                   01174000
INUM:=INUM+1;                                                                 01175000
OUT;                                                                          01176000
INTRIP.INTLO:=TRUE;                                                           01177000
COMMENT***ENGAGE INT.TRIP.LO***;                                              01178000
HOLD(46.8); COMMENT**THIS IS INTLO TIME***;                                   01179000
INTRIP.INTLO:=FALSE;                                                          01180000
ACTIVATE INTLOQ.FIRST  ;                                                      01181000
IF CLOCK THEN                                                                 01182000
BEGIN                                                                         01183000 B66
   HOLD(72.4);                                                                  01184000
   COMMENT***RE-ENTRANT TIME UP TO HANDING OF PERIODIC                          01185000
   PROCESSES TO INTIM********;                                                  01186000
   FREELO2;                                                                     01187000
   FOR I :=2 STEP 1 UNTIL 8 DO                                                  01188000
       FREETASKBLOCK.TASK(I+2):=INTRIP.                                         01189000
   PPTABLE(POINTER,I);                                                          01190000
   P(16) QUA INTIM.PTR:=POINTER;                                                01191000
       COMMENT***PPTABLE STORES UP TO7 PERIODIC                                 01192000
       PROCESS AT THE MOMENT GOVERNED BY THE NO                                 01193000
       OF WDS AVAILABLE IN TASK TO INTIM WDS4-10***;                            01194000
   IF POINTER=10 THEN                                                           01195000
   BEGIN                                                                        01196000 B67
      POINTER:=1;                                                                  01197000
      FOR I:=1 STEP 1 UNTIL NUMOFPROCESSES DO                                      01198000
      INTRIP.PPTABLE(1,I+1):=0;                                                    01199000
   END                                                                          01200000 E67
   ELSE                                                                         01201000
   POINTER:=POINTER+1;                                                          01202000
   CALLEDPROCESS:-P(16);                                                        01203000
   TASKHANDED:-FREETASKBLOCK;                                                   01204000
   HANDTASKANDSETSTATE;                                                         01205000
   COMMENT***TIME TAKEN TO FORM AND HAND TASK TO INTIM;                         01206000
   CLOCK:=FALSE;                                                                01207000
   IF CALLEDPROCESS.PD(8)=3 THEN Y:=1 ELSE Y:=0;                                01208000
   HOLD(Y*(18.6+4.0*(CALLEDPROCESS.PI//16)));                                   01209000
   COMMENT**REMAINDER OF R-TIME TO HANDING OF INTIM TSK***;                     01210000
END**OF CLOCKINT**;                                                           01211000 E66
CPUSCHED;                                                                     01212000
INTRIP.SUSPCINT:=FALSE;                                                       01213000
SYSCHED;                                                                      01214000
RUNSELCPROC: LASTSTATE:=UNSUSPENDED.PD(8);                                    01215000
COMMENT*******************************;                                       01216000
IF FLAG THEN                                                                  01217000
BEGIN                                                                         01218000 B68
   OUTTEXT("SELECTED PROCESS STATE IS  ");                                      01219000
   OUTTEXT(IF UNSUSPENDED.PD(8)=3 THEN "SUSP(UNBLOCKED)" ELSE                   01220000
   "SUSP(INTERRUPTED)");                                                        01221000
   OUTTV("AT RUNTIME =",TIME);                                                  01222000
   OUTTV("AND ITS PI=",UNSUSPENDED.PI);                                         01223000
   OUTIMAGE;                                                                    01224000
END;                                                                          01225000 E68
UNSUSPENDED.PD(8):=1;                                                         01226000
CALLINGPROCESS:-UNSUSPENDED;                                                  01227000
IF LASTSTATE=2 THEN                                                           01228000
BEGIN                                                                         01229000 B69
   FOR I:=0 STEP 1 UNTIL 7 DO                                                   01230000
   MYCPU.G(I):=UNSUSPENDED.PD(I+24);                                            01231000
   MYCPU.CCS:=UNSUSPENDED.PD(23);                                               01232000
    COMMENT**IF SUSP(INT) DENEST GRS FROM PD**,                                 01233000
   DENESTING OF CON. CODES IS TO CATER FOR THE CASE                             01234000
   WHEN A PROCESS IS PRE-EMPTED AFTER A +VE FETCH BUT                           01235000
   BEFORE EXAMINING ITS CCS,AND PRE-EMPTING PROCESS                             01236000
   EXECUTING A =VE FETCH ON SAME CPU THEN***;                                   01237000
   HOLD(148.8);                                                                 01238000
   COMMENT**RE-ENT. TIME WHEN EXITING TO SUSPENDED INTE-                        01239000
   RRUPTED PROCESS FOLLOWING BLOCK,TRAP OR INTERRUPT*****;                      01240000
   PAOVERHEAD := PAOVERHEAD + (TIME - STARTTIME);                               01241000
   IF ( CALLINGPROCESS.PI>30 AND CALLINGPROCESS.PI<50 ) THEN                    01242000
   FBOVERHEAD := FHOVERHEAD + TIME - STARTTIME ;                                01243000
   IF CALLINGPROCESS.T NE "BACKGROUND" THEN                                     01244000
   MYCPU.STARTTIME:=TIME;                                                       01245000
   IF LPAQ.FIRST == NONE THEN                                                   01246000
   BEGIN                                                                        01247000 B70
      IF UNSUSPENDED.REMAININGACTIME>0.00  THEN ACTIVATE                           01248000
      UNSUSPENDED AT (TIME+UNSUSPENDED.REMAININGACTIME)                            01249000
      ELSE ACTIVATE UNSUSPENDED AFTER CURRENT;                                     01250000
   END                                                                          01251000 E70
   ELSE                                                                         01252000
   IF (LPAQ.FIRST =/= NONE AND INTRIP.LPA == (THIS                              01253000
   PROCESSALLOCATOR)) THEN                                                      01254000
   BEGIN                                                                        01255000 B71
      ACTIVATE LPAQ.FIRST AFTER CURRENT;                                           01256000
      INTRIP.CUCP:-UNSUSPENDED;                                                    01257000
   END                                                                          01258000 E71
   ELSE                                                                         01259000
   IF (LPAQ.FIRST =/=NONE AND INTRIP.LPA =/=                                    01260000
   (THIS PROCESSALLOCATOR)) THEN                                                01261000
```

```
      BEGIN                                                              B72
        IF UNSUSPENDED.REMAININGACTIME > 0.00 THEN ACTIVATE
        UNSUSPENDED AT (TIME+UNSUSPENDED.REMAININGACTIME) ELSE ACTIVATE
        UNSUSPENDED AFTER CURRENT;
      END;                                                               E72
      COMMENT****TO ACCOUNT FOR REMAINING ACTIVITY TIME
      OF THE INTERRUPTED PROCESS***************;
      COMMENT**************************************;
      IF FLAG THEN
      BEGIN                                                              B73
        WRITE("PROCESS SELECTED TO RUN IS  ",UNSUSPENDED.T,UNSUSPENDED.PI);
        OUTTV("CN CPU NO. =",MYCPU.NUMBER);
        OUTTV("AT RUNTIME =",TIME);
        OUTIMAGI;
      END;                                                               E73
      COMMENT**************************;
      PASSIVATE;
      GOTO PASTART;
    END**OF SUSF INT ***                                                 E69
    ELSE
    BEGIN                                                                B74
      IF LASTSTATE=3 THEN
      BEGIN                                                              B75
        UNSUSPENDED.INIT:=UNSUSPENDED.INIT+1;
        COMMENT**INIT STORES TIMES AP INITIATED***;
        IF CALLINGPROCESS.T NE "BACKGROUND" THEN
        MYCPU.STARTTIME:=TIME;
        HOLD(165.4);
        COMMENT***RE-ENTRANT TIME WHEN EXITING TO SUSP(UNBLK) PROCESS
        FOLLOWING BLOCK,TRAP OR INTERRUPT*****;
        PROLC1(UNSUSPENDED);
        HOLD(35.0);
        COMMENT***PROLO ACTIVITY TIME****;
        IF CALLINGPROCESS.INPUTQ.FIRST IS TASKBLOCK THEN
        TASKFOUND:-CALLINGPROCESS.INPUTQ.FIRST
        QUA TASKBLOCK ELSE ERROR(7);
        TASKFCUND.OUT;
        UNSUSPENDED.PROLO:=FALSE;
        ACTIVATE UNSUSPENDED.PROLOQ.FIRST;
        LOADTASK;
        COMMENT*******************************;
        IF (FLAG AND CALLINGPROCESS.PI NE 16) THEN
        BEGIN                                                            B76
          WRITE("LOADED TASK IN G0-G7 FOR PROCESS  ",CALLINGPROCESS.T,
          CALLINGPROCESS.PI);
          FOR I :=0 STEP 1 UNTIL 7 DO
          BEGIN                                                          B77
            CUTINT(MYCPU.G(I),5);
            CUTIMAGE;
          END;                                                           E77
        END;                                                             E76
        COMMENT***********************************;
        FREELC1;
        MYCPU.CCS:=1;
        IF CALLINGPROCESS.PI > 100 THEN
        MYCPU.STARTTIME := TIME;
        PAOVERHEAD := PAOVERHEAD + TIME - STARTTIME;
        IF ( CALLINGPROCESS.PI>30 AND CALLINGPROCESS.PI<50 ) THEN
        FROVERHEAD := FBOVERHEAD + TIME - STARTTIME;
        CALLINGPROCESS.RELEVANTCPU:-MYCPU;
        IF LPAQ.FIRST == NONE THEN ACTIVATE UNSUSPENDED
        AFTER CURRENT ELSE
        IF (LPAC.FIRST =/= NONE AND INTRIP.LPA == (THIS
        PROCESSALLOCATOR)) THEN
        BEGIN                                                            B78
          INTRIP.CUCP:-CALLINGPROCESS;
          ACTIVATE LPAQ.FIRST AFTER CURRENT;
        END                                                              E78
        ELSE
        IF (LPAC.FIRST =/= NONE AND INTRIP.LPA =/=
        (THIS PROCESSALLOCATOR)) THEN ACTIVATE UNSUSPENDED
        AFTER CURRENT;
        COMMENT***********************************;
        IF FLAG THEN
        BEGIN                                                            B79
          WRITE("PROCESS SELECTED TO RUN IS  ",UNSUSPENDED.T,UNSUSPENDED.PI);
          CUTTV("TO RUN ON CPU NO.=",MYCPU.NUMBER);
          CUTTV("NO. OF TASKS IN INPUT QUEUE =",UNSUSPENDED.INPUTQ.CARDINAL);
          CUTTV("AT RUNTIME =",TIME);
          CUTIMAGE;
        END;                                                             E79
        COMMENT***********************************;
        PASSIVATE;
        GOTO PASTART;
      END***SUSP UNBL ***;                                               E75
    END;                                                                 E74


    FETCHING:
    STARTTIME := TIME;
    HOLD(12.8);
    COMMENT**RE-ENTRANT TIME FOR UNSUCCESSFUL <FETCH>*****;
    PROLO1(CALLINGPROCESS);
    COMMENT*********************************;
    IF FLAG THEN
    BEGIN                                                                B80
      WRITE("CALL  TO FETCH BY PROCESS  ",CALLINGPROCESS.T,
   CALLINGPROCESS.PI);                                                      01359000
   OUTTV("ON CPU NO.=",MYCPU.NUMBER);                                       01360000
   OUTTV("AT RUNTIME =",TIME);                                              01361000
   OUTTV("AND INPQ TSKS=",CALLINGPROCESS.INPUTQ.CARDINAL);                  01362000
   OUTIMAGE;                                                                01363000
END;                                                                       01364000 E80
COMMENT***************************************;                            01365000
J:=CALLINGPROCESS.H;                                                       01366000
   COMMENT**J IS PRIORITY OF TASK TO BE FETCHED***;                        01367000
IF CALLINGPROCESS.INPUTQ.FIRST IS TASKBLOCK THEN                           01368000
TASKFOUND:-CALLINGPROCESS.INPUTQ.FIRST QUA TASKBLOCK ELSE                  01369000
TASKFOUND:-NONE;                                                           01370000
IF TASKFOUND == NONE THEN                                                  01371000
BEGIN                                                                      01372000 B81
   IF FLAG THEN                                                              01373000
   BEGIN                                                                     01374000 B82
      WRITE("FETCH IS NEG FOR PROCESS    ",CALLINGPROCESS.T,                   01375000
      CALLINGPROCESS.PI);                                                      01376000
      OUTIMAGE;                                                                01377000
   END;                                                                      01378000 E82
   GO TO NEGATIVE;                                                           01379000
END;                                                                       01380000 E81
IF (J GE TASKFOUND.TASK(2) AND TASKFOUND =/= NONE) THEN                    01381000
BEGIN                                                                      01382000 B83
   IF FLAG THEN                                                              01383000
   BEGIN                                                                     01384000 B84
      WRITE("FETCH IS POS FOR PROCESS    ",CALLINGPROCESS.T,                   01385000
      CALLINGPROCESS.PI);                                                      01386000
      OUTIMAGE;                                                                01387000
   END;                                                                      01388000 E84
   FOUND: TASKFOUND.OUT;                                                     01389000
   HOLD(33.2);                                                               01390000
   COMMENT***PROLO ACTIVITY TIME****;                                        01391000
   POSITIVE: CALLINGPROCESS.PROLO:=FALSE;                                    01392000
   QLENGTH(CALLINGPROCESS);                                                  01393000
   ACTIVATE CALLINGPROCESS.PROLOQ.FIRST ;                                    01394000
   HOLD(44.4);                                                               01395000
   COMMENT***REMAINDER OF RE-ENTRANT ACTIVITY TIME WHEN                      01396000
   REQUIRED TASK IS FOUND***;                                                01397000
   LOADTASK;                                                                 01398000
   FREELO1;                                                                  01399000
   MYCPU.CCS:=1;                                                             01400000
   PAOVERHEAD := PAOVERHEAD + TIME - STARTTIME;                              01401000
   IF ( CALLINGPROCESS.PI>30 AND CALLINGPROCESS.PI<50 ) THEN                 01402000
   FBOVERHEAD := FBOVERHEAD + TIME - STARTTIME;                              01403000
   IF LPAQ.FIRST == NONE THEN ACTIVATE CALLINGPROCESS                        01404000
   AFTER CURRENT ELSE                                                        01405000
   IF (LPAQ.FIRST =/= NONE AND INTRIP.LPA ==                                 01406000
   (THIS PROCESSALLOCATOR)) THEN ACTIVATE LPAQ.FIRST                         01407000
   AFTER CURRENT                                                             01408000
   ELSE                                                                      01409000
   IF (LPAQ.FIRST =/= NONE AND INTRIP.LPA =/=                                01410000
   (THIS PROCESSALLOCATOR)) THEN ACTIVATE CALLINGPROCESS                     01411000
   AFTER CURRENT;                                                            01412000
   PASSIVATE;                                                                01413000
   GOTO PASTART;                                                             01414000
END**FETCHN**TASK FOUND***                                                 01415000 E83
ELSE                                                                       01416000
NEGATIVE: BEGIN                                                            01417000 B85
   HOLD(IF CALLINGPROCESS.INPUTQ.EMPTY THEN 8.6                              01418000
   ELSE 16.0);                                                               01419000
   COMMENT***PROLO ACTIVITY TIME******;                                      01420000
   UNSUC: CALLINGPROCESS.PROLO:=FALSE;                                       01421000
   QLENGTH(CALLINGPROCESS);                                                  01422000
   ACTIVATE CALLINGPROCESS.PROLOQ.FIRST ;                                    01423000
   MYCPU.CCS:=0;                                                             01424000
      COMMENT***SET CC 0 ***;                                                01425000
   PAOVERHEAD := PAOVERHEAD + TIME - STARTTIME;                              01426000
   IF ( CALLINGPROCESS.PI>30 AND CALLINGPROCESS.PI<50 ) THEN                 01427000
   FBOVERHEAD := FBOVERHEAD + TIME - STARTTIME;                              01428000
   IF LPAQ.FIRST == NONE THEN ACTIVATE CALLINGPROCESS                        01429000
   AFTER CURRENT ELSE                                                        01430000
   IF (LPAQ.FIRST =/= NONE AND INTRIP.LPA ==                                 01431000
   (THIS PROCESSALLOCATOR)) THEN ACTIVATE LPAQ.FIRST                         01432000
   AFTER CURRENT                                                             01433000
   ELSE                                                                      01434000
   IF (LPAQ.FIRST =/= NONE AND INTRIP.LPA =/=                                01435000
   (THIS PROCESSALLOCATOR)) THEN ACTIVATE CALLINGPROCESS                     01436000
   AFTER CURRENT;                                                            01437000
   PASSIVATE;                                                                01438000
   GOTO PASTART;                                                             01439000
END**TASK NOT FOUND***;                                                    01440000 E85
                                                                           01441000
                                                                           01442000
SEEKING:                                                                   01443000
STARTTIME := TIME;                                                         01444000
HOLD(12.8);                                                                01445000
COMMENT***RE-ENTRANT TIME WHEN NO TASK FOUND****;                          01446000
PROLO1(CALLINGPROCESS);                                                    01447000
COMMENT**********************************;                                 01448000
IF FLAG THEN                                                               01449000
BEGIN                                                                      01450000 B86
   WRITE("CALL TO SEEK BY PROCESS    ",CALLINGPROCESS.T,                     01451000
   CALLINGPROCESS.PI);                                                       01452000
   OUTTV("ON CPU NO. =",MYCPU.NUMBER);                                       01453000
   OUTTV("AT RUNTIME =",TIME);                                               01454000
   OUTTV("AND INPQ TSKS=",CALLINGPROCESS.INPUTQ.CARDINAL);                   01455000
```

```
     OUTIMAGE;                                                               01456000
   END;                                                                    01457000 E86
   COMMENT**************************************;                          01458000
   J:=CALLINGPROCESS.M;                                                    01459000
   IF CALLINGPROCESS.INPUTQ.FIRST IS TASKBLOCK THEN                        01460000
   TASKFOUND:-CALLINGPROCESS.INPUTQ.FIRST QUA TASKBLOCK ELSE               01461000
   TASKFOUND:-NONE;                                                        01462000
   K:=0;                                                                   01463000
   IF TASKFOUND == NONE THEN GOTO NOTASK;                                  01464000
   WHILE (TASKFOUND.TASK(2) NE J AND TASKFOUND =/= NONE                    01465000
   AND TASKFOUND.SUC IS TASKBLOCK) DO                                      01466000
   BEGIN                                                                   01467000 B87
     TASKFOUND:-TASKFOUND.SUC QUA TASKBLOCK;                                 01468000
     K:=K+1;                                                                 01469000
   END;                                                                    01470000 E87
   IF TASKFOUND =/= NONE THEN                                              01471000
   BEGIN                                                                   01472000 B88
     TASKFOUND.OUT;                                                          01473000
     IF FLAG THEN                                                            01474000
     BEGIN                                                                   01475000 B89
       IF FLAG THEN WRITE("SEEK IS POS FOR PROCESS   ",CALLINGPROCESS.T,       01476000
       CALLINGPROCESS.PI);                                                     01477000
       OUTIMAGE;                                                               01478000
     END;                                                                    01479000 E89
     HOLD(33.2+10.6*K);                                                      01480000
     COMMENT***PROLO ACTIVITY TIME****;                                      01481000
     GOTO POSITIVE;                                                          01482000
   END***TASK FOUND***                                                     01483000 E88
   ELSE                                                                    01484000
   NOTASK: BEGIN                                                           01485000 B90
     IF FLAG THEN                                                            01486000
     BEGIN                                                                   01487000 B91
       WRITE("SEEK IS NEG FOR PROCESS   ",CALLINGPROCESS.T,                    01488000
       CALLINGPROCESS.PI);                                                     01489000
       OUTIMAGE;                                                               01490000
     END;                                                                    01491000 E91
     HOLD(8.6+10.6*K);                                                       01492000
     COMMENT***ACTIVITY TIME OF ENGAGING PROLO****;                          01493000
     GOTO UNSUC;                                                             01494000
   END** TASK NOT FOUND***;                                                01495000 E90
                                                                           01496000
                                                                           01497000
   BLOCKING:                                                               01498000
   STARTTIME := TIME;                                                      01499000
   HOLD(12.8);                                                             01500000
   COMMENT***RE-ENTRANT TIME FOR NO TASK,WHEN A TASK IS                    01501000
   FOUND (57.2-12.8=34.4) MUST BE ADDED*****;                              01502000
   PROLO1(CALLINGPROCESS);                                                 01503000
   COMMENT**************************************;                          01504000
   IF FLAG THEN                                                            01505000
   BEGIN                                                                   01506000 B92
     WRITE("CALL TO BLOCK BY PROCESS  ",CALLINGPROCESS.T,                    01507000
     CALLINGPROCESS.PI);                                                     01508000
     OUTTV("ON CPU NO.  =",MYCPU.NUMBER);                                    01509000
     OUTTV("AT RUNTIME =",TIME);                                             01510000
     OUTIMAGE;                                                               01511000
   END;                                                                    01512000 E92
   COMMENT**************************************;                          01513000
   J:=CALLINGPROCESS.S;                                                    01514000
   IF CALLINGPROCESS.INPUTQ.FIRST IS TASKBLOCK THEN                        01515000
   TASKFOUND:-CALLINGPROCESS.INPUTQ.FIRST QUA TASKBLOCK ELSE               01516000
   TASKFOUND:-NONE;                                                        01517000
   IF TASKFOUND == NONE THEN GO TO FAILURE;                                01518000
   IF (J GE TASKFOUND.TASK(2) AND TASKFOUND =/= NONE) THEN                 01519000
   BEGIN                                                                   01520000 B93
     IF FLAG THEN                                                            01521000
     BEGIN                                                                   01522000 B94
       WRITE("CALL TO BLOCK IS POS BY PROCESS  ",CALLINGPROCESS.T,             01523000
       CALLINGPROCESS.PI);                                                     01524000
       OUTIMAGE;                                                               01525000
     END;                                                                    01526000 E94
     GO TO FOUND;                                                            01527000
   END                                                                     01528000 E93
   ELSE                                                                    01529000
   FAILURE: BEGIN                                                          01530000 B95
     IF FLAG THEN                                                            01531000
     BEGIN                                                                   01532000 B96
       WRITE("CALL TO BLOCK IS NEG FOR PROCESS  ",CALLINGPROCESS.T,            01533000
       CALLINGPROCESS.PI);                                                     01534000
       OUTIMAGE;                                                               01535000
     END;                                                                    01536000 E96
     FOR I:=1 STEP 1 UNTIL 7 DO                                              01537000
        CALLINGPROCESS.PD(I+24):=MYCPU.G(I);                                 01538000
     COMMENT***NEST GRS IN PD(24-31)***;                                     01539000
     CALLINGPROCESS.PD(8):=4;                                                01540000
     CALLINGPROCESS.PD(9):=J;                                                01541000
     MYCPU.CCS:=0;                                                           01542000
     HOLD(IF TASKFOUND==NONE THEN 55.4 ELSE 62.8);                           01543000
     COMMENT***PROLO ACTIVITY TIME*******;                                   01544000
     CALLINGPROCESS.PROLO:=FALSE;                                            01545000
     ACTIVATE CALLINGPROCESS.PROLOQ.FIRST ;                                  01546000
     HOLD(8.0);                                                              01547000
     COMMENT****REMAINDER OF RE-ENTRANT TIME******;                          01548000
     CPUSCHED;                                                               01549000
     GOTO RUNSELCPROC;                                                       01550000
   END**BLOCK N  NOT SUCCESS***;                                           01551000 E95
                                                                           01552000
```

```
FBLOCKING:                                                          01553000
STARTTIME := TIME;                                                  01554000
HOLD(12.8);                                                         01555000
COMMENT***RE-ENTRANT TIME FOR NO TASK,WHEN A TASK IS                01556000
FOUND (57.2-12.8=34.4) MUST BE ADDED*****;                          01557000
PROLOI(CALLINGPROCESS);                                             01558000
COMMENT*************************************;                       01559000
IF FLAG THEN                                                        01560000
BEGIN                                                               01561000 B97
  WRITE("CALL TO FBLCK BY PROCESS  ",CALLINGPROCESS.T,                01562000
  CALLINGPROCESS.PI);                                                 01563000
  OUTTV("CN CPU NO.  =",MYCPU.NUMBER);                                01564000
  OUTTV("AT RUNTIME =",TIME);                                         01565000
  OUTIMAGE;                                                           01566000
END;                                                                01567000 E97
COMMENT*************************************;                       01568000
J:=CALLINGPROCESS.W;                                                01569000
IF CALLINGPROCESS.INPUTQ.FIRST IS TASKBLOCK THEN                    01570000
TASKFOUND:-CALLINGPROCESS.INPUTQ.FIRST QUA TASKBLOCK ELSE           01571000
TASKFOUND:-NONE;                                                    01572000
IF TASKFCUND == NONE THEN GO TO FBLFAIL;                            01573000
IF (15 GE TASKFOUND.TASK(2) AND TASKFOUND =/= NONE) THEN            01574000
BEGIN                                                               01575000 B98
  IF FLAG THEN                                                        01576000
  BEGIN                                                               01577000 B99
    WRITE("CALL TO FBLOCK IS POS BY PROCESS  ",CALLINGPROCESS.T,        01578000
    CALLINGPROCESS.PI);                                                 01579000
    OUTIMAGE;                                                           01580000
  END;                                                                01581000 E99
  GO TO FOUND;                                                        01582000
END                                                                 01583000 E98
ELSE                                                                01584000
FBLFAIL: BEGIN                                                      01585000 B100
  IF FLAG THEN                                                        01586000
  BEGIN                                                               01587000 B101
    WRITE("CALL TO FBLCK IS NEG FOR PROCESS  ",CALLINGPROCESS.T,        01588000
    CALLINGPROCESS.PI);                                                 01589000
    OUTIMAGE;                                                           01590000
  END;                                                                01591000 E101
  FOR I:=1 STEP 1 UNTIL 7 DO                                          01592000
     CALLINGPROCESS.PD(I+24):=MYCPU.G(I);                             01593000
  COMMENT***NEST GRS IN PD(24-31)***;                                 01594000
  CALLINGPROCESS.PD(8):=4;                                            01595000
  CALLINGPROCESS.PD(9):=J;                                            01596000
  MYCPU.CCS:=0;                                                       01597000
  CALLINGPROCESS.PROLO:=FALSE;                                        01598000
  ACTIVATE CALLINGPROCESS.PROLOQ.FIRST ;                              01599000
  CPUSCHED;                                                           01600000
  GOTO RUNSELCPROC;                                                   01601000
END**FBLOCK N  NOT SUCCESS***;                                      01602000 E100
                                                                    01603000
                                                                    01604000
                                                                    01605000
HND:                                                                01606000
STARTTIME := TIME;                                                  01607000
HOLD(72.4);                                                         01608000
COMMENT**<HAND>RE-ENTRANT TIME******;                               01609000
OGTI:=CALLINGPROCESS.G(0);                                          01610000
COMMENT*************************************;                       01611000
IF FLAG THEN                                                        01612000
BEGIN                                                               01613000 B102
  WRITE("CALL TO HAND BY PROCESS  ",CALLINGPROCESS.T,                 01614000
  CALLINGPROCESS.PI);                                                 01615000
  OUTTV("CN CPU NO.=",MYCPU.NUMBER);                                  01616000
  OUTTV("AT RUNTIME =",TIME);                                         01617000
  OUTTV("OGTI",CALLINGPROCESS.G(0));                                  01618000
  OUTTV("AND CALL INDEX=",CALLINGPROCESS.CIX);                        01619000
  OUTIMAGE;                                                           01620000
END;                                                                01621000 E102
COMMENT*************************************;                       01622000
IF CALLINGPROCESS.TIT(OGTI,2)=0 THEN ERROR(3)                       01623000
ELSE                                                                01624000
BEGIN                                                               01625000 B103
  FREELO2;                                                            01626000
  INSPECT CALLINGPROCESS DO                                           01627000
  BEGIN                                                               01628000 B104
    FREETASKBLOCK.TASK(2):=TIT(OGTI,4);                                 01629000
    CALLEDPROCESS:-P(TIT(OGTI,2));                                      01630000
    FREETASKBLOCK.TASK(3):=TIT(OGTI,3);                                 01631000
    FREETASKBLOCK.TASK(1):=PI;                                          01632000
    COMMENT*************************************                        01633000
    TIT(CGTI,1) STORES OGTI,                                            01634000
    TIT(CGTI,2) STORES CALLED PROCESS INDEX,                            01635000
    TIT(CGTI,3) STORES ICTI I.E. OGTI OF CALLED PROCESS,                01636000
    TIT(OGTI,4) STORES THE TASK PRIORITY,                               01637000
    ****************COMMENT ENDS****************;                       01638000
  END;                                                                01639000 E104
  FOR I:=4 STEP 1 UNTIL 10 DO                                         01640000
     FREETASKBLOCK.TASK(I):=CALLINGPROCESS.G(I-3);                    01641000
  FOR I:=0 STEP 1 UNTIL 7 DO                                          01642000
  CALLINGPROCESS.G(I):=0;                                             01643000
  COMMENT****RESET REGISTERS******;                                   01644000
  TASKHANDED:-FREETASKBLOCK;                                          01645000
  COMMENT**TASKHANDED IS REF USED INHANDTASK****;                     01646000
  HANDTASKANDSETSTATE;                                                01647000
  IF CALLEDPROCESS.PD(8)=3 THEN                                       01648000
  BEGIN                                                               01649000 B105
```

```
          Y:=1.0;
          SYSCHED;
          IF INTRIP.SUSPOINT THEN Z:=1    ELSE
          Z:=0;
          HOLD(Y*(18.6+4.0*(CALLINGPROCESS.PI
          //16)+4.8*Z));
          COMMENT***REMAINDER OF RE-ENTRANT TIME****;
        END;                                                            E105
        TASKHANDED:-NONE;
        PAOVERHEAD := PAOVERHEAD + TIME - STARTTIME;
        COMMENT***RESET TASKHANDED***;
        IF LPAQ.FIRST == NONE THEN ACTIVATE
        CALLINGPROCESS AFTER CURRENT ELSE
        IF (LPAQ.FIRST =/= NONE AND INTRIP.LPA ==
        (THIS PROCESSALLOCATOR)) THEN ACTIVATE LPAQ.FIRST
        AFTER CURRENT
        ELSE
        IF (LPAQ.FIRST =/= NONE AND INTRIP.LPA =/=
        (THIS PROCESSALLOCATOR)) THEN ACTIVATE CALLINGPROCESS
        AFTER CURRENT;
        PASSIVATE;
        GOTO PASTART;
      END;                                                              E103


      SELV:
      STARTTIME := TIME;
      HOLD(61.8);
      COMMENT***RE-ENTRANT ACTIVITY TIME*****;
      FREELO2;
      FREETASKBLOCK.TASK(2):=CALLINGPROCESS.D;
      COMMENT*********************************;
      IF FLAG THEN
      BEGIN                                                             B106
        WRITE("CALL TO SELF BY PROCESS  ",CALLINGPROCESS.T,
        CALLINGPROCESS.PI);
        OUTTV("ON CPU NO.=",MYCPU.NUMBER);
        OUTTV("AT RUNTIME =",TIME);
        OUTIMAGE;
      END;                                                              E106
      COMMENT***********************************;
        COMMENT**SET TASK PRIORITY=N***;
      FREETASKBLOCK.TASK(1):=CALLINGPROCESS.G(0);
      TASKHANDED:-FREETASKBLOCK;
        COMMENT**COPY OGTI AS THE ICTI***;
      FOR I:=4 STEP 1 UNTIL 10 DO
      FREETASKBLOCK.TASK(I):=CALLINGPROCESS.G(I-3);
      CALLEDPROCESS:-CALLINGPROCESS;
      FOR I:=0 STEP 1 UNTIL 7 DO
      CALLINGPROCESS.G(I):=0;
      HANDTASKANDSETSTATE;
      PAOVERHEAD := PAOVERHEAD + TIME - STARTTIME;
      IF LPAQ.FIRST == NONE THEN ACTIVATE CALLINGPROCESS
      AFTER CURRENT ELSE
      IF (LPAQ.FIRST =/= NONE AND INTRIP.LPA ==
      (THIS PROCESSALLOCATOR)) THEN ACTIVATE LPAQ.FIRST
      AFTER CURRENT
      ELSE
      IF(LPAQ.FIRST =/= NONE AND INTRIP.LPA=/=
      (THIS PROCESSALLOCATOR)) THEN ACTIVATE CALLINGPROCESS
      AFTER CURRENT;
      PASSIVATE;
      GOTO PASTART;
    END***OF PROCESSALLOCATOR***;                                       E34



        COMMENT********** CLOCK INTERRUPT ******************
        *
        * DESCRIPTION: SIMULATES THE ARRIVAL OF A CLOCK INTERR-
        *                   UPT EVERY 10 MSEC
        * FUNCTION: CREATES CPUS IN SYSTEM AND BACKGROUND
        *                PROCESSES RUNNING ON THEM.GENERATE A CLOCK
        *                INTERRUPT EVERY 10 MSEC AND ALERTS INTERRUPT
        *                TRIPLICATES
        * VARIABLES:
        * C(1:NUM): C(I) REFERS TO CPU NO. I
        * B(1:NUM): B(I)  "       " BACKGROUND WHOSE PI IS
        *                115+I
        * INPUT TO: CPU,INTERRUPT TRIPLICATES
        * OUTPUT FROM: NONE
        * ACTIVATES: CPU,BACKGROUND,INTERRUPT TRIPLICATES
        * ACTIVATED BY: NONE
        ******************** COMMENT ENDS *******************;

            PROCESS CLASS CLOCKINTERRUPT;
            BEGIN                                                       B107
      REF(CPU) ARRAY C(1:NUM);
      FOR I:=1 STEP 1 UNTIL NUM DO
      BEGIN                                                             B108
        C(I):-NEW CPU(I);
        P(115+I):-NEW BACKGROUND(115+I);
        P(115+I).FD(8):=1;
        P(115+I).T:-COPY("BACKGROUND");
        P(115+I).INPUTQ:-NEW HEAD;
        P(115+I).FROLOQ:-NEW HEAD;
        C(I).STARTTIME:=TIME;
        C(I).CURF:-P(115+I);                                                            01747000
        P(115+I).RELEVANTCPU:-C(I);                                                     01748000
        COMMENT****BACKGROUND PROCESS START WITH PI=116**;                              01749000
        ACTIVATE C(I) ;                                                                 01750000
        ACTIVATE P(115+I) ;                                                             01751000
      END;                                                                            01752000 E108
      INTRIP.LCPU:-C(NUM);                                                            01753000
                    WHILE TIME<SIMPERIOD DO                                           01754000
                  BEGIN                                                               01755000 B109
            INTRIP.CLOCKINT:=TRUE;                                                      01756000
        IF FLAG THEN OUTTV("CLOCKINT AT TIME=",TIME);                                   01757000
                            ACTIVATE INTRIP AFTER CURRENT;                              01758000
                            HOLD(10000.0);                                              01759000
                  END;                                                                01760000 E109
             END**CLOCKINTERRUPT***;                                                 01761000 E107
                                                                                     01762000
                                                                                     01763000
                                                                                     01764000
    COMMENT************************INITIATOR PROCESS*******                          01765000
    *                                                                                01766000
    *FUNCTION: TC INITIATE NOFBLCALL  AND WITHFBLCALL                                01767000
    *        PROCESSES BY HANDING CLOSE FILE TASKS                                   01768000
    *VARIABLES: NCNE                                                                 01769000
    *ACTIVATES: NCNE                                                                 01770000
    *ACTIVATED BY: NONE                                                              01771000
    **********************************************;                                  01772000
                                                                                     01773000
    AP CLASS INITIATOR;                                                              01774000
    BEGIN                                                                            01775000 B110
      INTEGER I, C;                                                                    01776000
      C := 0;                                                                          01777000
      A:                                                                               01778000
      IF C = 1 THEN GO TO E1                                                           01779000
      ELSE                                                                             01780000
      FOR I := 1 STEP 1 UNTIL NUMOFPROCESSES DO                                        01781000
      BEGIN                                                                            01782000 B111
        G(0) := I;                                                                       01783000
        G(2) := 4;                                                                       01784000
        HAND;                                                                            01785000
        COMMENT*** CLOSE FILE REQUEST TO INITIATE PROCESS INSTANCE****;                  01786000
      END;                                                                             01787000 E111
      C := C + 1;                                                                      01788000
      E1:                                                                              01789000
      BLOCK(10);                                                                       01790000
      GO TO A;                                                                         01791000
    END**** INITIATOR PROCESS ******;                                                01792000 E110
                                                                                     01793000
                                                                                     01794000
                                                                                     01795000
        COMMENT***************** BACKGROUND ******************                       01796000
        *                                                                            01797000
        * FUNCTION: OCCUPY A CPU WHEN IDLE TO SIMULATE BACKGROUND                    01798000
        *           WORK                                                             01799000
        * VARIABLES: NONE                                                            01800000
        * ACTIVATES:  "                                                              01801000
        * ACTIVATED BY: NONE                                                         01802000
        ******************** COMMENT ENDS******************;                          01803000
            AP CLASS BACKGROUND;                                                     01804000
            BEGIN                                                                    01805000 B112
      HLD: HOLD(SIMPERIOD-TIME);                                                       01806000
      IF TIME< SIMPERIOD THEN GOTO HLD ELSE                                            01807000
      BEGIN                                                                            01808000 B113
        RELEVANTCPU.BKGTIME:=RELEVANTCPU.BKGTIME+TIME-RELEVANTCPU.STARTTIME;             01809000
        PASSIVATE;                                                                       01810000
      END;                                                                             01811000 E113
            END;                                                                     01812000 E112
                                                                                     01813000
                                                                                     01814000
                                                                                     01815000
    COMMENT****************** DELAYSTAT *******************                          01816000
    *                                                                                01817000
    * DESCRIPTICN: CALCULATES AND REPORTS BASIC STATISTICS                           01818000
    *              OF DELAYS OF INTEREST FOR CALLS THROUGH                           01819000
    *              A MEMBER OF SYSTEM X FAMILY OF EXCHANGES                          01820000
    *                                                                                01821000
    * FUNCTION:                                                                      01822000
    *        CALCULATES AND PRINTS THE MIN, MAX, AND AVERAGE VALUES                  01823000
    *        FCR EACH DELAY OF INTEREST                                              01824000
    *                                                                                01825000
    * VARIABLES:                                                                     01826000
    *      TITLE   USER SUPPLIED TITLE FOR A PARTICULAR DELAY                        01827000
    *      OBS     NUMBER OF OBSERVATION                                             01828000
    *      SUM     SUM OF OBSERVATIONS VALUES                                        01829000
    *      MIN     MINIMUM SAMPLE VALUE                                              01830000
    *      MAX     MAXIMUM SAMPLE VALUF                                              01831000
    *                                                                                01832000
    * PROCEDURES:                                                                    01833000
    *    UPDATE(X)   ADDS 1 TO OBS                                                   01834000
    *                ADDS X TO SUM                                                   01835000
    *                UPDATES MAX VALUE                                               01836000
    *                UPDATES MIN VALUE                                               01837000
    *   REPORT       PRINTS TITLE , MAXIMUM, MINIMUM AND AVERAGE VALUES              01838000
    *                FOR EACH DELAY OF INTEREST                                      01839000
    *                                                                                01840000
    * INPUT TO:                                                                      01841000
    *                FINAL RUN REPORT                                                01842000
    * OUTPUT FRCM:                                                                   01843000
```

```
*                   DELAYS SAMPLE VALUES
* ACTIVATES:
*                   NONE
* ACTIVATFD BY:
*                   NONE
*
*************************************************************;

CLASS DELAYSTAT(TITLF);
VALUE TITLF; TEXT TITLE;
BEGIN                                                                B114
  INTEGER CBS;
  REAL SUM, MIN, MAX;

  PROCEDURE UPDATE(X); REAL X;
  BEGIN                                                              B115
    OBS := CBS + 1;
    SUM := SUM + X;
    IF OBS = 1 THEN MIN := MAX := X
    ELSE
    IF X < MIN THEN MIN := X
    ELSE
    IF X > MAX THEN MAX := X;
  END******* UPDATE *******;                                         E115

  PROCEDURE REPORT;
  BEGIN                                                              B116
    OUTIMAGE;
    OUTTEXT("DELAY STATISTICS FOR RESPCNSETIME  ");
    OUTLINE(TITLE);
    OUTTVR("MAXIMUM DELAY IN MICROSECONDS = ", MAX);
    OUTTVR("MINIMUM DELAY IN MICROSECONDS = ", MIN);
    OUTTVR("AVERAGE DELAY IN MICROSECONDS = ", (SUM/OBS));
  END****** REPORT ******;                                           E116

END****** DELAYSTAT ********************************;                E114


COMMENT********************** WITHFBLCALL PROCESS *********
*
* DESCRIPTION:
*      A SUITE OF PROCESS INSTANCES USED FOR THE EVALUATION
*      OF A PROPOSED NEW CALL TO THE PROCESS ALLOCATOR -
*      'FBLOCK<P>'- FOR RELEASE 2 OF MK II BL SYSTEM. THEY
*      USE THIS NEW CALL HERE
*      THEY ARE ADAPATION OF RASH PROCESS INSTANCES
*
* FUNCTION:
*     THE PRCCESS INSTANCES ARE USED TO LOAD THE SYSTEM BY
*          ISSUING REQUESTS TO THE STORAGE ALLOCATOR TO OPEN
*     AND CLCSE THEIR RESPECTIVE FILES
*
* VARIABLES:
*    COUNTER   TO REGISTER NO. OF TIMES AN INSTANCE TRAVERSES
*              ITS LOOP
* INPUT TO:
*    STORAGE ALLOCATOR
* OUTPUT FROM:
*    STORAGE ALLOCATOR, INTIM, INITIATOR
* ACTIVATES:
*    NONE
* ACTIVATED BY:
*    PROCESS ALLOCATOR
*
*********************************************************;

AP CLASS WITHFBLCALL;

BEGIN                                                                B117
  INTEGER COUNTER;
  A: FBLCCK(5);
  IF RELEVANTCPU.G(2) = 0 THEN GO TO A
  ELSE
  BEGIN                                                              B118
    IF RELEVANTCPU.G(2) = 4 THEN GO TO CLOSEFILE
    ELSE
    IF RELEVANTCPU.G(2) = 5 THEN GO TO OPENFILE
    ELSE
    OUTLINE("***** ERROR ***NOT AN OPEN OR CLOSE FILE MESSAGE RECEIVED*");
  END****** CF A TASK BEING RECEIVED ******;                         E118

  CLOSEFILE:
  HOLD(579.0);
  G(0) := 1;
  G(1) := PI;
  G(2) := 5;
  HAND;
  COMMENT ***** CLOSE FILE REQUEST TO SA ****;
  HOLD(459.0);
  COUNTER := CCUNTER + 1;
  IF FLAG THEN OUTTV("COUNTER OF WITHFBLCALL = ",COUNTER);
  GO TO A;

  OPENFILE:
  HOLD(459.0);
  G(0) := 1;
```

```
    G(1) := PI;                                                                  01941000
    G(2) := 4;                                                                   01942000
    HAND;                                                                        01943000
    COMMENT**** OPEN FILE REQUEST TO SA ****;                                    01944000
    HOLD(579.0);                                                                 01945000
    COUNTER := COUNTER + 1;                                                      01946000
    IF FLAG THEN OUTTV("COUNTER OF WITHFBLCALL = ",COUNTER);                     01947000
    GO TO A;                                                                     01948000
  END********* WITHFBLCALL *****;                                               01949000 E117
                                                                                01950000
                                                                                01951000
                                                                                01952000
  COMMENT********************** NOFBLCALL PROCESS *********                     01953000
  *                                                                             01954000
  * DESCRIPTION:                                                                01955000
  *       A SUITE OF PROCESS INSTANCES USED FOR THE EVALUATION                  01956000
  *       OF A PROPOSED NEW CALL TO THE PROCESS ALLOCATOR -                     01957000
  *       'FBLOCK<P>'- FOR RELEASE 2 CF MK II BL SYSTEM.                        01958000
  *       THEY ARE ADAPATION OF RASH PROCESS INSTANCES                          01959000
  *                                                                             01960000
  * FUNCTION:                                                                   01961000
  *      THE PROCESS INSTANCES ARE USED TO LOAD THE SYSTEM BY                   01962000
  *           ISSUING REQUESTS TO THE STORAGE ALLOCATOR TO OPEN AND             01963000
  *      AND CLOSE THEIR RESPECTIVE FILES                                       01964000
  *                                                                             01965000
  * VARIABLES:                                                                  01966000
  *     COUNTER  TO REGISTER NO. OF TIMES AN INSTANCE TRAVERSES                 01967000
  *              ITS LOOP                                                       01968000
  * INPUT TO:                                                                   01969000
  *     STORAGE ALLOCATOR                                                       01970000
  * OUTPUT FROM:                                                                01971000
  *    STORAGE ALLOCATOR, INTIM, INITIATOR                                      01972000
  * ACTIVATES:                                                                  01973000
  *    NONE                                                                     01974000
  * ACTIVATED BY:                                                               01975000
  *    PROCESS ALLOCATOR                                                        01976000
  *                                                                             01977000
  ************************************************************;                01978000
                                                                                01979000
  AP CLASS NCFBLCALL;                                                           01980000
                                                                                01981000
  BEGIN                                                                         01982000 B119
    INTEGER CCUNTER;                                                              01983000
    A:                                                                            01984000
    FETCH(15);                                                                    01985000
    IF CC = 0 THEN                                                                01986000
    BEGIN                                                                         01987000 B120
      BLOCK(5);                                                                     01988000
      COMMENT**** THE INSTANCE BLOCKS ITSELF WAITING FOR A MESSAGE**;               01989000
      GO TO A;                                                                      01990000
    END                                                                           01991000 E120
    ELSE                                                                          01992000
    BEGIN                                                                         01993000 B121
      IF RELEVANTCPU.G(2) = 4 THEN GO TO CLOSEFILE                                  01994000
      ELSE                                                                          01995000
      IF RELEVANTCPU.G(2) = 5 THEN GO TO OPENFILE                                   01996000
      ELSE                                                                          01997000
      BEGIN                                                                         01998000 B122
        OUTLINE("***ERROR** NOT AN OPEN OR CLOSE FILE MESSAGE RECEIVED*");            01999000
        GO TC A;                                                                      02000000
      END ** OF RECEIVING UNBLOCKING TASK FROM INTIM***;                            02001000 E122
    END **OF CCNDITION CODE +VE **;                                               02002000 E121
    CLOSEFILE:                                                                    02003000
    HOLD(579.0);                                                                  02004000
    G(0) := 1;                                                                    02005000
    G(1) := PI;                                                                   02006000
    G(2) := 5;                                                                    02007000
    HAND;                                                                         02008000
    COMMENT*** CLOSE FILE REQUEST TO SA ***;                                      02009000
    HOLD(459.0);                                                                  02010000
    COUNTER := COUNTER + 1;                                                       02011000
    IF FLAG THEN OUTTV("COUNTER OF NOFBLCALL = ",COUNTER);                        02012000
    GO TO A;                                                                      02013000
    OPENFILE:                                                                     02014000
    HOLD(459.0);                                                                  02015000
    G(0) := 1;                                                                    02016000
    G(1) := PI;                                                                   02017000
    G(2) := 4;                                                                    02018000
    HAND;                                                                         02019000
    COMMENT*** CPEN FILE REQUEST TO SA **;                                        02020000
    HOLD(579.0);                                                                  02021000
    COUNTER := COUNTER +1;                                                        02022000
    IF FLAG THEN OUTTV("COUNTER OF NOFBLCALL = ",COUNTER);                        02023000
    GO TC A;                                                                      02024000
  END ***** NOFBLCALL ******;                                                   02025000 E119
                                                                                02026000
                                                                                02027000
                                                                                02028000
  COMMENT***************** RECORD PROCESS **************                        02029000
  *                                                                             02030000
  * DESCRIPTICN:                                                                02031000
  *     FOR EVERY CALL GENERATED AND ENJECTED IN THE SYSTEM                     02032000
  *     THERE EXISTS A CALL RECORD PRCCESS FOR THE DURATION                     02033000
  *     OF THE CALL                                                             02034000
  *                                                                             02035000
  * FUNCTICN:                                                                   02036000
  *     INDICATES TO THE I/C SIS H/W ASSOCIATED WITH THE                        02037000
```

```
   *    CALL,THE CALL ARRIVAL AND WHEN SUBSCRIBER CLEARS
   *    DOWN.
   *    SELECTS THE STATE OF THE GENERATED CALL .
   *    GENERATES AN O/G SIS H/W INSTANCE IF STATE OF CALL > 1
   *    INDICATES TO O/G SIS H/W ASSOCIATED WITH THE CALL
   *    WHEN THE SUBSCRIBER ANSWERS.
   *
   * VARIABLES:
   *    SOC TC INDICATE 1 OUT OF 3 STATES FOR THIS CALL
   *    MYINCSISHW A REF. VARIABLE TO THE CORRESPONDING I/C
   *              SIS H/W
   *    MYOUTSISHW  A REF. VARIABLE TO CORRESPONDING O/G SIS H/W.
   *    CIRCUITNUM  REFERS TO THE CALL CIRCUIT NUMBER I.E. I/C CCT.
   *    OUTGCCTNUM  O/G CCT. IDPROCESS
   * INPUT FROM  I/C SIS H/W,CALL GENERATOR
   * OUTPUT TC   I/C SIS H/W,O/G SIS H/W.
   * ACTIVATES I/C SIS H/W, O/G SIS H/W
   * ACTIVATED BY I/C SIS H/W, O/G SIS H/W
   *
   ************************************************************;

   PROCESS CLASS CALLRECORD(CIRCUITNUM);
   COMMENT============================;
   INTEGER CIRCUITNUM;
   BEGIN                                                      B123
     INTEGER OUTGCCTNUM, K, SOC;
     REF(INCSISHW) MYINCSISHW;
     REF(OUTGSISHW) MYOUTSISHW;
     COMMENT**** NOW STARTS THE ACTION PART **********;
     MYINCSISHW :- SISHWINC(CIRCUITNUM);
     SOC := RANDINT(1,3,SD1);
     OUTTV("!!! S.O.C. SAMPLED HAS THE NO. ",SOC);
     OUTTV("!!! FOR ASSOCIATED I/C CCT. NO. ",CIRCUITNUM);
     MYINCSISHW.SOC := SOC;
     IF SOC GT 1 THEN
     BEGIN                                                    B124
       COMMENT*** SOC GT 1***;
       K := RANDINT(MININCCT,MAXOUTCCT,SD16);
       WHILE SISHWOUT(K) =/= NONE DO
       K := RANDINT(MININCCT,MAXOUTCCT,SD17);
       COMMENT*** K IS THE O/G CCT. NO. GENERATED***;
       SISHWOUT(K) :- NEW OUTGSISHW(K);
       COMMENT*** GENERATE A NEW INSTANCE OF OUTGSIS H/W***;
       OUTTV("!!! RELATED O/G CCT. IF S.O.C. > 1 IS NO. ",K);
       MYOUTSISHW :- SISHWOUT(K);
       OUTGCCTNUM := MYINCSISHW.OUTGCCTNUM := K;
       SISHWOUT(K).MYCALLREC :- THIS CALLRECORD;
     END *** SOC > 1 ****;                                    E124
     IF SOC = 1 THEN
     A:
     BEGIN                                                    B125
       MYINCSISHW.SEIZURE:= TRUE;
       COMMENT*** INDICATES TO I/C SIS H/W CALL ARRIVAL*****;
       ACTIVATE MYINCSISHW AFTER CURRENT;
     END                                                      E125
     ELSE
     IF SOC = 2 THEN
     B:
     BEGIN                                                    B126
       MYOUTSISHW.MESSAGE(8,1):=1;
       MYOUTSISHW.MESSAGE(8,2) := CIRCUITNUM;
       COMMENT***INDICATE TO O/G SIS H/W THAT SUBS. ANSWERS***;
       ACTIVATE MYOUTSISHW AFTER CURRENT;
     END                                                      E126
     ELSE
     IF SOC = 3 THEN
     C:
     BEGIN                                                    B127
       MYINCSISHW.SUBSCLRDOWN:=TRUE;
       COMMENT***INDICATE TO I/C SIS H/W  THAT SUBS. CLEARS DOWN**;
       ACTIVATE MYINCSISHW AFTER CURRENT;
     END;                                                     E127
     PASSIVATE;
     CALL(CIRCUITNUM) :- NONE;
     MYINCSISHW :- NONE;
     MYOUTSISHW :- NONE;
   END*** OF CALL RECORD***;                                  E123



   COMMENT************** CALL GENERATOR ****************
   *
   * DESCRIPTION:
   *    GENERATES TELEPHONE CALLS ACCORDING TO A SPECIFIE
   *     D TRAFFIC.CALLS INTER-ARRIVAL TIMES ARE -VE EXPON
   *    ENTIALY DISTRIBUTED.
   *
   * FUNCTION:
   *    ALLOCATES RANDOMLY THE NEWLY GENERATED CALL TO
   *    A FREE INCOMMING CIRCUIT
   *    CREATES NEW CALL RECORD AND INCOMMING SIS HARD
   *    WARE PROCESS INSTANCES FOR THE NEW CALL
   * VARIABLES:
   *    X A REF TO THE CREATED INCSISHW INSTANCE
   *    I AN INTEGER VARIABLE
   * INPUT FROM: NONE
   * OUTPUT TO:CALLRECORD
```

```
* ACTIVATED BY: NONE                                                    02135000
* ACTIVATES: CALLRECORD                                                 02136000
***************** COMMENT ENDS ********************;                    02137000
                                                                        02138000
PROCESS CLASS CALLGENERATOR;                                            02139000
BEGIN                                                                   02140000 B128
  INTEGER I, COUNTER;                                                     02141000
  REAL INTERVAL;                                                          02142000
  REF(INCSISHW)X;                                                         02143000
  LOOP:                                                                   02144000
  AGAIN: I := RANDINT(MININCCT,MAXOUTCCT,SD15);                           02145000
  IF SISHWINC(I) =/= NONE THEN GO TO AGAIN;                               02146000
  CALL(I) :- NEW CALLRECORD(I);                                           02147000
  SISHWINC(I) :- NEW INCSISHW(I);                                         02148000
  COMMENT***CREATE CALLRECORD AND INCSISHW FOR NEW CALL*;                 02149000
  CALL(I).MYINCSISHW :- SISHWINC(I) ;                                     02150000
  SISHWINC(I).MYCALLREC :- CALL(I);                                       02151000
  ACTIVATE CALL(I) AFTER CURRENT;                                         02152000
  COUNTER :=COUNTER + 1;                                                  02153000
  IF COUNTER GT 10 THEN                                                   02154000
  BEGIN                                                                   02155000 B129
    INTERVAL := NEGEXP(0.0000033,SD2);                                      02156000
    HOLD(INTERVAL);                                                         02157000
    COMMENT***CALL INTER-ARRIVAL TIME FOR 600E TRAFFIC**;                   02158000
  END;                                                                    02159000 E129
  COMMENT *** 10 CALLS ARE GENERATED AT START OF THE RUN ***;             02160000
  GO TO LOOP;                                                             02161000
END *** OF CALL GENERATOR PROCESS *****;                                02162000 E128
                                                                        02163000
                                                                        02164000
                                                                        02165000
COMMENT**************** I/C SIS H/W PROCESS ***************             02166000
*                                                                       02167000
* DESCRIPTION:                                                          02168000
*      EVERY CALL IN PROGRESS HAS ITS OWN I/C SIS H/W PROCESS           02169000
*      WHICH SENDS AND RECEIVES INFORMATION CONCERNING THE              02170000
*      CALL TO THE I/C SIS S/W.                                         02171000
* FUNCTION:                                                             02172000
*      SENDS SEIZURE MESSAGE TO I/C SIS S/W                             02173000
*      SENDS DIGITS DIALLED TO  I/C SIS S/W                             02174000
*      SENDS "SUBS CLEAR DOWN" MESSAGE TO I/C SIS S/W                   02175000
*                                                                       02176000
* VARIABLES:                                                            02177000
*      ANSWER  TO INDICATE "ANSWER" MESSAGE FROM I/C SIS S/W            02178000
*      SEIZURE TO INDICATE  CALL ARRIVAL                                02179000
*      SUBSCLRDOWN  "SUBS. CLEAR DOWN" MESSAGE                          02180000
*      MYCALLREC  REFERS TO CORRESPONDING CALL RECORD                   02181000
*      CCTFREE  FROM I/C SIS S/W                                        02182000
*      CCTNUM   CIRCUIT NUMBER<IDENTITY> OF I/C CALL                    02183000
*      OUTGCCTNUM THE CORRESPONDING O/G CCT. NO.                        02184000
*      SOC  TO INDICATE THE STATE OF CALL GENERATED                     02185000
*      RESPONSETIME AN ARRAY TO STORE DELAYS OF INTEREST FOR THIS CALL  02186000
*      TIMEOUT TIME DURING WHICH A RESPONSE IS EXPECTED BACK            02187000
*                                                                       02188000
* PROCEDURES:                                                           02189000
*  MYSIS   RETURNS A REF. TO THE SIS INSTANCE IN CHARGE OF THE CALL     02190000
* INPUT TO:                                                             02191000
*      I/C SIS S/W                                                      02192000
* OUTPUT FROM:                                                          02193000
*      I/C SIS S/W, CALLRECORD                                          02194000
* ACTIVATES                                                             02195000
*      CALLRECORD                                                       02196000
* ACTIVATED BY:                                                         02197000
*      I/C SIS S/W, CALL RECORD                                         02198000
***********************COMMENT ENDS ********************;               02199000
                                                                        02200000
PROCESS CLASS  INCSISHW(CCTNUM); INTEGER CCTNUM;                        02201000
COMMENT=====================================;                           02202000
BEGIN                                                                   02203000 B130
  INTEGER SOC,I,OUTGCCTNUM;                                               02204000
  BOOLEAN ANSWER, SUBSCLRDOWN, CCTFREE, SEIZURE;                          02205000
  REF(CALLRECORD) MYCALLREC;                                              02206000
  REAL ARRAY RESPONSETIME(1: 8);                                          02207000
  REAL TIMEOUT;                                                           02208000
                                                                          02209000
  REF(SISSW)PROCEDURE MYSIS;                                              02210000
  MYSIS:- P(26 + SISREP(CCTNUM)) QUA SISSW ;                              02211000
                                                                          02212000
  MYCALLREC :- CALL(CCTNUM);                                              02213000
  OUTTV("$$$ THIS I/C SIS H/W HAS S.O.C. = ",SOC);                        02214000
  OUTTV("$$$ THIS I/C CCT. NO. IS = ",CCTNUM);                            02215000
  IF SOC = 1 THEN GO TO A                                                 02216000
  ELSE                                                                    02217000
  IF SOC = 2 THEN GO TO B                                                 02218000
  ELSE                                                                    02219000
  IF SOC = 3 THEN GO TO C                                                 02220000
  ELSE                                                                    02221000
  BEGIN                                                                   02222000 B131
    OUTLINE("??? AN UNALLOWED SOC GENERATED ???");                          02223000
    OUTTV("*** THE SOC VALUE IS  ", SOC);                                   02224000
  END *** OF SOC IN ERROR***;                                             02225000 E131
  COMMENT**GO TO APPRORIATE STAGE OF CALL***;                             02226000
  A: IF NOT SEIZURE THEN                                                  02227000
  BEGIN                                                                   02228000 B132
    OUTLINE("??? I/C SIS HW ACTIVATED WITHOUT 'SEIZURE' MESSAGE ???");      02229000
    OUTLINE("??? THIS IS A FATAL ERROR--RUN ABBORTED ???");                 02230000
    OUTLINE("===================================");                        02231000
  ERROR(21);
  GO TO F1;
END*** OF FALSE SEIZURE MESSAGE ****;
MYSIS.MESSAGE(1,1):=1;
MYSIS.MESSAGE(1,2):=CCTNUM;
COMMENT**SEND SEIZURE MESSAGE AND I/C CCT NO**;
HOLD(31000.00);
COMMENT** PAUSE BEFORE SENDING FIRST DIGIT**;
MYSIS.MESSAGE(2,1):=1;
MYSIS.MESSAGE(2,2):=CCTNUM;
COMMENT** 1ST. DIGIT AND I/C CCT NO. TO I/C SIS S/W REP**;
HOLD(240000.00);
COMMENT** INTER-DIGITAL PAUSE**;
MYSIS.MESSAGE(3,1):=1;
MYSIS.MESSAGE(3,2):=CCTNUM;
COMMENT** 2ND DIGIT AND I/C CCT.NO. TO I/C SIS S/W REP **;
HOLD(180000.00);
COMMENT ** INTER-DIGITAL PAUSE***;
MYSIS.MESSAGE(4,1):=1;
MYSIS.MESSAGE(4,2):=CCTNUM;
IF MYCALLREC.MYOUTSISHW =/= NONE THEN
MYSIS.MESSAGE(4,3) := MYCALLREC.MYOUTSISHW.CCTNUM;
COMMENT**3RD DIGIT AND I/C CCT.NO***;
RESPONSETIME(1) := TIME;
COMMENT**START RESPONSE 1 MEASUREMENT***;
HOLD(180000.00);
COMMENT** INTER-DIGITAL PAUSE***;
MYSIS.MESSAGE(5,1):=1;
MYSIS.MESSAGE(5,2):=CCTNUM;
OUTLINE("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$");
OUTTV("$$$ THIS I/C SIS H/W NO. IS ",CCTNUM);
MYSIS.MESSAGE(5,3):=MYCALLREC.MYOUTSISHW.CCTNUM;
COMMENT** 4TH. DIGIT AND I/C CCT. NO.***;
RESPONSETIME(3) := TIME;
COMMENT**START RESPONSE3 MEASUREMENT***;
HOLD(190000.00);
COMMENT** INTER-DIGITAL PAUSE***;
MYSIS.MESSAGE(6,1):=1;
MYSIS.MESSAGE(6,2):=CCTNUM;
MYSIS.MESSAGE(6,3):=MYCALLREC.MYOUTSISHW.CCTNUM;
COMMENT**5TH.DIGIT TO AND I/C CCT NO TO I/C SIS S/W REP***;
HOLD(170000.00);
COMMENT** INTER-DIGITAL PAUSE***;
MYSIS.MESSAGE(7,1):=1;
MYSIS.MESSAGE(7,2):=CCTNUM;
MYSIS.MESSAGE(7,3):=MYCALLREC.MYOUTSISHW.CCTNUM;
COMMENT***6TH DIGIT AND I/C CCT.NO.***;
HOLD(230000.00);
COMMENT** INTER-DIGITAL PAUSE***;
MYSIS.MESSAGE(8,1):=1;
MYSIS.MESSAGE(8,2):=CCTNUM;
MYSIS.MESSAGE(8,3):=MYCALLREC.MYOUTSISHW.CCTNUM;
COMMENT**7TH. DIGIT AND I/C.NO.**;
HOLD(230000.00);
COMMENT**INTER-DIGITAL PAUSE***;
MYSIS.MESSAGE(9,1):=1;
MYSIS.MESSAGE(9,2):=CCTNUM;
MYSIS.MESSAGE(9,3):=MYCALLREC.MYOUTSISHW.CCTNUM;
COMMENT**8TH. DIGIT AND I/C CCT. NO. ***;
PASSIVATE;
GO TO LASTACTION;
BACK: PASSIVATE;
B:
IF NOT ANSWER THEN
BEGIN
  ERROR(22);
  GO TO BACK;
  COMMENT**WAITING FOR PROPER ANSWER MESSAGE***;
END**FALSE ANSWER**;
TIMEOUT := UNIFORM(1.0, 60000.0, SD10);
HOLD(TIMEOUT);
COMMENT**TIME TO CANCEL I/C SIS S/W REP T/O***;
MYSIS.MESSAGE(10,1):=1;
MYSIS.MESSAGE(10,2):=CCTNUM;
MYSIS.MESSAGE(10,3):=MYCALLREC.MYOUTSISHW.CCTNUM;
COMMENT** CANCEL T/O**;
ACTIVATE CALL(CCTNUM) AFTER CURRENT;
GO TO LASTACTION;
PASIVE:  PASSIVATE;
C:
IF NOT SUBSCLRDOWN  THEN
BEGIN
  ERROR(23);
  GO TO PASIVE;
  COMMENT**WAITING FOR TRUE MESSAGE**;
END;
MYSIS.MESSAGE(11,1):=1;
MYSIS.MESSAGE(11,2):=CCTNUM;
MYSIS.MESSAGE(11,3) := MYCALLREC.MYOUTSISHW.CCTNUM;
COMMENT *** IDLE MESSAGE TO I/C SIS S/W REP ***;
RESPONSETIME(7) := RESPONSETIME(8) := TIME;
Q: PASSIVATE;
IF NOT CCTFREE THEN
BEGIN
  ERROR(24);
  GO TO Q;
END ** FALSE "CCTFREE" MESSAGE ***;
```

```
  ACTIVATE CALL(CCTNUM) AFTER CURRENT;
  LASTACTION:
  SISHWINC(CCTNUM):-NONE;
  MYCAILREC :- NONE;
END ** OF I/C SIS H/W PROCESS **;



COMMENT************** SIGNAL INTERWORKING SUBSYSTEM (SIS) ******
*
* THE SIS MODEL LISTED BELOW HANDLES BOTH THE INCOMMING AND
* OUTGOING CIFCUITS. HOWEVER,FOR CERTAIN APPLICATIONS THERE
* IS A NEED TC SEPARATF THESE TWO FUNCTIONS.THE FOLLOWING
* TWO COMNENTS DESCPIBE THE I/C AND O/G PARTS OF SIS IN
* GREATER DETAIL.THE SIS LISTING IS FOR THE COMPLETE SIS.
**********************************************************;

COMMENT************************ I/C SIS S/W ******************
*
* DESCRIPTION:
*     INTERFACES I/C SIS H/W TO I/C CPS S/W AND PERFORMS
*     PRELIMINARY  PROCESSING
* FUNCTION:
*     IAM<SZ> TO I/C CPS S/W
*     SAMS    TO I/C CPS S/W
*     RELEASE TO I/C CPS S/W
*     ANSWER TC I/C SIS H/W
*     CCTFREE TO I/C SIS H/W
* VARIABLES:
*     X REF TC I/C SIS H/W
*     Y REF TC O/G SIS H/W
*
*     INCCT  STORES I/C CCT. NO.
*     MESSAGE AN ARRAY TO STOR I/C MYSIS.MESSAGES FROM I/C SIS H/W
*             AND I/C CALL IDPROCESS
*     JUMP A SWITCH TO JUMP TO THE APPROPRIATE HARDWARE MESSAGE
*           SEBVICING SEGMENT
* INPUT TO:
*     I/C SIS H/W,I/C CPS S/W
* OUTPUT FROM:
*     I/C SIS H/W,I/C CPS S/W,OG CPS S/W
* ACTIVATE:
*     I/C SIS H/W
* ACTIVATED BY:
*     NONE
************************* COMMENT ENDS *************************;
COMMENT*************** O/G SIS S/W *************************
*
* DESCRIPTION:
*     O/G SIGNAL INTER-WOR?ING SUBSYSTEM SOFTWARE
* FUNCTION:
*     INTERFACES BETWEEN CPS AND O/G CCTS HARDWARE
* VARIABLES:
*     MESSAGE INTEGER ARRAY TO RECORD INCOMING H/W MESSAGES
*     CASE A SWITCH TO BRANCH TO APPROPRIATE ACTION
*
*INPUT TO:
*     O/G CPS,O/G SIS H/W
*OUTPUT FROM:
*     O/G CPS,O/G SIS H/W,I/C CPS
*
*ACTIVATES:
*     O/G SIS H/W
*ACTIVATED BY:
*     NONE
************************** COMMENT ENDS ********************;

AP            CLASS SISSW;
COMMENT       ===============;
BEGIN
  SWITCH CASE:=T1,T2,T3,T4,T5,T6,T7,T8,T9,T10;
  SWITCH JUMP:=M1,M2,M3,M4,M5,M6,M7,M8,M9,M10,M11,
  M12,M13,M14,M15,M16,M17,M18,M19,M20;
  INTEGER TASK,I,INCCT;
  SHORT INTEGER ARRAY MESSAGE(1:24,1:3);
  REF(INCSISHW) X;
  REF(OUTGSISHW)Y;
  A: FETCH(15);
  IF CC>0 THEN
  BEGIN
    TASK:=RELEVANTCPU.G(1);
    GO TO CASE(TASK);
  END
  ELSE
  BEGIN
    D: I:=1;
    WHILE AND2(MESSAGE(I,1)=0 , I<=23) DO I:=I+1;
    IF MESSAGE(I,1) = 1 THEN
    BEGIN
      OUTLINE("**A H/# MESSAGE FROM AN SIS H/W IS FOUND");
      OUTTV("**THE MESSAGE INDEX IS = ",I);
      OUTTV("**RELATED TO I/C CCT. NO. ",MESSAGE(I,2));
      OUTTV("**WHOSE O/G CCT. NO. IS ",MESSAGE(I,3));
      IF MESSAGE(I,2) = 0 THEN
      OUTLINE("??? FAULTY H/W MESSAGE RECEIVED ????");
      GO TC JUMP(I);
    END
    ELSE
    BEGIN                                          B140
      BLOCK(15);
      GO TO A;
  END ** NO TASK AND NO MESSAGE**;                 E140
END;                                               E13B
T1: HOLD(1236.0);
COMMENT ** PROCESSING TIME ***;
GO TO A;
T2: HOLD(1368.0);
COMMENT** PROCESSING TIME ***;
I:=RELEVANTCPU.G(3);
X:-SISHWINC(I);
X.ANSWER:=TRUE;
ACTIVATE X AFTER CURRENT;
GO TO A; COMMENT**"ANSWER" TO I/C SIS H/W **;
T3: HOLD(1463.0);
COMMENT** PROCESSING TIME **;
I:=RELEVANTCPU.G(3);
X:-SISHWINC(I);
X.CCTFREE:=TRUE;
ACTIVATE X AFTER CURRENT;
GO TO A;
COMMENT*** "CCTFREE" TO I/C SIS H/W ***;
M1:MESSAGE(1,1):=0;
HOLD(1531.0);
COMMENT*** PROCESSING TIME **;
INCCT := MESSAGE(1,2);
G(0):=2+CPSREP(INCCT);
G(1):=1;
G(3):=INCCT;
HAND;
COMMENT **"IAM(SZ)" TO I/C CPS REP **;
GO TO D;
M2: MESSAGE(2,1):=0;
HOLD(1517.0);
COMMENT*** PROCESSING TIME**;
INCCT:=MESSAGE(2,2);
G(0):=2+CPSREP(INCCT);
G(1):=2;
G(3):=INCCT;
HAND;
COMMENT ** SAM1 T/O I/C CPS REP **;
GO TO D;
M3: MESSAGE(3,1):=0;
INCCT:=MESSAGE(3,2);
HOLD(1517.0);
COMMENT ** PROCESSING TIME **;
G(0):=2+CPSREP(INCCT);
G(1):=3;
G(3):=INCCT;
HAND;
COMMENT ** SAM2  TO I/C CPS REP **;
GO TO D;
M4: MESSAGE(4,1):=0;
INCCT:=MESSAGE(4,2);
HOLD(1517.0);
COMMENT ** PROCESSING TIME **;
X:- SISHWINC(INCCT);
X.RESPONSETIME(2) := TIME;
G(0):=2+CPSREP(INCCT);
G(1):=4;
G(3):=INCCT;
HAND;
COMMENT ** SAM3 TO I/C CPS REP **;
GO TO D;
M5: MESSAGE(5,1):=0;
INCCT:=MESSAGE(5,2);
HOLD(1517.0);
COMMENT ** PROCESSING TIME **;
X:-SISHWINC(INCCT);
X.RESPONSETIME(4) := TIME;
G(0):=2+CPSREP(INCCT);
G(1):=7;
G(3):=INCCT;
G(4):=X.OUTGCCTNUM;
HAND;
COMMENT ** SAM4 TO I/C CPS REP **;
GO TO D;
M6: MESSAGE(6,1):=0;
INCCT:=MESSAGE(6,2);
HOLD(1517.0);
COMMENT ** PROCESSING TIME **;
X:-SISHWINC(INCCT);
G(0):=2+CPSREP(INCCT);
G(1):=8;
G(3):=INCCT;
G(4):=X.OUTGCCTNUM;
HAND;
COMMENT ** SAM5 TO I/C CPS REP **;
GO TO D;
M7: MESSAGE(7,1):=0;
INCCT:=MESSAGE(7,2);
HOLD(1517.0);
COMMENT ** PROCESSING TIME **;
X:-SISHWINC(INCCT);
G(0):=2+CPSREP(INCCT);
```

```
G(1):=9;
G(3):=INCCT;
G(4):=X.OUTGCCTNUM;
HAND;
COMMENT ** SAM6 T/O I/C CPS REP ***;
GO TO D;
M8: MESSAGE(8,1):=0;
INCCT:=MESSAGE(8,2);
HOLD(1517.0);
COMMENT ** PROCESSING TIME **;
X:-SISHWINC(INCCT);
G(0):=2+CPSREP(INCCT);
G(1):=10;
G(3):=INCCT;
G(4):=X.OUTGCCTNUM;
HAND;
COMMENT ** SAM7 TO I/C CPS REP **;
GO TO D;
M9: MESSAGE(9,1):=0;
INCCT:=MESSAGE(9,2);
HOLD(1517.0);
COMMENT ** PROCESSING TIME ***;
X:-SISHWINC(INCCT);
G(0):=2+CPSREP(INCCT);
G(1):=11;
G(3):=INCCT;
G(4):=X.OUTGCCTNUM;
HAND;
COMMENT*** SAM8 TO I/C CPS REP **;
GO TO D;
M10: MESSAGE(10,1):=0;
HOLD(1265.0);
COMMENT ** PROCESSING TIME **;
GO TO D;
M11: MESSAGE(11,1):=0;
INCCT:=MESSAGE(11,2);
HOLD(1860.0);
COMMENT ** PROCESSING TIME ***;
COMMENT*** "METER OVER JUNCTION" SIGNAL IS NOT SIMULATED ***;
G(0):=2+CPSREP(INCCT);
G(1):=14;
G(3):=MESSAGE(11,2);
G(4):=SISHWINC(INCCT).OUTGCCTNUM;
HAND;
COMMENT **"RELEASE" MESSAGE TO I/C CPS **;
GO TO D;
T4: HOLD(3410.0);
COMMENT** PROCESSING TIME ***;
I:=RELEVANTCPU.G(4);
Y:-SISHWOUT(I);
RESPONSE(1).UPDATE(TIME - SISHWINC(RELEVANTCPU.G(3)).RESPONSETIME(1));
COMMENT ** END RESPONSE1 **;
Y.MESSAGE(1,1):=1;
Y.MESSAGE(1,2):=RELEVANTCPU.G(3);
COMMENT** SEIZE Y.MESSAGE TO O/G SIS H/W **;
ACTIVATE Y AFTER CURRENT;
GO TO A;
T5: HOLD(2297.0);
COMMENT ** PROCESSING TIME **;
I:=RELEVANTCPU.G(4);
Y:-SISHWOUT(I);
Y.MESSAGE(3,1):=1;
COMMENT ** DIGIT2 TO O/G SIS H/W **;
Y.MESSAGE(3,2):=RELEVANTCPU.G(3);
RESPONSE(3).UPDATE(TIME - SISHWINC(RELEVANTCPU.G(3)).RESPONSETIME(3));
ACTIVATE Y AFTER CURRENT;
GO TO A;
T6: HOLD(2297.0);
COMMENT*** PROCESSING TIME ***;
I:=RELEVANTCPU.G(4);
Y:-SISHWOUT(I);
Y.MESSAGE(4,1):=1;
COMMENT ** DIGIT3 TO O/G SIS H/W **;
Y.MESSAGE(4,2):=RELEVANTCPU.G(3);
ACTIVATE Y AFTER CURRENT;
GO TO A;
T7: HOLD(2297.0);
COMMENT **PROCESSING TIME**;
I:=RELEVANTCPU.G(4);
Y:-SISHWOUT(I);
Y.MESSAGE(5,1):=1;
COMMENT*** DIGIT 4 TO O/G SIS H/W***;
Y.MESSAGE(5,2):=RELEVANTCPU.G(3);
ACTIVATE Y AFTER CURRENT;
GO TO A;
T8: HOLD(2297.0);
COMMENT*** PROCESSING TIME **;
I:=RELEVANTCPU.G(4);
Y:-SISHWOUT(I);
Y.MESSAGE(6,1):=1;
Y.MESSAGE(6,2):=RELEVANTCPU.G(3);
ACTIVATE Y AFTER CURRENT;
GO TO A;
T9: HOLD(2297.0);
I:=RELEVANTCPU.G(4);
Y:-SISHWOUT(I);
Y.MESSAGE(7,1):=1;
```

```
     COMMENT*** DIGIT6 TO O/G SIS H/W***;
     Y.MESSAGE(7,2):=RELEVANTCPU.G(3);
     ACTIVATE Y AFTER CURRENT;
     GO TO A;
     T10: HOLD(2310.0);
     COMMENT** PROCESSING TIME **;
     I:=RELEVANTCPU.G(4);
     Y:-SISHWOUT(I);
     Y.MESSAGE(9,1):=1;
     Y.MESSAGE(9,2):=RELEVANTCPU.G(3);
     RESPONSE(8).UPDATE(TIME - SISHWINC(RELEVANTCPU.G(3)).RESPONSETIME(8));
     ACTIVATE Y AFTER CURRENT;
     COMMENT***IDLE OR CLEAR FORWARD TO O/G SIS H/W***;
     GO TO A;
     M12: MESSAGE(12,1):=0;
     HOLD(1615);
     COMMENT**PROCESSING TIME**;
     Y:-SISHWOUT(MESSAGE(12,3));
     Y.MESSAGE(2,1):=1;
     Y.MESSAGE(2,2):=MESSAGE(12,2);
     COMMENT **DIGIT1 TO O/G SIS H/W***;
     ACTIVATE Y AFTER CURRENT;
     GO TO D;
     M13: MESSAGE(13,1):=0;
     HOLD(1833.0);
     COMMENT** PROCESSING TIME ***;
     COMMENT**RE-SET T/O2 FLAG**;
     GO TO D;
     M14: MESSAGE(14,1):=0;
     HOLD(1833.0);
     COMMENT*** PROCESSING TIME **;
     COMMENT***RE-SET T/O3 FLAG**;
     GO TO D;
     M15: MESSAGE(15,1):=0;
     HOLD(1833.0);
     COMMENT** PROCESSING TIME**;
     COMMENT***RE-SET T/O4 FLAG**;
     GO TO D;
     M16: MESSAGE(16,1):=0;
     HOLD(1833.0);
     COMMENT***PROCESSING TIME**;
     COMMENT***RE-SET T/O5 FLAG***;
     GO TO D;
     M17: MESSAGE(17,1):=0;
     HOLD(1833.0);
     COMMENT***PROCESSING TIME***;
     COMMENT***RE-SET T/O6 FLAG**;
     GO TO D;
     M18: MESSAGE(18,1):=0;
     HOLD(2439.0);
     COMMENT***PROCESSING TIME**;
     COMMENT**RE-SET T/O7 FLAG**;
     SISHWINC(MESSAGE(18,2)).RESPONSETIME(6) := TIME;
     COMMENT*** START RESPONSE6 **;
     G(0):= 2+ CPSREP(MESSAGE(18,3));
     G(1):=18;
     G(3):=MESSAGE(18,2);
     G(4):=MESSAGE(18,3);
     HAND;
     COMMENT*** NO. RECEIVED TO O/G CPS REP.**;
     GO TO D;
     M19: MESSAGE(19,1):=0;
     HOLD(1236.0);
     COMMENT** PROCESSING TIME ***;
     COMMENT***RE-SET ANSWER FLAG ***;
     G(0):= 2+ CPSREP(MESSAGE(19,3));
     G(1):=20;
     G(3):=MESSAGE(19,2);
     G(4):=MESSAGE(19,3);
     HAND;
     COMMENT***ANSWER MESSAGE TO O/G CPS REP**;
     GO TO D;
     M20: MESSAGE(20,1):=0;
     HOLD(1875.0);
     COMMENT*** PROCESSING TIME ***;
     G(0):= 2 + CPSREP(MESSAGE(20,3));
     G(1):=23;
     G(3):=MESSAGE(20,2);
     G(4):=MESSAGE(20,3);
     HAND;
     COMMENT**CIRCUIT FREE TO O/G CPS REP. **;
     GO TO D;
   END *** OF SIS *****;                                                          E136



   COMMENT**************** CALL PROCESSING  SUBSYSTEM (CPS)  ******
   *
   * THE CPS MODEL LISTED BELOW HANDLES BOTH THE INCOMMING AND
   * OUTGOING CIRCUITS. HOWEVER,FOR CERTAIN APPLICATIONS THERE
   * IS A NEED TO SEPARATE THESE TWO FUNCTIONS.THE FOLLOWING
   * TWO COMMENTS DESCRIBE THE I/C AND O/G PARTS OF CPS IN
   * GREATER DETAIL.THE CPS LISTING IS FOR THE COMPLETE CPS.
   ***********************************************************************;

   COMMENT********************I/C CPS S/W ********************
   *
```

```
* DESCRIPTICN:
*     RESPONSIBLE FOR THE I/C CALL PROCESSING
* FUNCTICN:
*     PART-FILE READ TO STORAGE ALLCCATOR
*     IAM AND SAMS TO O/G SIS S/W
*     "NO RECEIVED" TC I/C SIS S/W
*     "SET UP PATH" TO DSS S/W
*     "CCT FREE" TO I/C SIS S/W
* VARIABLES:
*     MYINCSISREP REP OF I/C SIS
*     CASE A SWITCH TO SELECT ACTION ACCORDING TO I/C G<1>
*     TSK I/C G<1> VALUE
*     OUTGREP REP NO. OF O/G CPS
*     INCCT I/C CCT NO.
*     OUTCCT C/G CCT NO.
*     CALLREC REF TO CALLRECORD
*     X REF TC I/C SIS H/W
*    Y REF TO O/G SIS H/W
* INPUT TO:
*     I/C SIS S/W,O/G CPS,DSS S/W,O/G SIS S/W,SA
* OUTPUT FRCM:
*     I/C SIS S/W,O/G CPS,DSS S/W,SA
* ACTIVATES:
*     NONE
* ACTIVATED BY:
*     NONE
****************************** COMMENT ENDS ******************;
COMMENT*************O/G CPS S/W ******************************
*
* DESCRIPTION:
*     RESPONSIBLE FOR O/G CALL PROCESSING
* FUNCTICN:
*     "START SENDING" TO I/C CPS
*     "NO. RECEIVED" TO I/C CPS
*     "ANSWER" TO I/C SIS
*     "RELEASE" TO O/G SIS
*     "CLEAR SWITCH PATH" TO DSS S/W
*     "RELEASE STARTED" TO I/C CPS
*     "CALL RELEASED" TO I/C CPS
*
* VARIABLFS:
*     CASE  ASWITCH TO SELECT ACTION ACCORDING TO I/C TASK
*     OUTGREP  REP. NO. OF CALLED PROCESS
*     OUTCCT  O/G CIRCUIT NO.
*     INCCT  I/C CIRCUIT NO.
*     CALLREC A REF TO CALLRECORD
*
* INPUT TO:
*     I/C CPS, I/C SIS, O/G SIS, DSS S/W
* OUTPUT FROM:
*     I/C CPS, O/G SIS, DSS S/W
* ACTIVATES:
*     NONE
* ACTIVATED BY:
*     NONE
*
**************** COMMENT ENDS ******************************;

AP      CLASS CPSSW;
COMMENT ============;
BEGIN                                                                  B141
  SWITCH CASE:=T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,
  T14,T15,T16,T17,T18,T19,T20,T21,T22,T23;
  INTEGER TSK,OUTGREP,OUTCCT,INCCT,I,U;
  REF(SISSW)MYINCSISREP;
  REF(OUTGSISHW)Y;
  B: TSK:=RELEVANTCPU.G(1);
  GO TO CASE(TSK);
  A: FETCH(15);
  IF CC>0 THEN GO TO B
  ELSE
  BEGIN                                                                B142
    BLOCK(15);
    GO TC B;
  END*NO TASK IN I/P Q**;                                              E142
  T1: HOLD(8025.0);
  COMMENT** PROCESSING TIME  AND RESPONSE TO IAM<SZ>**;
  GO TO A;
  T2: HOLD(4445.0);
  COMMENT ** PROCESSING TIME  AND SAM1 RESPONSE**;
  GO TC A;
  T3: HOLD(8070.0);
  COMMENT ** PROCESSING TIME  AND SAM2 RESPONSE**;
  GO TC A;
  T4: HOLD(5250.0);
  COMMENT ** PROCESSING TIME **;
  G(0):=1;G(1):=PI;
  G(2):=8;
  G(3):=RELEVANTCPU.G(3);
  BAND;
  COMMENT ** PART-FILE READ TO SA **;
  GO TO A;
  T5: HOLD(100.0);
  COMMENT *** PROCESSING TIME **;
  OUTGREP := RANDINT(0, 3, SD16);
  COMMENT ** DETERMINE RANDOMLEY O/G CPS REP. TOTAL NO.OF
  O/G CCTS IS TOTALCCTS, CPSREPS IS TOTAL NO. OF CPS REPLICATIONS*;
```

```
         G(0):=8 + OUTGREP;
         G(1):=17;
         G(3):=RELEVANTCPU.G(3);
         HAND;
         COMMENT **"O/G ROUTE SEIZE" TO O/G CPS REP **;
         HOLD(1825.0);
         COMMENT** PROCESSING TIME **;
         GO TO A;
         T6: HOLD(4770.0);
         COMMENT ** PROCESSING TIME ***;
         OUTCCT:=RELEVANTCPU.G(4);
         INCCT:=RELEVANTCPU.G(3);
         RESPONSE(2).UPDATE(TIME - SISHWINC(INCCT).RESPONSETIME(2));
         G(0):=3+SISREP(OUTCCT);;
         G(1):=4;
         G(3):=INCCT;
         G(4):=OUTCCT;
         HAND;
         COMMENT** END RESPONSETIME<2> AND "IAM" TO O/G SIS **;
         HOLD(2960.0);
         COMMENT ** PROCESSING TIME.**;
         SEEK(14);
         HOLD(10.0);
         COMMENT** PROCESSING TIME **;
         SEEK(15);
         HOLD(20.0);
         COMMENT ** PROCESSING TIME **;
         GO TO A;
         T7: HOLD(5605.0);
         COMMENT***PROCESSING TIME***;
         INCCT:=RELEVANTCPU.G(3);
         OUTCCT:=RELEVANTCPU.G(4);
         RESPONSE(4).UPDATE(TIME - SISHWINC(INCCT).RESPONSETIME(4));
         COMMENT** END TIME OF RESPONSE 4**;
         G(0):=3+SISREP(OUTCCT);
         G(1):=5;
         G(3):=INCCT;
         G(4):=OUTCCT;
         HAND;
         COMMENT***SAM1 TO O/G SIS S/W***;
         HOLD(1005.0);
         COMMENT**PROCESSING TIME ***;
         GO TO A;
         T8: HOLD(5605.0);
         COMMENT**PROCESSING TIME***;
         INCCT:=RELEVANTCPU.G(3);
         OUTCCT:=RELEVANTCPU.G(4);
         G(0):= 3 + SISREP(OUTCCT);
         G(1):= 6;
         G(3):=INCCT;
         G(4):=OUTCCT;
         HAND;
         COMMENT***SAM2 TO O/G SIS S/W REP **;
         HOLD(100.0);
         COMMENT*** PROCESSING TIME ***;
         GO TO A;
         T9: HOLD(5605.0);
         COMMENT** PROCESSING TIME ***;
         INCCT:=RELEVANTCPU.G(3);
         OUTCCT:=RELEVANTCPU.G(4);
         G(0):=3+SISREP(OUTCCT);
         G(1):=7;
         G(3):=INCCT;
         G(4):=OUTCCT;
         HAND;
         COMMENT** SAM3 TO O/G SIS S/W REP ***;
         HOLD(1005.0);
         COMMENT** PROCESSING TIME ***;
         GO TO A;
         T10: HOLD(5605.0);
         COMMENT ** PROCESSING TIME **;
         INCCT:=RELEVANTCPU.G(3);
         OUTCCT:=RELEVANTCPU.G(4);
         G(0):=3+SISREP(OUTCCT);
         G(1):=8;
         G(3):=INCCT;
         G(4):=OUTCCT;
         HAND;
         COMMENT**SAM4 TO O/G SIS S/W REP ***;
         HOLD(1005.0);
         COMMENT** PROCESSING TIME***;
         GO TO A;
         T11: HOLD(5530.0);
         COMMENT**PROCESSING TIME ***;
         INCCT:=RELEVANTCPU.G(3);
         OUTCCT:=RELEVANTCPU.G(4);
         G(0):=3+SISREP(OUTCCT);
         G(1):=9;
         G(3):=INCCT;
         G(4):=OUTCCT;
         HAND;
         COMMENT ** FAM TO O/G SIS S/W REP ***;
         HOLD(915.0);
         COMMENT** PROCESSING TIME **;
         GO TO A;
         T12: HOLD(3615.0);
```

```
COMMENT ** PROCESSING TIME **;
 INCCT:=RELEVANTCPU.G(3);
 OUTCCT:=RELEVANTCPU.G(4);
G(0):=3+SISREP(INCCT);
G(1):=1;
G(3) := INCCT;
G(4) := OUTCCT;
HAND;
COMMENT** NO. RECEIVED TO I/C SIS REP **;
HOLD(1980.0);
COMMENT ** PROCESSING TIME ***;
G(0):=2;
G(1):=1;
G(3):=INCCT;
G(4):=OUTCCT;
HAND;
COMMENT *** SET:UP SWITCH PATH MESSAGE TO DSS S/W**;
HOLD(1690.0);
COMMENT ** PROCESSING TIME **;
GO TO A;
T13: HOLD(3765.0);
COMMENT ** PROCESSING TIME **;
 INCCT:=RELEVANTCPU.G(3);
OUTCCT:=RELEVANTCPU.G(4);
G(0):=8+CPSREP(OUTCCT);
G(1):=19;
G(3):=INCCT;
G(4):=OUTCCT;
HAND;
COMMENT ** SET UP COMPLETE  TO O/G CPS REP **;
HOLD(520.0);
COMMENT ** PROCESSING TIME **;
SEEK(14);
HOLD(10.0);
COMMENT ** PROCESSING TIME **;
SEEK(15);
HOLD(20.0);
COMMENT** PROCESSING TIME***;
ACTIVATE SISHWINC(INCCT) AFTER CURRENT;
GO TO A;
T14: HOLD(805.0);
COMMENT ** PROCESSING TIME **;
 INCCT:=RELEVANTCPU.G(3);
OUTCCT:=CALL(INCCT).OUTGCCTNUM;
G(0):=8+CPSREP(OUTCCT);
G(1):=21;
G(3):=INCCT;
G(4):=OUTCCT;
HAND;
COMMENT ** RELEASE MESSAGE TO O/G CPS REP**;
HOLD(200.0);
COMMENT **PROCESSING TIME **;
GO TO A;
T15: HOLD(1365.0);
COMMENT ** PROCESSING TIME **;
GO TO A;
T16: HOLD(100.0);
COMMENT ** PROCESSING TIME **;
 INCCT:=RELEVANTCPU.G(3);
OUTCCT:=RELEVANTCPU.G(4);
G(0):=3+SISREP(INCCT);
G(1):=3;
G(3):=INCCT;
G(4):=OUTCCT;
HAND;
COMMENT ** CIRCUIT FREE TO I/C SIS REP**;
HOLD(695.0);
COMMENT ** PROCESSING TIME **;
SEEK(14);
HOLD(10.0);
SEEK(15);
HOLD(20.0);
GO TO A;
T17: HOLD(5950.0);
COMMENT ** PROCESSING TIME**;
OUTCCT := RANDINT(MININCCT, MAXOUTCCT, SD18);
WHILE SISHWOUT(OUTCCT) =/= NONE DO
OUTCCT := RANDINT(MININCCT, MAXOUTCCT, SD19);
COMMENT ***OUTCCT STORES THE ABSOLUTE O/G CCT. NO. ***;
SISHWOUT(OUTCCT) :- NEW OUTGSISHW(OUTCCT);
COMMENT **GENERATE AN INSTANCE OF O/G SIS H/W ***;
CALL(RELEVANTCPU.G(3)).MYOUTSISHW:-SISHWOUT(OUTCCT);
CALL(RELEVANTCPU.G(3)).OUTGCCTNUM:=OUTCCT;
SISHWINC(RELEVANTCPU.G(3)).OUTGCCTNUM:=OUTCCT;
SISHWOUT(OUTCCT).MYCALLREC :- CALL(RELEVANTCPU.G(3));
COMMENT ** LINK IT TO ITS CALLRECORD **;
G(0):=8+CPSREP(RELEVANTCPU.G(3));
G(1):=6;
G(3):=RELEVANTCPU.G(3);
G(4):=OUTCCT;
HAND;
COMMENT ** START SENDING MESSAGE TO I/C CPS REP **;
HOLD(220.0);
COMMENT** PROCESSING TIME ***;
GO TO A;
T18: HOLD(2235.0);
```

```
    G(0):=8+CPSREP(RELEVANTCPU.G(3));
    G(1):=12;
    G(3):=RELEVANTCPU.G(3);
    G(4):=RELEVANTCPU.G(4);
    HAND;
    COMMENT ** NO. RECEIVED MESSAGE TO I/C CPS REP. **;
    HOLD(125.0);
    COMMENT **PROCESSING TIME **;
    GO TO A;
    T19: HOLD(615.0);
    COMMENT ** MESSAGE TO MSS NOT SIMULATED **;
    GO TO A;
    T20: HOLD(2870.0);
    G(0):=3+SISREP(RELEVANTCPU.G(3));
    G(1):=2;
    G(3):=RELEVANTCPU.G(3);
    G(4):=RELEVANTCPU.G(4);
    HAND;
    COMMENT ** ANSWER TO I/C SIS REP ***;
    HOLD(500.0);
    COMMENT ** PROCESSING TIME **;
    GO TO A;
    T21: HOLD(2880.0); OUTCCT:=RELEVANTCPU.G(4);
    COMMENT ** PROCESSING TIME ***;
    COMMENT **STORE PI OF O/G SIS REP.***;
    G(0):=3+SISREP(OUTCCT);
    G(1):=10; INCCT:=RELEVANTCPU.G(3);
    G(3) := INCCT;
    G(4):=OUTCCT;
    HAND;
    COMMENT **"RELEASE" TO O/G SIS REP **;
    HOLD(1325.0);
    COMMENT ** PROCESSING TIME ***;
    G(0):= 2;
    G(1):= 2; G(3):=INCCT; G(4):=OUTCCT;
    HAND;
    COMMENT**"CLEAR SWITCH PATH" TO DSS S/W **;
    HOLD(150.0);
    COMMENT ** PROCESSING TIME **;
    G(0):=8+CPSREP(INCCT);
    G(1):=15;
    G(3):=INCCT;
    G(4):=OUTCCT;
    HAND;
    COMMENT **"RELEASE STARTED" TO I/C CPS REP **;
    HOLD(1410.0);
    COMMENT **PROCESSING TIME **;
    GO TO A;
    T22: HOLD(3365.0);
    COMMENT ** PROCESSING TIME **;
    G(0):=8+CPSREP(RELEVANTCPU.G(3));
    G(1):=16;
    G(3):=RELEVANTCPU.G(3);
    G(4):=RELEVANTCPU.G(4);
    HAND;
    COMMENT **"CALL RELEASED" TO I/C CPS REP **;
    HOLD(2050.0);
    COMMENT **PROCESSING TIME **;
    SEEK(14);
    HOLD(10.0);
    SEEK(15);
    HOLD(20.0);
    GO TO A;
    T23: HOLD(1640.0);
    COMMENT ** PROCESSING TIME **;
    GO TO A;
END ** CPS PROCESS ***;



COMMENT*****************DSS S/W*******************************
*                       =======
* DESCRIPTION:
*    DIGITAL SWITCH SUB-SYSTEM SOFTWARE PROCESS
* FUNCTION:
*     SETS UP AND CLEAR PATHS OF INDIVIDUAL TELEPHONE CALLS
* VARIABLES:
*     X REFERS TO I/C SIS H/W
*     RESPONSE1 "SET UP PATH" RESPONSE FROM DSS H/W
*     RESPONSE2 "CLEAR PATH"  RESPONSE FROM DSS H/W
*     INCCTS   I/C CCT FOR SWITCH SET-UP
*     OUTCCTS O/G "  "    "     "
*     INCCTC   I/C "  "    "     CLEAR DOWN
*     OUTCCTC "  "   "    "     "
*     CASE  A SWITCH TO SELECT  APPROPRIATE ACTION
* INPUT FROM:
*     I/C CPS, O/G CPS ,DSS H/W
* OUTPUT TO:
*     I/C CPS, O/G CPS ,DSS H/W
* ACTIVATES:
*     DSS H/W
* ACTIVATED BY:
*     NONE
*
*****************************COMMENT ENDS ******************;

AP         CLASS  DSSSW;
```

```
COMMENT ===============;
BEGIN
  SWITCH CASE:= T1,T2;
  BOOLEAN RESPONSE1,RESPONSE2;
  INTEGER INCCTS, OUTCCTS, INCCTC, OUTCCTC;
  A: FETCH(15);
  IF CC>0 THEN GO TO CASE( RELEVANTCPU.G(1))
  ELSE
  IF RESPONSE1 THEN
  BEGIN
    RESPONSE1:=FALSE;
    HOLD(1920.0);
    COMMENT PROCESSING TIME **;
    RESPONSE(6).UPDATE( TIME - SISHWINC(INCCTS).RESPONSETIME(6));
    COMMENT ** END RESPONSE6 STORED IN CALLREC **;
    G(0):= 2+CPSREP( INCCTS);
    G(1) := 13; G(3):= INCCTS; G(4) := OUTCCTS;
    HAND;
    COMMENT ** SWITCH RESPONSE TO I/C CPS REP **;
    GO TO A;
  END*** RESPONSE1**
  ELSE
  IF RESPONSE2 THEN
  BEGIN
    RESPONSE2:=FALSE;
    HOLD(1905.0);
    COMMENT ** PROCESSING TIME ***;
    COMMENT **CALCULATE AND STORE PI OF O/G CPS REP.**;
    RESPONSE(7).UPDATE( TIME - SISHWINC( INCCTC ).RESPONSETIME(7));
    G(0):=2+CPSREP( OUTCCTC );G( 1):=22;
    G(3) := INCCTC;
    G(4) := OUTCCTC;
    HAND;
    COMMENT ** SWITCH RESPONSE TO O/G CPS REP **;
    GO TO A;
  END ***RESPONSE2 **
  ELSE
  BEGIN
    BLOCK( 10 );
    GO TO A;
  END ** OF CC=0 PART OF IF STATEMENT**;
  T1: HOLD(1920.0);
  COMMENT ** PROCESSING TIME**;
  DIGITALSWITCH.SETUPPATH:=TRUE;
  DIGITALSWITCH.INCCTS:=RELEVANTCPU.G(3);
  DIGITALSWITCH.OUTCCTS:=RELEVANTCPU.G(4);
  COMMENT ** SO THAT DSS H/W ENDS RESPONSES LATER **;
  ACTIVATE DIGITALSWITCH AFTER CURRENT;
  GO TO A;
  T2: HOLD(1905.0);
  COMMENT **PROCESSING TIME **;
  DIGITALSWITCH.CLEARPATH:=TRUE;
  DIGITALSWITCH.INCCTC:=RELEVANTCPU.G(3);
  DIGITALSWITCH.OUTCCTC:=RELEVANTCPU.G(4);
  ACTIVATE DIGITALSWITCH AFTER CURRENT;
  GO TO A;
END ** DSS S/W **;



COMMENT********************* DSS H/W ************************
*
* DESCRIPTION:
*     DIGITAL SWITCH HARDWARE
* FUNCTION:
*     CONNECTS I/C AND O/G CCTS
* VARIABLES:
*     SETUPPATH  TO INDICATE ARRIVAL OF THAT MESSAGE
*     CLEARPATH  TO INDICATE ARRIVAL OF THAT MESSAGE
*     INCCTS   I/C CCT FOR SWITCH SET-UP
*     OUTCCTS O/G "   "    "      "
*     INCCTC   I/C "   "    "    CLEAR DOWN
*     OUTCCTC "   "   "    "       "
*     SETUPTIME TIME TO SET-UP SWITCH
*     CLEARTIME TIME TO CLEAR DOWN THE CONNECTION
* INPUT TO:
*     DSS S/W
* OUTPUT FROM:
*     DSS S/W
* ACTIVATES:
*     NONE
* ACTIVATED BY:
*     DSS S/W
********************** COMMENT ENDS ************************;

PROCESS  CLASS DSSHW;
COMMENT ============;
BEGIN
  BOOLEAN SETUPPATH,CLEARPATH;
  INTEGER INCCTS, OUTCCTS, INCCTC, OUTCCTC;
  REAL SETUPTIME, CLEARTIME;
  B: IF SETUPPATH THEN
  BEGIN
    SETUPPATH:=FALSE;
    SETUPTIME := UNIFORM( 1.0, 10000.0, SD11);
    HOLD(SETUPTIME);
    COMMENT ** TIME TAKEN TO SET UP THE PATH ***;
```

```
          DSSHANDLER.INCCTS := INCCTS;
          DSSHANDLER.OUTCCTS := OUTCCTS;
          DSSHANDLER.RESPONSE1:=TRUE;
          COMMENT***SWITCH RESPONSE TO DSS S/W **;
          RESPONSE(5).UPDATE(TIME - SISHWINC(INCCTS).RESPONSETIME(5));
          PASSIVATE;
          GO TC B;
        END ** CF SET UP PATH**
        ELSE
        IF CLEARPATH THEN
        BEGIN
          CLEARPATH:=FALSE;
          CLEARTIME := UNIFORM(1.0, 10000.0, SD12);
          HOLD(CLEARTIME);
          COMMENT ** TIME TAKEN TO CLEAR SWITCH PATH ***;
          DSSHANDLER.INCCTC := INCCTC;
          DSSHANDLER.OUTCCTC := OUTCCTC;
          DSSHANDLER.RESPCNSE2:=TRUE;
          COMMENT ** SWITCH RESPONSE TO DSS S/W **;
          PASSIVATE;
          GO TC B;
        END **CLEAR PATH**
        ELSE
        BEGIN
          ERROR(20);
          PASSIVATE;
          GO TC B;
        END ** ERRCNIOUS ACTIVATION OF DSSHW *;
      END *** DSS.HW**;



      COMMENT***************** O/G SIS H/W ************************
      *
      * DESCRIPTICN
      *     O/G SIGNAL INTER-WORKING SUBSYSTEM
      * FUNCTICN:
      *     RESETS T/OS ON O/G SIS S/W. TELLS O/G SIS REP. WHEN
      *     SUBS. ANSWERS AND CLEARS DOWN
      * VARIABLES:
      *     MESSAGE AN ARRAY TO STORE I/C MESSAGES AND I/C CCT NO.
      *     MYCALLREC REF T/O CALLRECORD OF THIS CALL
      *     CASE A SWITCH TO JUMP TO APPROPRIATE CODE
      *     MYSIS REF TO O/G SIS S/W REPLICATION
      *     CCTNUM C/G CCT. NO.
      *     TIMEOUT TIME WITHIN WHICH A RESPCNSE MUST BE SENT
      *     SENDBACK TIME TO SEND BACK BUSY SIGNAL
      *     SENDFREE "   "   "   "  FREE   "
      * INPUT TO:
      *     O/G SIS S/W REP
      * OUTPUT FRCM:
      *     O/G SIS S/W REP,MYCALLREC
      * ACTIVATES:
      *     CALLRECCRD
      * ACTIVATED BY:
      *     MYCALLREC, O/G SIS REP
      *************************** COMMENT ENDS *******************;

      PROCESS CLASS OUTGSISHW(CCTNUM);
      COMMENT========================;
      INTEGER CCTNUM;
      BEGIN
        INTEGER ARRAY MESSAGE (1:10,1:2);
        REF(CALLRECCRD)MYCALLREC;
        SWITCH CASE:=M1,M2,M3,M4,M5,M6,M7,M8,M9;
        INTEGER I;
        REAL TIMEOUT, SENDBACK, SENDFREE;
        REF(SISSW)MYSIS;
        MYSIS:-P(26+SISREP(CCTNUM)) QUA SISSW;
        A: I:=1;
        WHILE AND2(MESSAGE(I,1)=0 , I<9) DO I:=I+1;
        IF AND2(MESSAGE(I,1)=1 , I<=9) THEN
        BEGIN
          OUTLINE("**A H/W MESSAGE TO THIS O/G SIS H/W IS FOUND");
          OUTTV("**THE MESSAGE INDEX IS ",I);
          OUTTV("**RELATED TO I/C CCT. NO. ",MESSAGE(I,2));
          OUTTV("***THIS O/G CCT. NO. IS ",CCTNUM);
          GO TO CASE(I);
        END
        ELSE
        BEGIN
          OUTTV("!!! NO MESSAGE FOUND FOR THIS O/G CCT. WITH NO. ",CCTNUM);
          ERROR(13);
          OUTTEXT("*** AT TIME = ");
          OUTFIX(TIME,3,10);
          OUTIMAGE;
          OUTTV("!!! ASSOCIATED I/C CCT. NO. IS ",MESSAGE(I,2));
          OUTLINE("!!! FATAL ERROR-- RUN ABBORTED!!!!");
          OUTLINE("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
          GO TC E1;
        END **ACTIVATION OF O/G SIS H/W IN ERROR **;
        M1: MESSAGE(1,1):=0;
        TIMEOUT := UNIFORM(1.0, 34500.0, SD3);
        HOLD(TIMEOUT);
        COMMENT** SEIZURE TIME ***;
        MYSIS.MESSAGE(12,1):=1;
        MYSIS.MESSAGE(12,2):=MESSAGE(1,2);
      MYSIS.MESSAGE(12,3):=CCTNUM ;
      PASSIVATE;
      COMMENT** T/C1 TO O/G SIS REP***;
      GO TO A;
      M2: MESSAGE(2,1):=0;
      TIMECUT := UNIFORM(1.0, 50000.0, SD4);
      HOLD(TIMECUT);
      COMMENT** DIGIT1 TIME**;
      MYSIS.MESSAGE(13,1):=1;
      MYSIS.MESSAGE(13,2):=MESSAGE(2,2);
      COMMENT** T/O2 TO O/G SIS REP **;
      PASSIVATE;
      GO TO A;
      M3: MESSAGE(3,1):=0;
      TIMEOUT := UNIFORM(1.0, 170000.0, SD5);
      HOLD(TIMECUT);
      COMMENT ***DIGIT2 TIME **;
      MYSIS.MESSAGE(14,1):=1;
      MYSIS.MESSAGE(14,2):=MESSAGE(3,2);
      COMMENT** T/O3 TO O/G SIS REP ***;
      PASSIVATE;
      GO TC A;
      M4: MESSAGE(4,1):=0;
      TIMEOUT := UNIFORM(1.0, 90000.0, SD6);
      HOLD(TIMECUT);
      COMMENT ** DIGIT3 TIME **;
      MYSIS.MESSAGE(15,1):=1;
      MYSIS.MESSAGE(15,2):=MESSAGE(4,2);
      COMMENT ** T/O4 TO O/G SIS REP ***;
      PASSIVATE;
      GO TO A;
      M5: MESSAGE(5,1):=0;
      TIMEOUT := UNIFORM(1.0, 150000.0, SD7);
      HOLD(TIMEOUT);
      COMMENT*** DIGIT4 TIME **;
      MYSIS.MESSAGE(16,1):=1;
      MYSIS.MESSAGE(16,2):=MESSAGE(5,2);
      COMMENT *** T/O5 TO O/G SIS REP **;
      PASSIVATE;
      GO TO A;
      M6: MESSAGE(6,1):=0;
      TIMEOUT := UNIFORM(1.0, 120000.0, SD8);
      HOLD(TIMECUT);
      COMMENT ** DIGIT5 TIME**;
      MYSIS.MESSAGE(17,1):=1;
      MYSIS.MESSAGE(17,2):=MESSAGE(6,2);
      COMMENT** T/O6 TO O/G SIS REP**;
      PASSIVATE;
      GO TO A;
      M7: MESSAGE(7,1):=0;
      TIMEOUT := UNIFORM(1.0, 70000.0, SD9);
      HOLD(TIMEOUT);
      COMMENT ** DIGIT6 TIME **;
      MYSIS.MESSAGE(18,1):=1;
      MYSIS.MESSAGE(18,2):=MESSAGE(7,2);
      MYSIS.MESSAGE(18,3):=CCTNUM;
      COMMENT ** T/O7 TO O/G SIS REP **;
      SISHWINC(MESSAGE(7,2)).RESPONSETIME(5) := TIME;
      ACTIVATE CALL(MYCALLREC.CIRCUITNUM) AFTER CURRENT;
      GO TO LASTACTION;
      M8: MESSAGE(8,1):=0;
      MYSIS.MESSAGE(19,1):=1;
      MYSIS.MESSAGE(19,2):=MESSAGE(8,2);
      MYSIS.MESSAGE(19,3):=CCTNUM;
      COMMENT ***"ANSWER" TO O/G SIS S/W REP**;
      GO TO LASTACTION;
      M9: MESSAGE(9,1):=0;
      SENDBACK := UNIFORM(1.0, 3000.0, SD13);
      HOLD(SENDBACK);
      COMMENT ** TIME TO SEND BACK"BUSY" SIGNAL**;
      SENDFREE := UNIFORM(1.0, 100000.0, SD14);
      HOLD(SENDFREE);
      COMMENT ** TIME TO SEND"FREE"SIGNAL **;
      MYSIS.MESSAGE(20,1):=1;
      MYSIS.MESSAGE(20,3):=CCTNUM;
      MYSIS.MESSAGE(20,2):=MESSAGE(9,2);
      COMMENT **"FREE" MESSAGE TO O/G SIS REP**;
      LASTACTION:
      SISHWOUT(CCTNUM)  :- NONE;
      MYCALLREC :- NONE;
  END **CALL OF THIS SIS H/W ***;                                      E151




  COMMENT*****************  HYPOTHETICAL DMNSC ********************
  *
  *THE FOLLOWING 11 PROCESSES SIMULATE THE FUNCTIONING OF THE
  *HYPOTHETICAL DMNSC -SEE CHAPTER 6 OF THESIS.THE PROCESSES TASK
  *INDEX TABLES HAVE BEEN DESIGNED TO INSURE THAT THE RESULTING INTER-
  *PROCESS COMMUNICATION EXERCISES ALL THE SERVICES OFFERED BY THE
  *OPERATING SYSTEM, SO THAT THE DETAILED OUTPUT TRACE COULD BE USED
  *TO VERIFY THE LOGIC OF THE SIMULATOR.
  *THE PROCESSES MAY BE DIVIDED INTO 4 FUNCTIONAL GROUPS AS FOLLOWS:
  *LINE CIRCUIT HANDLER:
  *    DIRECTLY INTERFACES WITH THE INCOMMING AND OUTGOING LINES AND
```

```
      *    DEALS WITH ALL THE SIGNALLING ASPECTS I.E. LCH1, LCH2.                03396000
      *SWITCH HANDLER:                                                           03397000
      *     INTERFACES WITH THE EXCHANGE TRUNCKING AND HANDLES ALL SET UP AND    03398000
      *     CLEAR DOWN FUNCTIONS   I.E. SH .                                     03399000
      *CALL CONTROL:                                                             03400000
      *    FOR MONITCRING AND PROGRESSING ALL ASPECTS OF THE CALL,INCLUDING      03401000
      *    THE FUNCTICNS OF NETWORK ROUTING,CALL SUPERVISION AND TRAFFIC         03402000
      *    RECORDING I.E. NT, ICS, ILCR, TR, CCS, OLCR.                          03403000
      *MF TYPE CALLS:                                                            03404000
      *    FOR THE SELECTION OF A FREE REGISTSR VIA THE S A I.E. IRSP,           03405000
      *    OKSP, MF FEC OR SEN ARE NEEDED FOR MF SIGNALLING                      03406000
      *THE DMNSC HANDLRS TRUNK AND JUNCTION TRAFFIC ONLY                         03407000
      ***************************** COMMENT ENDS ************************;     03408000
                                                                                 03409000
                                                                                 03410000
      AP CLASS TR;                                                               03411000
      COMMENT***TRAFFIC RECORDING,PI=21***;                                      03412000
      BEGIN                                                                      03413000 B154
        START: SEEK(13);                                                           03414000
        IF CC>0 THEN HOLD(800.0);                                                  03415000
        BLOCK( 15 );                                                               03416000
        WHILE CC>0 DO                                                              03417000
        BEGIN                                                                      03418000 B155
          HOLD(200.0);                                                               03419000
          BLOCK( 15 );                                                               03420000
        END;                                                                       03421000 E155
        PASSIVATE;                                                                 03422000
        GOTO START;                                                                03423000
      END***TR***;                                                               03424000 E154
                                                                                 03425000
                                                                                 03426000
                                                                                 03427000
      AP CLASS LCH1(PI);INTEGER PI;                                              03428000
      COMMENT***LINE CIRCUIT HANDLER1 PI=11***;                                  03429000
      BEGIN                                                                      03430000 B156
        PROCEDURE LASTJOB;                                                         03431000
        BEGIN                                                                      03432000 B157
          BLOCK( 15 );                                                               03433000
          WHILE CC>0 DO                                                              03434000
          BEGIN                                                                      03435000 B158
            HOLD(400.0);                                                               03436000
            BLOCK( 15 );                                                               03437000
          END;                                                                       03438000 E158
          PASSIVATE;                                                                 03439000
          GOTO START;                                                                03440000
        END**LASTJOB**;                                                            03441000 E157
        START: BLOCK(3);                                                           03442000
        IF CC>0 THEN                                                               03443000
        BEGIN                                                                      03444000 B159
          PASSIVATE;                                                                 03445000
          GOTO START;                                                                03446000
        END                                                                        03447000 E159
        ELSE                                                                       03448000
        BEGIN                                                                      03449000 B160
          G(0):=1;                                                                   03450000
          HAND;                                                                      03451000
          SEEK(10);                                                                  03452000
          IF CC>0 THEN                                                               03453000
          BEGIN                                                                      03454000 B161
            HOLD(100.0);                                                               03455000
            G(0):=3;                                                                   03456000
            G(2):=6;                                                                   03457000
            HAND;                                                                      03458000
            FETCH(3);                                                                  03459000
            IF CC=0 THEN LASTJOB                                                       03460000
            ELSE                                                                       03461000
            BEGIN                                                                      03462000 B162.
              COMMENT***TASK(1) STORES I/C TI***;                                        03463000
              IF PA.TASKFOUND.TASK(1)=1 THEN LASTJOB                                     03464000
              ELSE                                                                       03465000
              BEGIN                                                                      03466000 B163
                G(0):=2;                                                                   03467000
                HAND;                                                                      03468000
                LASTJOB;                                                                   03469000
              END;                                                                       03470000 E163
            END;                                                                       03471000 B162
          END                                                                        03472000 E161
          ELSE                                                                       03473000
          BEGIN                                                                      03474000 B164
            HOLD(200.0);                                                               03475000
            G(0):=1;                                                                   03476000
            SELF(7);                                                                   03477000
            LASTJOB;                                                                   03478000
          END;                                                                       03479000 E164
        END;                                                                       03480000 E160
      END***LCH1***;                                                             03481000 E156
                                                                                 03482000
                                                                                 03483000
                                                                                 03484000
      AP CLASS ICS;                                                              03485000
      COMMENT***INCOMING CALL SUPERVISION,PI=14***;                              03486000
      BEGIN                                                                      03487000 B165
        SWITCH TI := LEGA, LEGB, LEGC;                                             03488000
        PROCEDURE LASTJOB;                                                         03489000
        BEGIN                                                                      03490000 B166
          BLOCK( 15 );                                                               03491000
          WHILE CC>0 DO                                                              03492000
```

```
    BEGIN                                                    03493000 B167
      HOLD(200.));                                             03494000
      BLCC K( 15);                                             03495000
    END;                                                     03496000 E167
    PASSIVATE;                                               03497000
    GOTO STAFT;                                              03498000
  LND***LASTJCB***;                                        03499000 E166
  START: HLCCK(3);                                         03500000
  IF CC=0 THEN                                             03501000
  BEGIN                                                    03502000 B168
    PASSIVATE;                                               03503000
    GOTO STAFT;                                              03504000
  END                                                      03505000 E168
  ELSE                                                     03506000
  BEGIN                                                    03507000 B169
    G(0):=2;                                                 03508000
    HAND;                                                    03509000
    HOLD(400.0);                                             03510000
    FETCH(6);                                                03511000
    IF CC>0 THEN                                             03512000
    HLGIN                                                    03513000 B170
      COMMENT**TASK(1) STORES I/C TI***;                       03514000
      GO TC TI(PA.TASKFOUND.TASK(1));                          03515000
      LEGA: G(0):=1;                                           03516000
      HAND;                                                    03517000
      HOLD(600.0);                                             03518000
      LASTJCB;                                                 03519000
      LEGB: G(0):=3;                                           03520000
      HAND;                                                    03521000
      HOLD(500.0);                                             03522000
      LASTJCB;                                                 03523000
      LEGC: G(0):=2;                                           03524000
      SELF(6);                                                 03525000
      G(0):=5;                                                 03526000
      HAND;                                                    03527000
    END**CC70**                                              03528000 E170
    ELSE                                                     03529000
    BEGIN                                                    03530000 B171
      G(0):=4;                                                 03531000
      HAND;                                                    03532000
      HOLD(700.0);                                             03533000
      G(0):=6;                                                 03534000
      HAND;                                                    03535000
      LASTJOB;                                                 03536000
     END**CC NGO***;                                         03537000 E171
  END**ARRIVAL OF UNBLOCKING TASK***;                      03538000 E169
END**ICS***;                                             03539000 E165
                                                         03540000
                                                         03541000
                                                         03542000
AP CLASS ILCS;                                           03543000
COMMENT***INCCMING LINE CIRCUIT ROUTINER,PI=19***;       03544000
BEGIN                                                    03545000 B172
  PROCEDURE LASTJOB;                                       03546000
  BEGIN                                                    03547000 B173
    HOLD(1200.0);                                            03548000
    BLOCK(15);                                               03549000
    WHILE CC>0 DO                                            03550000
    BEGIN                                                    03551000 B174
      HOLD(100.0);                                             03552000
    END;                                                     03553000 E174
    PASSIVATE;                                               03554000
    GOTO STAKT;                                              03555000
  END**LASTJCB***;                                         03556000 E173
  START: BLCCK(3);                                         03557000
  IF CC=0 THEN                                             03558000
  BEGIN                                                    03559000 B175
    PASSIVATE;                                               03560000
    GOTO STAKT;                                              03561000
  END**NO PERIODIC UNBLOCKING TASK***                      03562000 E175
  ELSE                                                     03563000
  BEGIN                                                    03564000 B176
    SEEK(6);                                                 03565000
    IF CC>0 THEN                                             03566000
    BEGIN                                                    03567000 B177
      G(0):=2;                                                 03568000
      HAND;                                                    03569000
      HOLD(200.0);                                             03570000
      LASTJOB;                                                 03571000
    END**CC>0***                                             03572000 E177
    ELSE                                                     03573000
    BEGIN                                                    03574000 B178
      FETCH(10);                                               03575000
      IF CC>0 THEN                                             03576000
      BEGIN                                                    03577000 B179
        IF PA.TASKFOUND.TASK(1)=3 THEN                           03578000
        BEGIN                                                    03579000 B180
          G(0):=3;                                                 03580000
          HAND;                                                    03581000
          LASTJOH;                                                 03582000
        END**TASK 1 =3***                                        03583000 E180
        ELSE                                                     03584000
        BEGIN                                                    03585000 B181
          IF PA.TASKFOUND.TASK(1)=1 THEN                           03586000
          BEGIN                                                    03587000 B182
            G(0):=1;                                                 03588000
            HAND;                                                    03589000
            LASTJOB;                                                        03590000
          END                                                             03591000 E182
        END**TASK 1 NE3***                                              03592000 E181
      END**CC>0 AFTER FETCH 10 ***                                    03593000 B179
      ELSE                                                            03594000
      LASTJOB;                                                        03595000
    END** CC=0 AFTER SEEK 10                                        03596000 E178
  END**ARRIVAL OF PERIODIC UNBLOCKING TASK FROM INTIM             03597000 E176
  OF P=3***;                                                      03598000
END***ILCR***;                                                  03599000 E172
COMMENT** PERICDIC UNBLOCKING TASKS FRCM INTIM ARE              03600000
CHARACTARIZED BY:-                                              03601000
1- TASK PRIORITY=3                                              03602000
2- I/O TI=0 ****;                                               03603000
                                                                03604000
                                                                03605000
                                                                03606000
AP CLASS IRSP;                                                  03607000
COMMENT***INCOMING REGISTER SELECTION PROCESS PI=16***;         03608000
BEGIN                                                           03609000 B183
  PROCEDURE LASTJOB;                                              03610000
  BEGIN                                                           03611000 B184
    HOLD(1300.0);                                                   03612000
    BLOCK(15);                                                      03613000
    WHILE CC>0 DO BLOCK(15);                                        03614000
    PASSIVATE;                                                      03615000
    GOTO START                                                      03616000
  END***LASTJOB***;                                             03617000 E184
  START:BLOCK(3);                                               03618000
  IF CC=0 THEN                                                  03619000
  BEGIN                                                         03620000 B185
    PASSIVATE;                                                      03621000
    GOTO START;                                                     03622000
  END**NO ARRIVAL OF UNBLOCKING TASK***                         03623000 E185
  ELSE                                                          03624000
  BEGIN                                                         03625000 B186
    FETCH(11);                                                      03626000
    IF CC=0 THEN                                                    03627000
    BEGIN                                                           03628000 B187
      HOLD(100.0);                                                    03629000
      LASTJOB;                                                        03630000
    END                                                             03631000 E187
    ELSE                                                            03632000
    BEGIN                                                           03633000 B188
      IF CC>0 THEN                                                    03634000
      BEGIN                                                           03635000 B189
        IF PA.TASKFOUND.TASK(1)=2 THEN                                  03636000
        BEGIN                                                           03637000 B190
          G(0):=2;                                                        03638000
          HAND;                                                           03639000
          LASTJOB;                                                        03640000
        END**TI=2**                                                     03641000 E190
        ELSE                                                            03642000
        BEGIN                                                           03643000 B191
          IF PA. TASKFOUND.TASK(1)=3 AND                                  03644000
          PA.TASKFOUND.TASK(5)=6 THEN                                     03645000
          BEGIN                                                           03646000 B192
            G(0):=1;                                                        03647000
            G(2):=9;                                                        03648000
            HAND;                                                           03649000
            LASTJOB;                                                        03650000
          END                                                             03651000 E192
          ELSE                                                            03652000
          BEGIN                                                           03653000 B193
            G(0):=1;                                                        03654000
            G(2):=7;                                                        03655000
            HAND;                                                           03656000
            LASTJOB;                                                        03657000
          END;                                                            03658000 E193
        END**TI=3**;                                                    03659000 E191
      END;                                                            03660000 E189
    END**CC>0***;                                                   03661000 E188
  END**UNBLOCKING TASK**;                                         03662000 E186
END**IRSP***;                                                   03663000 E183
                                                                03664000
                                                                03665000
                                                                03666000
AP CLASS NR;                                                    03667000
COMMENT*** NETWORK ROUTING,PI=18***;                            03668000
BEGIN                                                           03669000 B194
  START: SEEK(7);                                                 03670000
  IF CC=0 THEN                                                    03671000
  BEGIN                                                           03672000 B195
    HOLD(300.0);                                                    03673000
    BLOCK(15);                                                      03674000
    WHILE CC=0 DO BLOCK(15);                                        03675000
    PASSIVATE;                                                      03676000
    GOTO START;                                                     03677000
  END**CC>0**                                                   03678000 E195
  ELSE                                                          03679000
  BEGIN                                                         03680000 B196
    G(0):=1;                                                        03681000
    HAND;                                                           03682000
    HOLD(800.0);                                                    03683000
    BLOCK(15);                                                      03684000
    WHILE CC>0 DO BLOCK(15);                                        03685000
    PASSIVATE;                                                      03686000
```

```
          GOTO START;
        END**CC>0 AFTER SEEK 7 ***;                              E196
      END**NR***;                                                E194



      AP CLASS SH;
      COMMENT***SWITCH HANDLER PI=13***;
      BEGIN                                                      B197
        SWITCH SW:=TI1,TI2,TI3,TI4,TI5,TI6;
        START: BLOCK(3);
        IF CC=0 THEN
        BEGIN                                                    B198
          PASSIVATE;
          GOTO START;
        END                                                      E198
        ELSE
        BEGIN                                                    B199
          AGAIN:FETCH(14);
          IF CC=0 THEN
          BEGIN                                                  B200
            BLOCK(15);
            WHILE CC>0 DO BLOCK(15);
            PASSIVATE;
            GOTO START;
          END                                                    E200
          ELSE
          IF CC>0 THEN
          BEGIN                                                  B201
            GO TO SW(PA.TASKFOUND.TASK(1));
             TI1:G(0):=1;
            HAND;
            HOLD(700.0);
            GOTO AGAIN;
             TI2:G(0):=3;
            HAND;
            HOLD(600.0);
            GOTO AGAIN;
             TI3:G(0):=2;
            HAND;
            HOLD(1100.0);
            GOTO AGAIN;
             TI4:G(0):=4;
            HAND;
            HOLD(1200.0);
            GOTO AGAIN;
            TI5:G(0):=5;
            HAND;
            HOLD(1300.0);
            GOTO AGAIN;
             TI6:G(0):=6;
            HAND;
            HOLD(1300.0);
            GOTO AGAIN;
          END;                                                   E201
        END**UNBLOCKING TASK***;                                 E199
      END**SH***;                                                E197



      AP CLASS OCS;
      COMMENT***OUTGOING CALL SUPERVISION,PI:=15***;
      BEGIN                                                      B202
        SWITCH PT:=ICTI1,ICTI2,ICTI3,ICTI4;
        START: BLOCK(3);
        IF CC=0 THEN
        BEGIN                                                    B203
          PASSIVATE;
          GOTO START;
        END                                                      E203
        ELSE
        BEGIN                                                    B204
          FIN:  FETCH(13);
          IF CC=0 THEN
          BEGIN                                                  B205
            BLOCK(15);
            WHILE CC>0 DO BLOCK(15);
            PASSIVATE;
            GOTO START;
          END**CC=0**                                            E205
          ELSE
          BEGIN                                                  B206
            IF CC>0  THEN
            BEGIN                                                B207
              GO TO PT(PA.TASKFOUND.TASK(1));
              ICTI1:  G(0):=2;
              HAND;
              HOLD(600.0);
              GOTO FIN;
              ICTI2:  G(0):=1;
              HAND;
              HOLD(800.0);
              GOTO FIN;
              ICTI3:  G(0):=4;
              HAND;
              HOLD(1000.0);
              GOTO FIN;
            ICTI4:  G(0):=3;                                                    03784000
            HAND;                                                               03785000
            HOLD(1000.0);                                                       03786000
            GOTO FIN;                                                           03787000
          END;                                                                03788000 E207
        END**CC>0**;                                                        03789000 E206
      END**UNBLOCKING TASK ARRIVES***;                                    03790000 E204
    END**OCS***;                                                        03791000 E202
                                                                        03792000
                                                                        03793000
                                                                        03794000
    AP CLASS OLCR;                                                      03795000
    COMMENT***OUTGOING LINE CIRCUIT ROUTINER,*PI=20***;                 03796000
    BEGIN                                                               03797000 B208
      SWITCH P:=L1,L2,L3;                                                 03798000
      BLK:   BLCCK(15);                                                   03799000
      IF CC=0 THEN                                                        03800000
      BEGIN                                                               03801000 B209
        PASSIVATE;                                                          03802000
        GOTO BLK;                                                           03803000
      END                                                                 03804000 E209
      ELSE                                                                03805000
      BEGIN                                                               03806000 B210
        IF CC>0 THEN                                                        03807000
        BEGIN                                                               03808000 B211
          GO TC P(PA.TASKFOUND.TASK(1));                                      03809000
          L1:  G(0):=3;                                                       03810000
          HAND;                                                               03811000
          HOLD(1100.0);                                                       03812000
          GOTO BLK;                                                           03813000
          L2:  G(0):=1;                                                       03814000
          HAND;                                                               03815000
          HOLD(1300.0);                                                       03816000
          GOTO BLK;                                                           03817000
          L3:  G(0):=2;                                                       03818000
          G(3):=6;                                                            03819000
              COMMENT***G(3)VALUE SET HERE****;                               03820000
          HAND;                                                               03821000
          HOLD(1400.0);                                                       03822000
          GOTO BLK;                                                           03823000
        END;                                                                03824000 E211
      END**CC>0**;                                                        03825000 E210
    END**OLCR***;                                                       03826000 E208
                                                                        03827000
                                                                        03828000
                                                                        03829000
    AP CLASS ORSP;                                                      03830000
    COMMENT***OUTGOING REGISTER SELECTION PROCESS**PI=17***;            03831000
    BEGIN                                                               03832000 B212
      SWITCH PT:=LEG1,LEG2;                                               03833000
      BLOK:  BLCCK(15);                                                   03834000
      IF CC=0 THEN                                                        03835000
      BEGIN                                                               03836000 B213
        PASSIVATE;                                                          03837000
        GOTO BLCK;                                                          03838000
      END                                                                 03839000 E213
      ELSE                                                                03840000
      BEGIN                                                               03841000 B214
        IF CC>0 THEN                                                        03842000
        BEGIN                                                               03843000 B215
          G(0):=1;                                                            03844000
          SELF(7);                                                            03845000
          HOLD(1400.0);                                                       03846000
          GO TC PT(PA.TASKFOUND.TASK(1));                                     03847000
          LEG1:G(0):=1;                                                       03848000
          HAND;                                                               03849000
          HOLD(400.0);                                                        03850000
          GOTO BLCK;                                                          03851000
          LEG2:G(0):=2;                                                       03852000
          HAND;                                                               03853000
          HOLD(600.0);                                                        03854000
          GOTO BLCK;                                                          03855000
        END;                                                                03856000 E215
      END**CC>0**;                                                        03857000 E214
    END**ORSP***;                                                       03858000 E212
                                                                        03859000
                                                                        03860000
                                                                        03861000
    AP CLASS LCH2;                                                      03862000
    COMMENT***LINE CIRCUIT HANDLER2 ,PI=12***;                          03863000
    BEGIN                                                               03864000 B216
      SWITCH SW:=LG1,LG2,LG3;                                             03865000
      START: BLOCK(3);                                                    03866000
      IF CC=0 THEN                                                        03867000
      BEGIN                                                               03868000 B217
        PASSIVATE;                                                          03869000
        GOTO START;                                                         03870000
      END                                                                 03871000 E217
      ELSE                                                                03872000
      BEGIN                                                               03873000 B218
        BLCK:    BLOCK(15);                                                 03874000
        IF CC=0 THEN                                                        03875000
        BEGIN                                                               03876000 B219
          PASSIVATE;                                                          03877000
          GOTO START                                                          03878000
        END                                                                 03879000 E219
        ELSE                                                                03880000
        BEGIN                                                          03881000 B220
          IF PA.TASKFOUND.TASK(1) EQ 0 THEN GOTO BLCK                    03882000
          ELSE                                                           03883000
          BEGIN                                                          03884000 B221
            GO TC SW(PA.TASKFOUND.TASK(1));                                03885000
            LG1:  G(0):=2;                                                 03886000
            SELF(7);                                                       03887000
            G(0):=1;                                                       03888000
            HAND;                                                          03889000
            HOLD(1100.0);                                                  03890000
            GOTC BLCK;                                                     03891000
            LG2:IF G(3)=6 THEN                                             03892000
            BEGIN                                                          03893000 B222
                COMMENT**G(3)VALUE TESTED HERE***;                           03894000
              G(0):=2;                                                       03895000
              G(3):=9;                                                       03896000
              HAND;                                                          03897000
              HCLD(900.0);                                                   03898000
              GOTC BLCK;                                                     03899000
            END**G 3 =6**                                                  03900000 E222
            ELSE                                                           03901000
            BEGIN                                                          03902000 B223
              G(0):=2;                                                       03903000
              G(3):=5;                                                       03904000
              HAND;                                                          03905000
              HCLD(900.0);                                                   03906000
              GOTC BLCK;                                                     03907000
            END**G 3 NE 6**;                                               03908000 E223
            LG3:  G(0):=3;                                                 03909000
            HAND;                                                          03910000
            HOLD(600.0);                                                   03911000
            GOTO BLCK;                                                     03912000
          END**CC>0**;                                                   03913000 E221
        END;                                                           03914000 E220
      END*UNBLOCKING TASK***;                                        03915000 E218
    END**LCH2***;                                                  03916000 E216
                                                                   03917000
                                                                   03918000
                                                                   03919000
    COMMENT                                                        03920000
                                                                   03921000
                                                                   03922000
                      I N I T I A L I S A T I O N  1               03923000
                      ==============================               03924000
                                                                   03925000
    ;                                                              03926000
    COMMENT** THE FOLLOWING IS AN EXAMPLE OF INITIALISING A MODEL  03927000
    * OF A MEMBER OF SYSTEM X FAMILY OF EXCHANGES - THE DMNSC *******; 03928000
    CALLSGEN :- NEW CALLGENERATOR;                                 03929000
    COMMENT**NOW CREATE PROCESSES INSTANCES***;                    03930000
    P(25):-DSSHANDLER:-NEW DSSSW(25);                              03931000
    P(25).T :- CCPY("DSSHANDLER");                                 03932000
    FOR K:= 0 STEP 1 UNTIL 1 DO                                    03933000
    BEGIN                                                          03934000 B224
      P(26 + K) :- NEW SISSW(26 + K);                                03935000
      P(26 + K).T :- COPY("SISSW");                                  03936000
    END;                                                           03937000 E224
    DIGITALSWITCH :- NEW DSSHW;                                    03938000
    FOR K := 0 STEP 1 UNTIL 3 DC                                   03939000
    BEGIN                                                          03940000 B225
      P(49 + K) :- NEW CPSSW(49 + K);                                03941000
      P(49 + K).T :- COPY("CPSSW");                                  03942000
    END;                                                           03943000 E225
    COMMENT***NOW CREATE INPUTQ,PROLOQ FOR ALL PROCESSES           03944000
    ,SET THEIR STATES TO BLOCKED AND REQUESTED TASK OF             03945000
    PRIORITY 15 ******;                                            03946000
    FOR I:=25 STEP 1 UNTIL 27,49 STEP 1                            03947000
    UNTIL 52 DO                                                    03948000
    BEGIN                                                          03949000 B226
      P(I).INPUTC :- NEW HEAD;                                       03950000
      P(I).PROLOC :- NEW HEAD;                                       03951000
      P(I).PD(8):=4;                                                 03952000
      P(I).PD(9):=15;                                                03953000
    END;                                                           03954000 E226
    FOR I:=1 STEP 1 UNTIL 10 DO                                    03955000
    FOR J:=0 STEP 1 UNTIL 2 DO                                     03956000
    INTRIP.PPTABLE(I,J+2):=25+J;                                   03957000
    COMMENT***PERIODIC PROCESSES TABLE IS INITIALIZED              03958000
    WITH DSSSW AND SISSW HAVING PERIODICITY OF 10 MSEC             03959000
    ****NEXT INITIALIZE PROCESSES TIT'S*****;                      03960000
    FOR I:=2 STEP 1 UNTIL 6 DO                                     03961000
    FOR J:=1 STEP 1 UNTIL 4 DO                                     03962000
    BEGIN                                                          03963000 B227
      P(26).TIT(I,J):=ININT;                                         03964000
      P(25).TIT(I,J):=P(27).TIT(I,J):=P(26).TIT(I,J);                03965000
    END*** OF INITIALIZING DSSSW AND SIS REPLICATIONS;             03966000 E227
    FOR I:=25 STEP 1 UNTIL 27 DO P(I).PERIODIC:=TRUE;              03967000
    FOR I:=1 STEF 1 UNTIL 12 DO                                    03968000
    FOR J:=1 STEF 1 UNTIL 4 DO                                     03969000
    P(49).TIT(I,J):=ININT;                                         03970000
    FOR I:=0 STEF 1 UNTIL 2 DO                                     03971000
    FOR J:=1 STEF 1 UNTIL 12 DO                                    03972000
    FOR K:=1 STEF 1 UNTIL 4 DO                                     03973000
    P(50+I).TIT(J,K):=P(49).TIT(J,K);                              03974000
    COMMENT*** BY SO INITIALIZINJ ,TIT CONTENTS OF ONE             03975000
    REPLICATION NEED ONLY BE READ IN, OTHERS TIT S ARE             03976000
    COPIED FROM THIS ONE*****;                                     03977000
```

```
FOR I:=1 STEP 1 UNTIL 5 DO
FOR J:=1 STEP 1 UNTIL 4 DO
P(14).TIT(I,J):=ININT;
RESPONSE(1) :- NEW DELAYSTAT("TSELEC HW-HW");
RESPONSE(2) :- NEW DELAYSTAT("TSELEC TS-TS");
RESPONSE(3) :- NEW DELAYSTAT("ST TIME HW");
RESPONSE(4) :- NEW DELAYSTAT("ST TIME TS");
RESPONSE(5) :- NEW DELAYSTAT("SWITCH HW");
RESPONSE(6) :- NEW DELAYSTAT("SWITCH SW");
RESPONSE(7) :- NEW DELAYSTAT("TRELEASE");
RESPONSE(8) :- NEW DELAYSTAT("TCLR F/D TRA");



COMMENT


                    I N I T I A L I S A T I O N   2
                    ================================

;

NUMOFPROCESSES := ININT;
P(24) :- STARTER :- NEW INITIATOR(24);
P(24).T :- COPY("INITIATOR");
FOR I := 1 STEP 1 UNTIL NUMOFPROCESSES DO
BEGIN                                                        B228
  STARTER.TIT(I,1) := I;
  STARTER.TIT(I,2) := 40 + I;
  STARTER.TIT(I,3) := 6;
  STARTER.TIT(I,4) := 10;
END** OF STARTER TIT INITIALISATION ***;                     E228
INTRIP:-NEW INTRIPLICATES;
FOR I:=1 STEP 1 UNTIL NUMOFPROCESSES DO
BEGIN                                                        B229
  P(40+I):-NEW NOFBLCALL(40+I);
  P(40+I).T:- COPY("NOFBLCALL");
  COMMENT****CREATE RASH PROCESSES AS SUSPENDED***;
  INSPECT P(40+I) DO
  BEGIN                                                      B230
    INPUTQ:-NEW HEAD;
    PROLCQ:-NEW HEAD;
    PD(8):=4;
    PD(9):=15;
  END***INSPECT  STATEMENT****;                              E230
END*** FOR STATEMENT**;                                      E229
FOR I:= 1 STEP 1 UNTIL 10 DO
FOR J:= 1 STEP 1 UNTIL NUMOFPROCESSES DO
INTRIP.PPTABLE(I, J+2) := 40+J;
INTRIP.PPTABLE(1,2) := 24;
COMMENT*** 24 IS THE PI OF INITIATOR PROCESS****
NOW CREATE INSTANCES OF INTIM AND SA *****;
COMMENT***INITIALIZE PPTABLE ACCORDING TO   NUMOFPROCESSES;
P(16):-NEW INTIM(16);
P(14):-NEW STORAGEALLOCATOR(14);
FOR I:=1 STEP 1 UNTIL NUMOFPROCESSES DO
BEGIN                                                        B231
  P(14).TIT(I,1):=I;
  P(14).TIT(I,2):=40+I;
  P(14).TIT(I,4):=3;
END;                                                         E231
FOR I:= 14,16,24 DO
BEGIN                                                        B232
  P(I).INPUTQ:-NEW HEAD;
  P(I).PROLCQ:-NEW HEAD;
  P(I).PD(8):=4;
  P(I).PD(9):=15;
END;                                                         E232
P(14).T:-COPY("STORAGE ALLOCATOR");
P(16).T:-COPY("INTIM");
COMMENT***INIALIZE ALL PROCESS AS BLOCKED***;
FREELOQ:-NEW HEAD;
SUSPLOQ:-NEW HEAD;
INTLOQ:-NEW HEAD;
FREETASKLIST:-NEW HEAD;
LPAQ:-NEW HEAD;
FLAG:=TRUE;
COMMENT***TO TRIGGER THE TRACE PROGRAM***;

SD1 := 3010101; SD2 := 5010101; SD3 := 7010101;
SD4 := 9010101; SD5 := 1030303; SD6 := 5030303;
SD7 := 7030303; SD8 := 9030303; SD9 := 1050505;
SD10:= 3050505; SD11:= 7050505; SD12:= 9050505;
SD13:= 1070707; SD14:= 3070707; SD15:= 5070707;
SD16:= 9070707; SD17:= 1090909; SD18:= 3090909;
SD19:= 5090909; SD20:= 7090909;
FOR I:=1 STEP 1 UNTIL NUMOFPROCESSES DO
BEGIN                                                        B233
  P(40+I).TIT(1,1):=1;
  P(40+I).TIT(1,2):=14;
  P(40+I).TIT(1,3):=I;
  P(40+I).TIT(1,4):=10;
END;                                                         E233
EJECT(IF LINE GE 40 THEN 1 ELSE LINE+3);
IF FLAG THEN
BEGIN                                                        B234
  OUTLINE("                PA SIMULATION RUN TRACE");
```

```
      OUTLINE("             ==========================");          04075000
      EJECT(LINE+2);                                                04076000
      OUTTV("SIMULATED TIME =",SIMPERIOD);                          04077000
    END;                                                            04078000 E234
    CLINT:-NEW CLOCKINTERRUPT;                                      04079000
           PCINTER:=1;                                              04080000
         COMMENT*** CPU(NUM) IS LCPU AT BEG. OF SIM***;             04081000
    FOR I :=1 STEP 1 UNTIL 100 DO                                   04082000
    NEW TASKBLOCK.INTO(FREETASKLIST);                               04083000
            ACTIVATE CLINT AFTER CURRENT;                           04084000
            HOLD(SIMPERIOD);                                        04085000
    EJECT(LINE+40);                                                 04086000
    OUTLINE("*************************************************");    04087000
    OUTLINE("***  APROCESS ALLOCATOR SIMULATOR OF MK2 BL SYSTEM ***");   04088000
    OUTLINE("***  ========================================== ***");   04089000
    OUTLINE("***     A.M.SALIH PLYMOUTH POLYTECHNIC JUNE, 1979  ***");   04090000
    OUTLINE("***       PARAMETERS OF THIS SIMULATOR ARE:-       ***");   04091000
    OUTLINE("***         SIMULATION LANGUAGE: SIMULA            ***");   04092000
    OUTTEXT("***         NUMBER OF CPUS =                       ***");   04093000
    SETPOS(28); CUTINT(NUM,2);                                      04094000
    OUTIMAGE;                                                       04095000
    OUTLINE("***   EXPERIMENTAL EVALUATION OF FBLOCK CALLL TO PA***");   04096000
    OUTTEXT("***            SIMULATED TIME=                     ***");   04097000
    SETPOS(30); CUTFIX(SIMPERIOD,2,10);                             04098000
    OUTIMAGE;                                                       04099000
    OUTLINE("***                                                ***");   04100000
    OUTLINE("*************************************************");    04101000
    EJECT(LINE+2);                                                  04102000
    OUTIMAGE; CUTIMAGE;                                             04103000
    OUTLINE("SOFTWARE PROCESS AND THEIR PROCESS INDICES  ");        04104000
    OUTLINE("------------------------------------------");          04105000
    OUTTV("INDEX OF INTIM PROCESS IS",16);                          04106000
    OUTTV("INDEX OF STORAGE ALLOCATOR PROCESS IS",14);              04107000
    FOR I:=1 STEP 1 UNTIL NUMOFPROCESSES DO                         04108000
    BEGIN                                                           04109000 B235
      OUTTEXT("INDEX OF NOFBLCALL");                                  04110000
      OUTINT(I,2);                                                    04111000
      OUTTV("IS",40+I);                                               04112000
    END;                                                            04113000 E235
    EJECT(LINE+2);                                                  04114000
    OUTIMAGE;                                                       04115000
    FOR I := 14,16,41 STEP 1 UNTIL  (40 + NUMOFPROCESSES) DO        04116000
    IF P(I)=/=NONE THEN                                             04117000
    BEGIN                                                           04118000 B236
      OUTTV("**STATISTICS FOR PROCESS NO  ",I);                       04119000
      OUTLINE("------------------------------------");                04120000
      OUTTV("MAX NO. OF TASKS IN I/P Q =",P(I).MAX);                  04121000
      OUTTV("MIN NO. OF TASKS IN I/P Q =",P(I).MIN);                  04122000
      OUTTV("MAX PROC ALLOCS WAITING PROLO=",                         04123000
      P(I).PMAX);                                                     04124000
      OUTTV("MIN PROCS ALLOCS WAITING FOR PROLO=",                    04125000
      P(I).PMIN);                                                     04126000
      OUTTV("TIMES PROLO INITIATED =",P(I).PNUM);                     04127000
      OUTTVR("TIME PROC ALLOCS WAITING PROLO=",                       04128000
      P(I).PRWAIT);                                                   04129000
      OUTTV("TIMES PROCESS INITIATED =",                              04130000
      P(I).INIT);                                                     04131000
      OUTIMAGE;                                                       04132000
    END;                                                            04133000 E236
    EJECT(LINE+2);                                                  04134000
    FOR J :=1 STEP 1 UNTIL NUM DO                                   04135000
    BEGIN                                                           04136000 B237
      OUTTV("**STATISTICS FOR THE PROCESS ALLOCATOR OF CPU NO.",J);   04137000
      OUTLINE("--------------------------------------------------");  04138000
      OUTTV("TIMES PROCESS ALLOCATOR CALLED=",CLINT.C(J).MYPA.PAC);   04139000
      OUTTV("TIMES PROCESS ALLOCATOR INTERRUPTED=",                   04140000
      CLINT.C(J).MYPA.PAINT);                                         04141000
      OUTTVR("THIS PA OVERHEAD=",CLINT.C(J).MYPA.PAOVERHEAD);         04142000
      OUTTVR("THIS PA FETCH AND BLOCK OVERHEAD=",CLINT.C(J).MYPA.FBOVERHEAD); 04143000
      OUTIMAGE;                                                       04144000
    END;                                                            04145000 E237
    OUTIMAGE; CUTIMAGE;                                             04146000
    OUTLINE("**STATISTICS OF FREELO **");                           04147000
    OUTLINE("---------------------------");                         04148000
    OUTTV("MAX NO. OF PROCESS ALLOCATORS WAITING FOR FREELO=",FMAX);   04149000
    OUTTV("MIN NO. OF PROCESS ALLOCATORS WAITING FOR FREELO=",FMIN);   04150000
    OUTTV("TIMES FREELO ENGAGED =",FNUM);                           04151000
    OUTTV("TIME SPENT BY PROCESS ALLOCS WAITING FOR FREELO=",       04152000
    FRWAIT);                                                        04153000
    OUTIMAGE; OUTIMAGE;                                             04154000
    OUTLINE("**STATISTICS OF SUSPLO**");                            04155000
    OUTLINE("---------------------------");                         04156000
    OUTTV("MAX PROC ALLOCS WAITING SUSPLO=",SMAX);                  04157000
    OUTTV("MIN PROC ALLOCS WAITING SUSPLO=",SMIN);                  04158000
    OUTTV("TIMES SUSPLO ENGAGED =",SNUM);                           04159000
    OUTTV("TIME SPENT BY PROC ALLOCS WAITING SUSPLO=",SUSWAIT);     04160000
    OUTIMAGE; CUTIMAGE;                                             04161000
    OUTLINE("**STATISTICS OF INTLO**");                             04162000
    OUTLINE("---------------------------");                         04163000
    OUTTV("MAX PROC ALLOCS WAITING INTLO=",IMAX);                   04164000
    OUTTV("MIN PROC ALLOCS WAITING INTLO=",IMIN);                   04165000
    OUTTV("TIMES INTLO ENGAGED =",INUM);                            04166000
    OUTTV("TIME SPENT BY PROC ALLOCS WAITING INTLO=",INWAIT);       04167000
    OUTIMAGE; CUTIMAGE;                                             04168000
    OUTLINE("**CPUS STATISTICS**");                                 04169000
    OUTLINE("-----------------------");                             04170000
    OUTIMAGE;                                                       04171000
```

```
         FOR I:=1 STEP 1 UNTIL NUM DO                                                            04172000
         BEGIN                                                                                   04173000 B238
           OUTTV("FOR CPU NO.",I);                                                                 04174000
           OUTLINE("----------------------");                                                      04175000
           OUTTVR("TIME ON BACKGROUND PROCESS =",CLINT.C(I).BKGTIME);                              04176000
           OUTTVR("PERCENTAGE CPU OCCUPANCY OF PROCESSES OTHER THAN BACKGROUND=",                  04177000
           (SIMPERIOD - (CLINT.C(I).BKGTIME + CLINT.C(I).MYPA.PAOVERHEAD))/                        04178000
           SIMPERIOD*100);                                                                         04179000
           OUTIMAGE; OUTIMAGE;                                                                     04180000
         END;                                                                                    04181000 E238
         EJECT(2);                                                                               04182000
         OUTLINE("      RESULTS WITHOUT FBLOCK(P) CALL TO PA");                                  04183000
         OUTLINE("        ===============================");                                    04184000
         OUTTEXT("FOR A MULTI-PROCESSOR SYSTEM WITH");                                           04185000
         OUTINT(NUM,3);                                                                          04186000
         OUTTEXT(" CPUS AND"); OUTINT(NUMOFPROCESSES,2);                                         04187000
         OUTLINE("NCFBLCALL PROCESSES :-");                                                      04188000
         OUTLINE("                         TIMES LOOP TRAVERSED ");                              04189000
         FOR I:=1 STEP 1 UNTIL NUMOFPROCESSES DO                                                 04190000
         BEGIN                                                                                   04191000 B239
           OUTTEXT("NCFBLCALL"); OUTINT(I,2); OUTINT(P(40+I) QUA NOFBLCALL.                        04192000
           COUNTER,12); OUTIMAGE; OUTIMAGE;                                                        04193000
         END***OF OUTPUTTING COUNTERS CONTENTS***;                                               04194000 E239
       F1: END;                                                                                 04195000 E2
     END;                                                                                      04196000 E1
```

```
IDENTIFIER      LINE

A                216* 217  220* 221  453  474  485  497 1778 1791 1915 1916 1936 1948 1984 1990 2000 2013 2024 2087
                2216 2227 2406 2429 2434 2441 2448 2579 2589 2598 2607 2615 2623 2633 2785 2794 2797 2800 2808 2821
                2841 2856 2869 2882 2895 2908 2931 2951 2964 2967 2984 3006 3016 3019 3029 3055 3070 3073 3110 3124
                3139 3144 3153 3160 3271 3302 3311 3320 3329 3338 3347
AGAIN           2145 2146 3705 3720 3724 3728 3732 3736 3740
AND2             216  217  226  227  234  236  238  240  669 1126 1131 2415 3272 3273
ANSWER          2205 2295 2439
AP               160  294  404  449  520  576  643  790  798  817  677 1774 1804 1911 1980 2396 2775 3104 3411 3428
                3485 3543 3607 3667 3693 3747 3795 3830 3862
B                159  216* 217  220* 221 1053 1059*1063 2095 2218 2294 2783 2786 2790 3196 3208 3222 3228
BACK            2293 2298
BACKGROUND       159 1741 1804
BKGTIME          405 1167*1809*4176 4178
BLCK            3874 3882 3891 3899 3907 3912
BLK             3799 3803 3813 3817 3823
BLOCK            320  473  484  496  531  540  607 1790 1988 2428 2789 3143 3416 3420 3433 3437 3442 3491 3495 3500
                3549 3557 3613 3614 3618 3674 3675 3684 3685 3697 3708 3709 3751 3762 3763 3799 3834 3866 3874
BLOCKING         792 1498
BLOK            3834 3838 3851 3855
C                158 1028*1737 1740 1746 1747 1748 1750 1753 1776 1777 1779 1788*2104 2220 2311 4139 4141 4142 4143
                4176 4178*
CALL             161 2111 2147 2150 2151 2152 2213 2308 2329 2955 2993 2994 2996 3357
CALLEDPROCES     790 1055 1059 1060 1061 1082 1084 1085 1086 1087 1092 1093 1095 1096 1098 1099 1112*1124 1126 1131
                1132 1203 1208 1209 1630 1648 1697
CALLGENERATO     166 2139 3929
CALLINGPROCE     409  647  671  675* 699  790  969 1062 1115 1116*1144 1159 1227 1242*1244 1287 1295 1296 1303 1305
                1306 1316 1319*1321 1327 1354 1358 1359 1362 1366 1368 1369 1375 1376 1385 1386 1392 1393 1394 1402*
                1404 1411 1418 1421 1422 1423 1427*1429 1436 1447 1451 1452 1455 1459 1460 1461 1476 1477 1488 1489
                1503 1507 1508 1514 1515 1516 1523 1524 1533 1534 1538 1540 1541 1545 1546 1558 1562 1563 1569 1570
                1571 1578 1579 1588 1589 1593 1595 1596 1598 1599 1610 1614 1615 1618 1619 1623 1627 1641 1643 1654
                1662 1668 1680 1684 1685 1692 1696 1697 1699 1702 1709
CALLRECORD       161 2059 2084 2147 2206 3265
CALLSGEN         166 3929
CANCEL           685  704
CARDINAL         805  821  832  853  882  916  994 1032 1057 1114 1170 1339 1362 1455
CASE            2399 2410 2778 2784 3107 3111 3266 3279
CC               371  373  546 1986 2407 2786 3111 3415 3417 3434 3443 3453 3460 3492 3501 3512 3550 3558 3566 3576
                3614 3619 3627 3634 3671 3675 3685 3698 3706 3709 3714 3752 3760 3763 3769 3800 3807 3835 3842 3867
                3875
CCS              324  333  343  365  373  403 1162 1232 1315 1400 1424 1542 1597
CCTFREE         2205 2324 2446
CCTNUM           224* 226* 227* 231* 234* 236* 238* 240*2201*2211 2213 2215 2236 2241 2246 2251 2253 2260 2262 2263
                2270 2271 2276 2277 2282 2283 2288 2289 2305 2306 2308 2319 2320 2329 2331 3260 3262 3270 3278 3283
                3299 3354 3362 3373 3377
CIRCUITNUM      2059 2061 2067 2070 2098 2111 3357
CIX              298  314  322  332  342  352  363 1062 1144 1619
CLEARPATH       3156 3193 3211 3213
CLEARTIME       3195 3214 3215
CLINDEX          792 1144
CLINT            164 1028*4079 4084 4139 4141 4142 4143 4176 4178*
CLOCK            698  789 1182 1207
CLCCKINT         640  650  690  710 1756
CLOCKINTERRU     164 1735 4079
CLOSEFILE        459  475 1919 1926 1994 2003
COPY            1743 3932 3936 3942 4002 4014 4046 4047
COUNTER          523  550* 552* 554 1914 1934*1935 1946*1947 1983 2011*2012 2022*2023 2141 2153*2154 4193
CPSREP           231  234 2454 2464 2474 2486 2498 2510 2522 2534 2546 2562 2674 2685 2695 2936 2956 2998 3008 3047
                3058 3120 3134
CPSSW           2775 3941
CPU              158  302  401  410  642  793 1737 1740
CPUSCHED        1022 1212 1549 1600
CUCP             643  673*1257 1327
CURP             404  409  656* 941  939 1008*1028*1747
CURRENT          315  325  335  345  356  366  674  688  707 1250 1256 1265 1323 1328 1333 1405 1408 1412 1430 1433
                1437 1662 1665 1669 1703 1706 1710 1758 2091 2100 2108 2152 2308 2329 2440 2447 2578 2588 2597 2606
                2614 2622 2631 2641 2950 3152 3159 3357 4084
CUSP             643  647  682  684* 685  699  701  703* 704
D                298  355 1680 2414 2459 2469 2479 2491 2504 2516 2528 2540 2552 2556 2568 2642 2647 2652 2657 2662
                2667 2680 2691 2701
DELAYSTAT        168 1852 3981 3982 3983 3984 3985 3986 3987 3988
DIGITALSWITC     167 3148 3149 3150 3152 3156 3157 3158 3159 3938
DSSHANDLER       165 3202 3203 3204 3217 3218 3219 3931
DSSHW            167 3190 3938
```

```
IDENTIFIER       LINE

DSSSW             165 3104 3931
EJECT            4071 4076 4086 4102 4114 4134 4182
EMPTY            1014 1057 1101 1418
ERROR             197  652  935  983 1297 1623 2232 2297 2314 2326 3226 3284
ESCAPE            658  677  713
EVTIME            684  703
E1               1779 1789 2233 3291 4195
FAILURE          1518 1530
FBLFAIL          1573 1585
FBLOCK            361 1915
FBLOCKING         792 1553
FBOVERHEAD        788 1243*1320*1403*1428*4143
FETCH             330  544 1985 2406 2785 3110 3459 3511 3575 3626 3705 3759
FETCHING          792 1350
FIN              3759 3775 3779 3783 3787
FIRST             844  863  864  872  905  958 1016 1047 1087 1096 1181 1246 1253 1256 1260 1295 1296 1300 1322 1324
                 1328 1331 1368 1369 1394 1404 1406 1407 1410 1423 1429 1431 1432 1435 1460 1461 1515 1516 1546 1570
                 1571 1599 1661 1663 1664 1667 1702 1704 1705 1708
FLAG              153  453  464  477  494  553  581  598  648  676  679  693  708  893  928  944 1003 1042 1110 1149
                 1217 1270 1303 1335 1356 1373 1383 1449 1474 1476 1486 1505 1521 1531 1560 1576 1586 1612 1682 1757
                 1935 1947 2012 2023 4054 4072
FMAX              151  833* 854*4149
FMIN              151  834* 855*4150
FNUM              151  837* 858*4151
FOLLOW           1108
FOUND            1389 1527 1582
FREELO            153  831  839  843  852  860  871
FREELOO           155  831  832  844  852  853  872 4049
FREELO1           828 1314 1399
FREELO2           848 1187 1626 1679
FREETASKBLOC      794  864  870 1189 1204 1629 1631 1632 1641 1645 1680 1692 1693 1696
FREETASKLIST      155  842  863  864  867 4052 4083
FRENG             151  832  833* 834* 853  854* 855*
FRSTART           787  830  835  836  851  856  857
FRWAIT            152  835* 856*4153
G                 301  313* 354* 407  454  455  457  459  466* 467* 468* 479* 480* 481* 489* 490  491* 492* 526  527
                  528  535  536  537  592  972 1161 1231 1309 1538 1593 1610 1618 1641 1643 1692 1696 1699 1783 1784
                 1916 1919 1921 1928 1929 1930 1940 1941 1942 1994 1996 2005 2006 2007 2016 2017 2018 2409 2437 2444
                 2454 2455 2456 2464 2465 2466 2474 2475 2476 2486 2487 2488 2498 2499 2500 2501 2510 2511 2512 2513
                 2522 2523 2524 2525 2534 2535 2536 2537 2546 2547 2548 2549 2562 2563 2564 2565 2571 2573 2576 2582
                 2586 2587 2592 2596 2601 2605 2610 2613 2617 2621 2626 2629 2630 2674 2675 2676 2677 2685 2686 2687
                 2688 2695 2696 2697 2698 2783 2803*2804 2805*2814 2815 2816*2824 2825 2827 2828 2829 2830 2844 2845
                 2848 2849 2850 2851 2859 2360 2861 2862 2863 2864 2872 2873 2874 2875 2876 2877 2885 2886 2887 2888
                 2889 2890 2898 2899 2900 2901 2902 2903 2913 2914 2915 2916 2917 2918 2923 2924 2925 2926 2934 2935
                 2936 2937 2938 2939 2954 2956 2957 2958 2959 2970 2971 2972 2973 2974 2975 2993 2994 2995 2996 2998*
                 2999 3000*3001 3008*3009 3010*3011*3021*3022 3023*3024*3030 3033 3034*3035 3036 3041 3042*3047 3048
                 3049 3050 3058*3059 3060*3361*3111 3120 3121*3134*3135 3136 3149 3150 3157 3158 3450 3456 3457 3467
                 3476 3508 3516 3520 3524 3526 3531 3534 3568 3580 3588 3638 3647 3648 3654 3655 3681 3717 3721 3725
                 3729 3733 3737 3772 3776 3780 3784 3810 3814 3818 3819 3844 3848 3852 3886 3888 3892 3895 3896 3903
                 3904 3909
H                 298  334 1366
HAND              310  469  482  493  529  538  596 1785 1931 1943 2008 2019 2457 2467 2477 2489 2502 2514 2526 2538
                 2550 2566 2678 2689 2699 2806 2817 2831 2852 2865 2878 2891 2904 2919 2927 2940 2960 2976 3002 3012
                 3025 3037 3043 3051 3062 3122 3137 3451 3458 3468 3509 3517 3521 3527 3532 3538 3569 3581 3589 3639
                 3649 3656 3682 3718 3722 3726 3730 3734 3738 3773 3777 3781 3785 3811 3815 3821 3849 3853 3889 3897
                 3905 3910
HANDTASKANDS     1071 1205 1647 1700
HEAD              155  309 1054 1075 1744 1745 3950 3951 4018 4019 4041 4042 4049 4050 4051 4052 4053
HLD              1806 1807
HND               792 1606
HOLD              462  475  486  524  533  542  548  555  605  840  861  890  955 1012 1044 1083 1094 1157 1179 1184
                 1209 1238 1289 1293 1352 1390 1395 1418 1445 1480 1492 1500 1543 1547 1555 1608 1654 1677 1759 1806
                 1927 1933 1939 1945 2004 2010 2015 2021 2157 2238 2243 2248 2257 2267 2273 2279 2285 2302 2432 2435
                 2442 2451 2461 2472 2482 2494 2507 2519 2531 2543 2554 2559 2569 2580 2590 2599 2608 2616 2624 2635
                 2644 2649 2654 2659 2664 2669 2682 2693 2792 2795 2798 2801 2809 2819 2822 2833 2836 2839 2842 2854
                 2857 2867 2870 2880 2883 2893 2896 2906 2909 2921 2929 2932 2942 2945 2948 2952 2962 2965 2968 2978
                 2981 2983 2985 3004 3007 3014 3017 3020 3027 3030 3039 3045 3053 3056 3064 3067 3069 3071 3116 3130
                 3146 3154 3200 3215 3295 3305 3314 3323 3332 3341 3350 3367 3370 3415 3419 3436 3455 3475 3494 3510
                 3518 3522 3533 3548 3552 3570 3612 3629 3673 3683 3719 3723 3727 3731 3735 3739 3774 3778 3782 3786
                 3812 3816 3822 3846 3850 3354 3890 3898 3906 3911 4085
HPCPS             134  145  489
I                 151  206  207  211  298  312  313* 353  354* 403  668  669* 670* 671  786  899  900* 913  924  925*
                  926  938  939  946* 955  967  968  969* 978  979  980* 981  989*1007*1024 1027 1028*1160 1161*1188
```

```
IDENTIFIER     LINE

               1189 1190 1198 1199 1230 1231*1307 1309 1537 1538*1592 1593*1640 1641*1642 1643 1695 1696*1698 1699
               1738 1740*1741*1742 1743 1744 1745 1746 1747*1748*1750 1751 1776 1781 1783 2141 2145 2146 2147*2148*
               2150*2151*2152 2204 2402 2414 2415*2416 2419 2420 2421 2422 2424 2437 2438 2444 2445 2571 2572 2582
               2583 2592 2593 2601 2602 2610 2611 2617 2618 2626 2627 2780 3267 3271 3272*3273*3276 3277 3279 3288
               3947 3950 3951 3952 3953 3955 3957 3961 3964 3965*3967*3968 3970 3971 3974 3978 3980 4003 4005*4006*
               4007 4008 4011 4013*4014 4016 4024 4026 4033 4035*4036*4037 4039 4041 4042 4043 4044 4064 4066 4067
               4068*4069 4082 4108 4111 4112 4116 4117 4119 4121 4122 4124 4126 4127 4129 4131 4172 4174 4176 4178*
               4190 4192*
ICS            3485
ICTI1          3750 3772
ICTI2          3750 3776
ICTI3          3750 3780
ICTI4          3750 3784
IDLE            666  673  675  682  692  696  701 1014
ILCR           3543
IMAX            152  995*1033*1171*4164
IMIN            152  996*1034*1172*4165
INCCT          2402 2453 2454 2456 2463 2464 2466 2471 2474 2476 2481 2484 2486 2488 2493 2496 2498 2500 2506 2509
               2510 2512 2518 2521 2522 2524 2530 2533 2534 2536 2542 2545 2546 2548 2558 2562 2565 2780 2825 2826
               2829 2844 2846 2850 2859 2363 2872 2876 2885 2889 2898 2902 2913 2915 2917 2925 2934 2938 2950 2954
               2955 2958 2970 2972 2974 3034 3035 3042 3047 3049
INCCTC         3109 3133 3135 3157 3194 3217*
INCCTS         3109 3118 3120 3121 3149 3194 3202*3206
INCOMCCTS       133  139
INCSISHW        162 2064 2143 2148 2201 2404
INENG           152  994  995* 996*1032 1033*1034*1170 1171*1172*
ININT           138  139  140  141  142  144 3964 3970 3980 4000
INIT            298 1285*4131
INITIATOR       157 1774 4001
INPUTQ          309  821 1055 1099 1295 1296 1339 1362 1368 1369 1418 1455 1460 1461 1515 1516 1570 1571 1744 3950
               4018 4041
INREAL          146
INSTART         787  992  997  998 1030 1035 1036 1168 1173 1174
INT            1141 1146
INTERRUPT       681  700  789 1141 1146
INTERVAL       2142 2156 2157
INTIM           576 1191 4031
INTLO           640  993 1001 1015 1031 1039 1046 1169 1177 1180
INTLOQ          155  993  994 1016 1031 1032 1047 1169 1170 1181 4051
INTO            842  867 4083
INTRIP          156  584  586  601  988 1014*1031 1039 1040 1043 1046 1169 1177 1180 1189 1199 1213 1253 1257 1260
               1324 1327 1331 1406 1410 1431 1435 1652 1663 1667 1704 1708 1753 1756 1758 3957 4010 4026 4027
INTRIPLICATE    156  638 4010
INUM            152  999*1037*1175*4166
INWAIT          152  997*1035*1173*4167
IRSP           3607
J               151  786  951  952* 967  970  972*1366 1381 1459 1465 1514 1519 1541 1569 1596 3956 3957*3962 3964
               3965*3969 3970 3972 3974*3979 3980 4025 4026*4135 4137 4139 4141 4142 4143
JOB             817* 819
JUMP           2400 2424
K               152  786 1024 1463 1469*1480 1492 2063 2075 2076 2077 2079*2081 2082 2083 2084 3933 3935*3936 3939
               3941*3942 3973 3974*
L              1053 1057 1058 1063 1064
LAST           1056 1103 1104
LASTACTION     2292 2309 2330 3358 3364 3376
LASTJOB        3431 3460 3464 3469 3478 3489 3519 3523 3536 3546 3571 3582 3590 3595 3610 3630 3640 3650 3657
LASTSTATE       786 1215 1228 1283
LCH1           3428
LCH2           3862
LCPU            642  646  655  656* 695  989 1008*1009 1040 1043 1753
LEGA           3488 3516
LEGB           3488 3520
LEGC           3488 3524
LEG1           3833 3848
LEG2           3833 3852
LG1            3865 3886
LG2            3865 3892
LG3            3865 3909
LINE           4071*4076 4086 4102 4114 4134
LINK            415
LINKAGE        1077
LOADTASK        963 1301 1398
LOOP            524  557 2144 2161
```

```
IDENTIFIER        LINE

LPA                644  646  647  666  671  675* 681  688  692  695  696  698  699  700  707 1253 1260 1324 1331 1406
                  1410 1431 1435 1663 1667 1704 1708
LPAQ               155  666  692  696 1014 1246 1253 1256 1260 1322 1324 1328 1331 1404 1406 1407 1410 1429 1431 1432
                  1435 1661 1663 1664 1667 1702 1704 1705 1708 4053
L1                3798 3810
L2                3798 3814
L3                3798 3818
M                  298  344 1459
MAX                298  822*1024 1026 1028*1040 1856 1862 1866*1874 4121
MAXINCCT           134  141
MAXOUTCCT          134  143  161  162  163 2075 2077 2145 2987 2989
MESSAGE           2097 2098 2235 2236 2240 2241 2245 2246 2250 2251 2253 2259 2260 2263 2269 2270 2271 2275 2276 2277
                  2281 2282 2283 2287 2288 2289 2304 2305 2306 2318 2319 2320 2403 2415 2416 2420 2421 2422 2450 2453
                  2460 2463 2470 2471 2480 2481 2492 2493 2505 2506 2517 2518 2529 2530 2541 2542 2553 2557 2558 2564
                  2575 2576 2584 2586 2594 2596 2603 2605 2612 2613 2619 2621 2628 2629 2634 2637 2638 2639*2643 2648
                  2653 2658 2663 2668 2672 2674 2676 2677 2681 2685 2687 2688 2692 2695 2697 2698 3264 3272 3273 3277
                  3288 3293 3297 3298*3299 3303 3307 3308*3312 3316 3317*3321 3325 3326*3330 3334 3335*3339 3343 3344*
                  3348 3352 3353*3354 3356 3359 3360 3361*3362 3365 3372 3373 3374*
MIN                298  823*1856 1862 1864*1875 4122
MININCCT           133  140  161  162  163 2075 2077 2145 2987 2989
MINOUTCCT          134  142
MYCALLREC         2084 2151 2206 2213 2252 2253 2263 2271 2277 2283 2289 2306 2320 2332 2996 3265 3357 3378
MYCPU              410  793  931  940  941  948  972 1006 1151 1161 1162 1167*1231 1232 1245 1273 1288 1309 1315 1317
                  1321 1338 1360 1400 1424 1453 1509 1538 1542 1564 1593 1597 1616 1686
MYINCSISHW        2064 2067 2071 2083 2089 2091 2106 2108 2112 2150
MYINCSISREP       2781
MYOUTSISHW        2065 2082 2097 2098 2100 2113 2252 2253 2263 2271 2277 2283 2289 2306 2320 2993
MYPA               406  408  409  410  646  695 4139 4141 4142 4143 4178
MYSIS             2210 2211 2235 2236 2240 2241 2245 2246 2250 2251 2253 2259 2260 2263 2269 2270 2271 2275 2276 2277
                  2281 2282 2283 2287 2288 2289 2304 2305 2306 2318 2319 2320 3269 3270 3297 3298 3299 3307 3308 3316
                  3317 3325 3326 3334 3335 3343 3344 3352 3353 3354 3360 3361 3362 3372 3373 3374
M1                2400 2450 3266 3293
M10               2400 2553
M11               2400 2557
M12               2401 2634
M13               2401 2643
M14               2401 2648
M15               2401 2653
M16               2401 2658
M17               2401 2663
M18               2401 2668
M19               2401 2681
M2                2400 2460 3266 3303
M20               2401 2692
M3                2400 2470 3266 3312
M4                2400 2480 3266 3321
M5                2400 2492 3266 3330
M6                2400 2505 3266 3339
M7                2400 2517 3266 3348
M8                2400 2529 3266 3359
M9                2400 2541 3266 3365
N                  320* 323  330* 334  340* 344  350* 355
NEGATIVE          1379 1417
NEGEXP            2156
NO                 197  199  202
NOFBLCALL         1980 4013 4192
NOSUSPROC         1090 1127 1132
NOTASK            1464 1485
NR                3667
NUM                133  144  158  159 1027 1737 1738 1753 4094 4135 4172 4186
NUMBER             401* 655  931  948 1006 1009 1043 1151 1273 1338 1360 1453 1509 1564 1616 1686
NUMOFPROCESS       133 1198 1781 4000 4003 4011 4025 4033 4064 4108 4116 4187 4190
OBS               1855 1860*1862 1876
OCS               3747
OGTI               786 1610 1623 1629 1630 1631
OLCR              3795
OPENFILE           457  462 1921 1938 1996 2014
ORSP              3830
OR2                220  221  226
OUT                657  667  694  697  812  838  859  870  888  922 1000 1038 1176 1298 1389 1473
OUTCCT            2780 2824 2827 2830 2845 2348 2851 2860 2861 2864 2873 2874 2877 2886 2887 2890 2899 2900 2903 2914
                  2918 2926 2935 2936 2939 2955 2956 2959 2971 2975 2987 2988 2989 2991*2993 2994 2995 2996 3001 3030
                  3033 3036 3042 3050
```

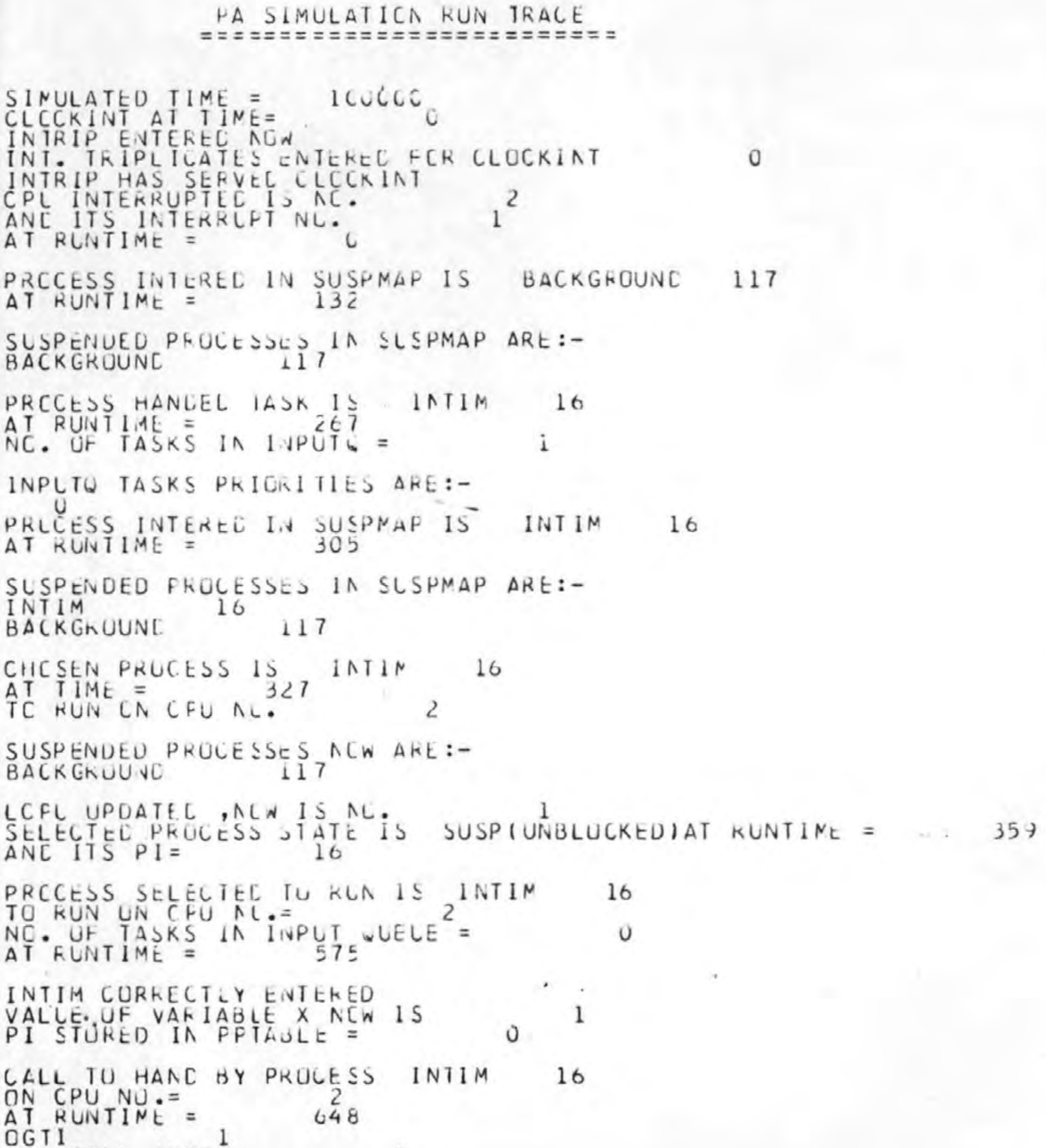
```
IDENTIFIER          LINE

OUTCCTC             3109 3134 3136 3158 3194 3218*
OUTCCTS             3109 3121 3150 3194 3203*
OUTFIX               185 3286 4098
OUTGCCTNUM          2063 2083*2204 2501 2513 2525 2537 2549 2565 2955 2994 2995
OUTGREP             2780 2811 2814
OUTGSISHW            163 2065 2079 2405 2782 2991 3260
OUTIMAGE             177  186  193  212  602  653  897  901  933  949  953 1010 1117 1122 1154 1224 1275 1310 1341 1363
                    1377 1387 1456 1478 1490 1511 1525 1535 1566 1580 1590 1620 1688 1871 3287 4095 4099 4103*4115 4132
                    4144 4146*4154*4161*4168*4171 4180*4193*
OUTINT               176  202  211 1121 1309 4094 4111 4186 4187 4192*
OUTLINE              190  461  465  478  494  581  649  677  680  709  898  930  950  984 1118 1873 1923 1999 2223 2229
                    2230 2231 2261 2418 2423 3275 3289 3290 4074 4075 4087 4088 4089 4090 4091 4092 4096 4100 4101 4104
                    4105 4120 4138 4147 4148 4155 4156 4162 4163 4169 4170 4175 4183 4184 4188 4189
OUTTEXT              175  184  192  201  209  210 1219 1220 1872 3285 4093 4097 4110 4185 4187 4192
OUTTV                171  454  554  600  601  655  693  896  900  931  932  947  948  952 1005 1006 1009 1043 1113 1114
                    1151 1152 1153 1222 1223 1273 1274 1338 1339 1340 1360 1361 1362 1453 1454 1455 1509 1510 1564 1565
                    1616 1617 1618 1619 1686 1687 1757 1935 1947 2012 2023 2069 2070 2081 2214 2215 2224 2262 2419 2420
                    2421 3276 3277 3278 3283 3288 4077 4106 4107 4112 4119 4121 4122 4123 4125 4127 4130 4137 4139 4140
                    4149 4150 4151 4152 4157 4158 4159 4160 4164 4165 4166 4167 4174
OUTTVR               181  654 1874 1875 1876 4128 4142 4143 4176 4177
P                    160  361* 364  900  939  946  952 1007 1040 1191 1203 1630 1741 1742 1743 1744 1745 1747 1748 1751
                    2211 3270 3798 3809 3931 3932 3935 3936 3941 3942 3950 3951 3952 3953 3964 3965*3967 3970 3974*3980
                    4001 4002 4013 4014 4016 4031 4032 4035 4036 4037 4041 4042 4043 4044 4046 4047 4066 4067 4068 4069
                    4117 4121 4122 4124 4126 4127 4129 4131 4192
PA                   303  315  325  335  345  356  366  943 3464 3515 3578 3586 3636 3644 3645 3716 3771 3809 3847 3882
                    3885
PAC                  786 1143*4139
PAINT                787 1147*1152 4141
PAOVERHEAD           788 1241*1318*1401*1426*1659*1701*4142 4178
PAQ                  298  805  806* 807*
PARTFILE             455  486
PASIVE              2310 2315
PASSIVATE            316  326  336  346  357  367  713 1278 1344 1413 1438 1670 1711 1810 2110 2291 2293 2310 2323 3207
                    3221 3227 3300 3310 3319 3328 3337 3346 3422 3439 3445 3497 3503 3554 3560 3615 3621 3676 3686 3700
                    3710 3754 3764 3802 3837 3869 3877
PASTART             1139 1279 1345 1414 1439 1671 1712
PD                   301  969 1059 1060 1061 1082 1126 1131 1132 1161 1162 1164 1208 1215 1220 1226 1231 1232 1538 1540
                    1541 1593 1595 1596 1648 1742 3952 3953 4020 4021 4043 4044
PERIODIC             300 1124 3967
PI                   294* 527  536  656  671  892  895  989 1008 1028*1093 1112 1116 1209 1223 1242*1272 1303 1306 1316
                    1319*1337 1359 1376 1386 1402*1427*1452 1477 1489 1508 1524 1534 1563 1579 1589 1615 1632 1654 1685
                    1929 1941 2006 2017 2803 3428*
PMAX                 299  806*4124
PMIN                 299  807*4126
PNUM                 299  810*4127
POINTER              151 1190 1191 1195 1197 1202*4080
POSITIVE            1392 1482
PPTABLE              584  586  601  641 1190 1199 3957 4026 4027
PROC                 797  798  801  802* 803  813
PROCESS              294  401  638  784 1735 2059 2139 2201 3190 3260
PROCESSALLOC         303  406  408  644  784  943 1254 1261 1325 1332 1407 1411 1432 1436 1664 1668 1705 1709
PROLO                300  802  813 1084 1095 1299 1392 1421 1545 1598
PROLOQ               309  802  805 1087 1096 1300 1394 1423 1546 1599 1745 3951 4019 4042
PROLO1               797 1098 1292 1354 1447 1503 1558
PRSTART              299  801  808  809
PRWAIT               299  808*4129
PT                  3750 3771 3833 3847
PTR                  579  584  586  601 1191
Q                   1075 1099 1100 1101 1103 1104 1114 1119 2323 2327
QLEN                 298  821  822* 823*
QLENGTH              817 1086 1393 1422
QUFUE               1054 1055 1056 1057*
RANDINT             2068 2075 2077 2145 2811 2987 2989
RASH                 520
RELEVANTCPU          302  313  324  333  343  354  365  373  454  455  457  459  466  467  468  479  480  481  489* 491
                     492  940 1040 1321 1748 1809*1916 1919 1921 1994 1996 2409 2437 2444 2571 2573 2576 2582 2586 2587
                    2592 2596 2601 2605 2610 2613 2617 2621 2626 2629 2630 2783 2805 2816 2824 2825 2844 2845 2859 2860
                    2872 2873 2885 2886 2898 2899 2913 2914 2934 2935 2954 2970 2971 2993 2994 2995 2996 2998 3000 3008
                    3010 3011 3021 3023 3024 3030 3034 3058 3060 3061 3111 3149 3150 3157 3158
REMAININGACT         304  684  703 1248 1249 1263 1264
REPORT              1869
RESPONSE             168 2573 2587 2630 2826 2846 3118 3133 3206 3981 3982 3983 3984 3985 3986 3987 3988
RESPONSETIME        2207 2255 2265 2322*2485 2497 2573 2587 2630 2672 2826 2846 3118 3133 3206 3356
```

```
IDENTIFIER      LINE

RESPONSE1       3108 3113 3115 3204
RESPONSE2       3108 3127 3129 3219
RUNNINGPROCE     791 1159 1161 1162 1164 1165 1166
RUNSELCPROC     1215 1550 1601
S                298  323 1514
SD1              134 2068 4057
SD10             135 2301 4060
SD11             135 3199 4060
SD12             135 3214 4060
SD13             135 3366 4061
SD14             135 3369 4061
SD15             135 2145 4061
SD16             135 2075 2811 4062
SD17             135 2077 4062
SD18             135 2987 4062
SD19             136 2989 4063
SD2              134 2156 4057
SD20             136 4063
SD3              134 3294 4057
SD4              134 3304 4058
SD5              134 3313 4058
SD6              134 3322 4058
SD7              135 3331 4059
SD8              135 3340 4059
SD9              135 3349 4059
SECS            1051 1063 1064 1083 1094
SEEK             340 2835 2838 2944 2947 2980 2982 3066 3068 3414 3452 3565 3670
SEEKING          792 1443
SEIZURE         2089 2205 2227
SELF             350 3477 3525 3845 3887
SELV             792 1675
SENDBACK        3268 3366 3367
SENDFREE        3268 3369 3370
SETPOS          4094 4098
SETUPPATH       3148 3193 3196 3198
SETUPTIME       3195 3199 3200
SH              3693
SIMPERIOD        137  146 1754 1806 1807 4077 4085 4098 4178 4179
SIMULATION       148
SISHWINC         162 2067 2146 2148 2150 2151 2331 2438 2445 2484 2496 2509 2521 2533 2545 2565 2573 2587 2630 2672
                2826 2846 2950 2995 3118 3133 3206 3356
SISHWOUT         163 2076 2079 2082 2084 2572 2583 2593 2602 2611 2618 2627 2637 2988 2991 2993 2996 3377
SISREP           224  226 2211 2827 2848 2361 2874 2887 2900 2915 2972 3021 3033 3270
SISSW           2210 2211 2396 2781 3269 3270 3935
SMAX             151  883* 917*4157
SMIN             151  884* 918*4158
SNUM             151  887* 921*4159
SOC             2063 2068 2069 2071*2072 2086 2094 2103 2204 2214 2216 2218 2220 2224
START            580  608  646  714 3414 3423 3440 3442 3446 3498 3500 3504 3555 3557 3561 3616 3618 3622 3670 3677
                3687 3697 3701 3711 3751 3755 3765 3866 3870 3878
STARTER          157 4001 4005 4006 4007 4008
STARTTIME        405  788 1140 1167 1241 1243 1245 1288 1317 1318 1320 1351 1401 1403 1426 1428 1444 1499 1554 1607
                1659 1676 1701 1746 1809
STORAGEALLOC     449 4032
SUBSCLRDOWN     2106 2205 2312
SUC             1105 1120 1466 1468
SUM             1856 1861*1876
SUSENG           151  882  883* 884* 916  917* 918*
SUSPLO           153  881  889  904  915  923  957
SUSPLOQ          155  881  882  905  915  916  958 4050
SUSPLO1          876 1085 1165
SUSPLO2          909 1025
SUSPMAP          154  669  892  900  925  938  952  980
SUSPOINT         640  650  662  664 1002 1213 1652
SUSPROC          876  877  892  895*1079 1127 1132
SUSTART          787  880  885  886  914  919  920
SUSWAIT          152  885* 919*4160
SW              3696 3716 3865 3885
SYSCHED          976 1214 1651
T                171  173* 175  181* 184  190* 192  206  207* 209  305  312  656  895  900  946  952 1007 1008 1092
                1112 1115 1116 1166 1244 1272 1287 1305 1337 1358 1375 1385 1451 1476 1488 1507 1523 1533 1562 1578
                1588 1614 1684 1743 3932 3936 3942 4002 4014 4046 4047
TASK             424  969  972 1060 1103*1105*1121 1126 1131 1189 1381 1465 1519 1574 1629 1631 1632 1641 1680 1692
```

```
IDENTIFIER      LINE

                 1696 2402 2409 2410 3464 3515 3578 3586 3636 3644 3645 3716 3771 3809 3847 3882 3885
TASKBLOCK         415  794  863  864  867 1076 1103 1105 1121 1295 1297 1368 1369 1460 1461 1466 1468 1515 1516 1570
                 1571 4083
TASKFOUND         794  842  969  972 1296 1298 1369 1370 1371 1381*1389 1461 1462 1464 1465*1466 1468*1471 1473 1516
                 1517 1518 1519*1543 1571 1572 1573 1574*3464 3515 3578 3586 3636 3644 3645 3716 3771 3809 3847 3882
                 3885
TASKHANDED        794 1056 1060 1103 1105 1108 1126 1131 1204 1645 1658 1693
TI               3488 3515
TIME              654  684  693  703  801  808  830  835  851  856  880  885  896  914  919  932  947  992  997 1005
                 1030 1035 1113 1140 1153 1167 1168 1173 1222 1241 1243 1245 1249 1264 1274 1288 1317 1318 1320 1340
                 1351 1361 1401 1403 1426 1428 1444 1454 1499 1510 1554 1565 1607 1617 1659 1676 1687 1701 1746 1754
                 1757 1806 1807 1809 2255 2265 2322 2485 2497 2573 2587 2630 2672 2826 2846 3118 3133 3206 3286 3356
TIMEOUT          2208 2301 2302 3268 3294 3295 3304 3305 3313 3314 3322 3323 3331 3332 3340 3341 3349 3350
TIT               301  586  590  591 1623 1629 1630 1631 3964 3965*3970 3974*3980 4005 4006 4007 4008 4035 4036 4037
                 4066 4067 4068 4069
TITLE            1852 1853*1873
TI1              3696 3717
TI2              3696 3721
TI3              3696 3725
TI4              3696 3729
TI5              3696 3733
TI6              3696 3737
TOTALCCTS         133  138  143
TR               3411
TSK              2780 2783 2784
TSKIN             863  868
T1               2399 2432 2778 2792 3107 3146
T10              2399 2624 2778 2883
T11              2778 2896
T12              2778 2909
T13              2778 2932
T14              2779 2952
T15              2779 2965
T16              2779 2968
T17              2779 2985
T18              2779 3007
T19              2779 3017
T2               2399 2435 2778 2795 3107 3154
T20              2779 3020
T21              2779 3030
T22              2779 3056
T23              2779 3071
T3               2399 2442 2778 2798
T4               2399 2569 2778 2801
T5               2399 2580 2778 2809
T6               2399 2590 2778 2822
T7               2399 2599 2778 2842
T8               2399 2608 2778 2857
T9               2399 2616 2778 2870
U                2780
UNIFORM          2301 3199 3214 3294 3304 3313 3322 3331 3340 3349 3366 3369
UNSUC            1421 1494
UNSUSPENDED       790  939  940  941  943 1215 1220 1223 1226 1227 1231 1232 1248 1249*1250 1257 1263 1264*1265 1272*
                 1285*1292 1299 1300 1322 1332 1337*1339
UPDATE           1858 2573 2587 2630 2826 2846 3118 3133 3206
V                 171  173  176  181* 185  206  207* 210
W                 299  364 1569
WAIT              666  692  696  802  831  852  881  915  993 1031 1169
WITHFBLCALL      1911
WRITE             206  656  895  946 1007 1008 1092 1112 1116 1272 1305 1337 1358 1375 1385 1451 1476 1488 1507 1523
                 1533 1562 1578 1588 1614 1684
X                 298  579  580  582  584  586  600  601 1053 1056*1063 1064 1076 1105*1106 1858*1861 1862 1864*1866*
                 2143 2404 2438 2439 2440 2445 2446 2447 2484 2485 2496 2497 2501 2509 2513 2521 2525 2533 2537 2545
                 2549
Y                 786 1053 1061*1063 1077 1100 1104 1105 1106 1108 1119 1120*1121 1208*1209 1650 1654 2405 2572 2575
                 2576 2578 2583 2584 2586 2588 2593 2594 2596 2597 2602 2603 2605 2606 2611 2612 2613 2614 2618 2619
                 2621 2622 2627 2628 2629 2631 2637 2638 2639 2641 2782
Z                 786 1652 1653 1655


TOTAL NUMBER OF DIFFERENTLY SPELLED IDENTIFIERS AVAILABLE IN THIS PROGRAM :   463

NO DIAGNOSTICS FOR THIS COMPILATION.
```

APPENDIX C

A SAMPLE OF THE TRACE PROGRAM OUPUT



```
               PA SIMULATION RUN TRACE
              ==========================

SIMULATED TIME =      100000
CLOCKINT AT TIME=           0
INTRIP ENTERED NOW
INT. TRIPLICATES ENTERED FOR CLOCKINT                   0
INTRIP HAS SERVED CLOCKINT
CPU INTERRUPTED IS NO.          2
AND ITS INTERRUPT NO.          1
AT RUNTIME =            0

PROCESS INTERED IN SUSPMAP IS    BACKGROUND    117
AT RUNTIME =          132

SUSPENDED PROCESSES IN SUSPMAP ARE:-
BACKGROUND          117

PROCESS HANDEL TASK IS    INTIM     16
AT RUNTIME =          267
NO. OF TASKS IN INPUTQ =              1

INPUTQ TASKS PRIORITIES ARE:-
   0
PROCESS INTERED IN SUSPMAP IS    INTIM     16
AT RUNTIME =          305

SUSPENDED PROCESSES IN SUSPMAP ARE:-
INTIM          16
BACKGROUND          117

CHOSEN PROCESS IS     INTIM     16
AT TIME =          327
TO RUN ON CPU NO.              2

SUSPENDED PROCESSES NOW ARE:-
BACKGROUND          117

LCPU UPDATED ,NEW IS NO.            1
SELECTED PROCESS STATE IS   SUSP(UNBLOCKED)AT RUNTIME =         359
AND ITS PI=          16

PROCESS SELECTED TO RUN IS   INTIM      16
TO RUN ON CPU NO.=          2
NO. OF TASKS IN INPUT QUEUE =             0
AT RUNTIME =          575

INTIM CORRECTLY ENTERED
VALUE OF VARIABLE X NOW IS          1
PI STORED IN PPTABLE =          0

CALL TO HAND BY PROCESS   INTIM      16
ON CPU NO.=          2
AT RUNTIME =          648
OGTI          1
AND CALL INDEX=          2
```

```
PROCESS HANDED TASK IS    RASH     41
AT RUNTIME =          663
NO. OF TASKS IN INPUTQ =              1
HANDING PROCESS IS    INTIM    16

INPUTQ TASKS PRIORITIES ARE:-
    5
PROCESS INTERED IN SUSPMAP IS   RASH     41
AT RUNTIME =          713

SUSPENDED PROCESSES IN SUSPMAP ARE:-
RASH        41
BACKGROUND       117

SUSPQINT TRIGGERED AT TIME=          713
BY PROCESS ALLOCATOR ON CPU NO.               2
BECAUSE HIGHEST PRIO. SUSP PROCESS IS     RASH     41
AND LCPU CURP IS    BACKGROUND   116
AND LCPU IS NO.              1

INTRIP ENTERED NOW
INTRIP ENTERED BECAUSE SUSPQINT TRIGGERED
CPU INTERRUPTED IS NO.           1
AND ITS INTERRUPT NO.           1
AT RUNTIME =          724

VALUE OF VARIABLE X NOW IS           2
PI STORED IN PPTABLE =           41

CALL TO HAND BY PROCESS   INTIM     16
ON CPU NO.=           2
AT RUNTIME =          819
OGTI          1
AND CALL INDEX=           2

PROCESS HANDED TASK IS    RASH     42
AT RUNTIME =          835
NO. OF TASKS IN INPUTQ =              1
HANDING PROCESS IS    INTIM    16

INPUTQ TASKS PRIORITIES ARE:-
    5
PROCESS INTERED IN SUSPMAP IS   BACKGROUND    116
AT RUNTIME =          856

SUSPENDED PROCESSES IN SUSPMAP ARE:-
RASH        41
BACKGROUND       116
BACKGROUND       117

PROCESS INTERED IN SUSPMAP IS   RASH     42
AT RUNTIME =          884

SUSPENDED PROCESSES IN SUSPMAP ARE:-
RASH        41
RASH        42
BACKGROUND       116
BACKGROUND       117




SUSPQINT TRIGGERED AT TIME=          903
BY PROCESS ALLOCATOR ON CPU NO.               2
BECAUSE HIGHEST PRIO. SUSP PROCESS IS     RASH     41
AND LCPU CURP IS    BACKGROUND   116
AND LCPU IS NO.              1

CHOSEN PROCESS IS    RASH     41
AT TIME =          903
TO RUN ON CPU NO.           1

SUSPENDED PROCESSES NOW ARE:-
RASH        42
BACKGROUND       116
BACKGROUND       117
```

APPENDIX D

A SAMPLE OF OUTPUT REPORTS OF FBLOCK(P) EXPERIMENT

```
*****************************************************
***  APROCESS ALLOCATOR SIMULATOR OF MK2 BL SYSTEM ***
***  ========================================== ***
***     A.M.SALIH PLYMOUTH POLYTECHNIC JUNE,1979   ***
***       PARAMETERS OF THIS SIMULATOR ARE:=       ***
***         SIMULATION LANGUAGE: SIMULA            ***
***        NUMBER OF CPUS = 2                      ***
***  EXPERIMENTAL EVALUATION OF FBLOCK CALL TO PA ***

***           SIMULATED TIME=  7200000.00          ***
***                                                ***
*****************************************************



SOFTWARE PROCESS AND THEIR PROCESS INDICES
==========================================
INDEX OF INTIM PROCESS IS          16
INDEX OF STORAGE ALLOCATOR PROCESS IS          14
INDEX OF NOFBLCALL  1IS           41
INDEX OF NOFBLCALL  2IS           42
INDEX OF NOFBLCALL  3IS           43


**STATISTICS FOR PROCESS NO          14
========================================
MAX NO. OF TASKS IN I/P Q =          1
MIN NO. OF TASKS IN I/P Q =          0
MAX PROC ALLOCS WAITING PROLO=            0
MIN PROCS ALLOCS WAITING FOR PROLO=            0
TIMES PROLO INITIATED =       5760
TIME PROC ALLOCS WAITING PROLO=           0.00
TIMES PROCESS INITIATED =       1440

**STATISTICS FOR PROCESS NO          16
========================================
MAX NO. OF TASKS IN I/P Q =          1
MIN NO. OF TASKS IN I/P Q =          0
MAX PROC ALLOCS WAITING PROLO=            0
MIN PROCS ALLOCS WAITING FOR PROLO=            0
TIMES PROLO INITIATED =       2160
TIME PROC ALLOCS WAITING PROLO=           0.00
TIMES PROCESS INITIATED =        720

**STATISTICS FOR PROCESS NO          41
========================================
MAX NO. OF TASKS IN I/P Q =          2
MIN NO. OF TASKS IN I/P Q =          0
MAX PROC ALLOCS WAITING PROLO=            0
MIN PROCS ALLOCS WAITING FOR PROLO=            0
TIMES PROLO INITIATED =       4321
TIME PROC ALLOCS WAITING PROLO=           0.00
TIMES PROCESS INITIATED =        720

**STATISTICS FOR PROCESS NO          42
========================================
MAX NO. OF TASKS IN I/P Q =          2
MIN NO. OF TASKS IN I/P Q =          0
MAX PROC ALLOCS WAITING PROLO=            0
MIN PROCS ALLOCS WAITING FOR PROLO=            0
TIMES PROLO INITIATED =       4321
TIME PROC ALLOCS WAITING PROLO=           0.00
TIMES PROCESS INITIATED =        720

**STATISTICS FOR PROCESS NO          43
========================================
MAX NO. OF TASKS IN I/P Q =          2



MIN NO. OF TASKS IN I/P Q =          0
MIN NO. OF TASKS IN I/P Q =           0
MAX PROC ALLOCS WAITING PROLO=            0
MIN PROCS ALLOCS WAITING FOR PROLO=             0
TIMES PROLO INITIATED =        4321
TIME PROC ALLOCS WAITING PROLO=           0.00
TIMES PROCESS INITIATED =         720


**STATISTICS FOR THE PROCESS ALLOCATOR OF CPU NO.            1
==================================================
TIMES PROCESS ALLOCATOR CALLED=       10153
TIMES PROCESS ALLOCATOR INTERRUPTED=           721
THIS PA OVERHEAD=  2186061.00
THIS PA FETCH AND BLOCK OVERHEAD=   589584.69

**STATISTICS FOR THE PROCESS ALLOCATOR OF CPU NO.            2
==================================================
TIMES PROCESS ALLOCATOR CALLED=        5834
TIMES PROCESS ALLOCATOR INTERRUPTED=          2159
THIS PA OVERHEAD=  1819039.00
THIS PA FETCH AND BLOCK OVERHEAD=   370890.37


**STATISTICS OF FREELO **
==========================
MAX NO. OF PROCESS ALLOCATORS WAITING FOR FREELO=          1
MIN NO. OF PROCESS ALLOCATORS WAITING FOR FREELO=          0
TIMES FREELO ENGAGED =      14547
TIME SPENT BY PROCESS ALLOCS WAITING FOR FREELO=          568

**STATISTICS OF SUSPLO**
==========================
MAX PROC ALLOCS WAITING SUSPLO=          0
MIN PROC ALLOCS WAITING SUSPLO=          0
TIMES SUSPLO ENGAGED =       14544
TIME SPENT BY PROC ALLOCS WAITING SUSPLO=          0

**STATISTICS OF INTLO**
==========================
MAX PROC ALLOCS WAITING INTLO=          1
MIN PROC ALLOCS WAITING INTLO=          0
TIMES INTLO ENGAGED =      12312
TIME SPENT BY PROC ALLOCS WAITING INTLO=       12960

**CPUS STATISTICS**
=========================

FOR CPU NO.         1
========================
TIME ON BACKGROUND PROCESSES =  2517245.00
PERCENTAGE CPU OCCUPANCY OF PROCESSES OTHER THAN BACKGROUND=          34.68

FOR CPU NO.         2
========================



TIME ON BACKGROUND PROCESSES =  4129133.00
PERCENTAGE CPU OCCUPANCY OF PROCESSES OTHER THAN BACKGROUND=          17.39


   RESULTS WITHOUT FBLOCK(P) CALL TO PA
      ==============================
FOR A MULTI=PROCESSOR SYSTEM WITH  2 CPUS AND 3NOFBLCALL PROCESSES :=
                   TIMES LOOP TRAVERSED
NOFBLCALL  1          720

NOFBLCALL  2          720

NOFBLCALL  3          720
```

```
 **  ==============================================  ***
 **     A.M.SALIH PLYMOUTH POLYTECHNIC JUNE,1979     ***
 **      PARAMETERS OF THIS SIMULATOR ARE:-          ***
 **        SIMULATION LANGUAGE; SIMULA               ***
 **       NUMBER OF CPUS =  2                        ***
 **   EXPERIMENTAL EVALUATION OF FBLOCK CALL TO PA   ***

 **          SIMULATED TIME=  7200000.00             ***
 **                                                  ***
 ****************************************************



SOFTWARE PROCESS AND THEIR PROCESS INDICES
------------------------------------------
INDEX OF INTIM PROCESS IS          16
INDEX OF STORAGE ALLOCATOR PROCESS IS          14
.NDEX OF WITHFBLCALL 1IS          41
.NDEX OF WITHFBLCALL 2IS          42
.NDEX OF WITHFBLCALL 3IS          43


 *STATISTICS FOR PROCESS NO          14
 ---------------------------------------
MAX NO. OF TASKS IN I/P Q =          1
MIN NO. OF TASKS IN I/P Q =          0
 AX PROC ALLOCS WAITING PROLO=          0
 IN PROCS ALLOCS WAITING FOR PROLO=          0
 IMES PROLO INITIATED =          5760
 IME PROC ALLOCS WAITING PROLO=          0.00
 IMES PROCESS INITIATED =          1440

 *STATISTICS FOR PROCESS NO          16
 ---------------------------------------
 AX NO. OF TASKS IN I/P Q =          1
 IN NO. OF TASKS IN I/P Q =          0
 AX PROC ALLOCS WAITING PROLO=          0
 IN PROCS ALLOCS WAITING FOR PROLO=          0
 IMES PROLO INITIATED =          2160
 IME PROC ALLOCS WAITING PROLO=          0.00
 IMES PROCESS INITIATED =          720

 *STATISTICS FOR PROCESS NO          41
 ---------------------------------------
 AX NO. OF TASKS IN I/P Q =          2
 IN NO. OF TASKS IN I/P Q =          0
 AX PROC ALLOCS WAITING PROLO=          0
 IN PROCS ALLOCS WAITING FOR PROLO=          0
 IMES PROLO INITIATED =          3601
 IME PROC ALLOCS WAITING PROLO=          0.00
 IMES PROCESS INITIATED =          720

 *STATISTICS FOR PROCESS NO          42
 ---------------------------------------
 AX NO. OF TASKS IN I/P Q =          2
 IN NO. OF TASKS IN I/P Q =          0
 AX PROC ALLOCS WAITING PROLO=          0
 IN PROCS ALLOCS WAITING FOR PROLO=          0
 IMES PROLO INITIATED =          3601
 IME PROC ALLOCS WAITING PROLO=          0.00
 IMES PROCESS INITIATED =          720

 *STATISTICS FOR PROCESS NO          43
 ---------------------------------------
 AX NO. OF TASKS IN I/P Q =          2

 IN NO. OF TASKS IN I/P Q =          0
 AX PROC ALLOCS WAITING PROLO=          0
 IN PROCS ALLOCS WAITING FOR PROLO=          0
 IMES PROLO INITIATED =          3601
 IME PROC ALLOCS WAITING PROLO=          0.00
 IMES PROCESS INITIATED =          720


 *STATISTICS FOR THE PROCESS ALLOCATOR OF CPU NO.          1
 ----------------------------------------------------------
 IMES PROCESS ALLOCATOR CALLED=          8713
 IMES PROCESS ALLOCATOR INTERRUPTED=          721
 HIS PA OVERHEAD=  2050002.00
 HIS PA FBLOCK CALL OVERHEAD=   452905.19

 *STATISTICS FOR THE PROCESS ALLOCATOR OF CPU NO.          2
 ----------------------------------------------------------
 IMES PROCESS ALLOCATOR CALLED=          5114
 IMES PROCESS ALLOCATOR INTERRUPTED=          2158
 HIS PA OVERHEAD=  1757227.00
 HIS PA FBLOCK CALL OVERHEAD=   309479.87

*STATISTICS OF FREELO **
---------------------------
AX NO. OF PROCESS ALLOCATORS WAITING FOR FREELO=          0
IN NO. OF PROCESS ALLOCATORS WAITING FOR FREELO=          0
IMES FREELO ENGAGED =          14547
IME SPENT BY PROCESS ALLOCS WAITING FOR FREELO=          0


*STATISTICS OF SUSPLO**
---------------------------
AX PROC ALLOCS WAITING SUSPLO=          0
IN PROC ALLOCS WAITING SUSPLO=          0
IMES SUSPLO ENGAGED =          14542
IME SPENT BY PROC ALLOCS WAITING SUSPLO=          0


*STATISTICS OF INTLO**
---------------------------
AX PROC ALLOCS WAITING INTLO=          0
IN PROC ALLOCS WAITING INTLO=          0
IMES INTLO ENGAGED =          12310
IME SPENT BY PROC ALLOCS WAITING INTLO=          0


*CPUS STATISTICS**
-------------------------

FOR CPU NO.          1
---------------------
TIME ON BACKGROUND PROCESSES =  2652965.00
PERCENTAGE CPU OCCUPANCY OF PROCESSES OTHER THAN BACKGROUND=          34.68


FOR CPU NO.          2
---------------------
IME ON BACKGROUND PROCESSES =  4190591.00
ERCENTAGE CPU OCCUPANCY OF PROCESSES OTHER THAN BACKGROUND=          17.39


      RUN RESULTS WITH FBLOCK(P) CALL
      ===============================
 OR A MULTI-PROCESSOR SYSTEM WITH  2 CPUS AND 3WITHFBLCALL PROCESSES :-
                TIMES LOOP TRAVERSED
WITHFBLCALL 1          720

ITHFBLCALL 2          720

ITHFBLCALL 3          720
```

$$= \pm \sqrt{\left(\frac{\Sigma(xy)}{\Sigma x^2}\right)^2 \frac{\Sigma x^2}{\Sigma y^2}} \qquad = \pm\sqrt{\frac{(\Sigma(xy))^2}{(\Sigma x^2)(\Sigma y^2)}}$$

$$\text{or} \quad r \quad = \quad \frac{\Sigma(xy)}{\sqrt{(\Sigma x^2)(\Sigma y^2)}} \qquad \text{..........(7)}$$

when Y_{est} increases as x increases, then r is positive. This is known as the product-moment formula for correlation coefficient.

REFERENCES

ANDE 72 ANDERSON H.A. and SARGANT R.G.
"A Statistical Evaluation of the Scheduler of an Experimental Interactive Computing System"
In 'Statistical Computer Performance Evaluation'
Academic Press, 1972 pp 73-98

ANDE 72 ANDERSON R.R, HAYES J. and SHERMAN D.N.
"Simulated Performance of a Ring-Switched Data Network"
IEEE Trans on Comm. Vol.COM-20 No.3, June, 1972
pp 576-591

AMOS 76 AMOS E.and JOEL J.
"Electronic Switching : Central Office Systems of the World"
IEEE Press, 1976

ARGO 79 ARGOE D.T. and WILBER J.A.
"System Recovery in the 2B Processer"
GTE Automatic Electric Journal, July, 1979
pp 141-148

BABC 78 BABCICKY K.
"SIMULA Programming Efficiency Considerations"
Association of SIMULA Users Workshop
'Second Steps in SIMULA' University of Bradford
18th December, 1978

BAER 68 BAER J.L. and RUSSELL E.C.
"Modelling and Scheduling of Computer Programs for Parallel Processing Systems"
Proc. 2nd Conf. on Applications of Simulation
Dec. 1968 N.Y. pp 278-281

BAKE 75 BAKER F.T.
"Notes on Structured Programming"
In 'Structured Programming' Academic Press, 1972

BARK 69 BARKER P.E. and WATSON H.K.
"Calibrating the Simulation Model of the IBM System /360 Time Sharing System"
Proc. Conf. on Applications of Simulation, Los Angles
Dec 1969 pp 130-137

BARR 71 BARRON D.W.
"Computer.Operating Systems"
Champman and Hall Ltd, 1971

BAUE 74 BAUER M.J.
"A Simulation Approach to the Design of Dynamic Feedback Scheduling Algorithms for Time-Shared Computer Systems"
Proc. ACM-SIGSIM Symp. 4-6 June, 1974 pp 165-173

BEAR 79 BEAR D.
"Outline of a Theoretical Traffic Model of the Mark II BL System"
The General Electric Co. Ltd., Hirst Research Centre
Memorandum No. TRL/767, February, 1979

BEAS 73 BEASTALL H. and POVEY J.A.
"Teletraffic Studies of TXE4"
POEEJ, Vol. 65, Jan. 1973 pp 251-255

BECK 73 BECK D. S.
"The Plessey System 250-Facilities for Simulation of Telecommunication Equipment"
Proc. IEE Conf. on Software Engineering for Telecommunications Switching Systems, 2-5, April, 1973 IEE Pub. No. 97 pp 31-38

BEIL 77 BEILNER H. and GELENBE E. (Editors)
"An Approach to the Straightforward Production of Computer System Simulators"
In 'Modelling and Performance Evaluation of Computer Systems'
North-Holland Pub. Co. 1977

BELL 72 BELL T.E.
"Objectives and Problems in Simulating Computers"
Proc. AFIPS-FJCC Conf. 1972, pp 287-297

BELS 78 BELSNES D. and BRINGSRUD K.A.
"X.25 DTE Implemented in SIMULA"
EUROCOMP 78, London, May, 1978

BERN 68 BERNARD R. and GRANDJEAN C.
"Application of GPSS to the Simulation of Telephone Systems"
Proc. 2nd Conf. on Applications of Simulation 2-4 Dec., 1968 N.Y. pp 194-197

BIRT 73 BIRTWHISTLE G.M., DAHL O.J., MYHRHANG B. & NYGAARD K.
"SIMULA Begin"
Studentlitteratu, Sweden, 1973

BIRT 79 BIRTWHISTLE G.M.
"DEMOS - Discrete Event Modelling on SIMULA"
Macmillan, 1979

BLUN 67 BLUNDEN G.P.
"Implicit Interaction in Process Models"
Proc. IFIP CONF. on Simulation Programming Languages, May, 1967 Oslo, Norway pp 283-291

BLUN 74 BLUNK R.G.
"Simulation of an OS/360 MVT Environment with GPSS"
Proc. ACM-SIGSIM Symp. 4-6, June, 1974 pp 53-61

BOUT 78 BOUTE R.T.
"Logical Models for Computer Control of Telephone Exchanges"
3rd Inter. Conf. on Software Eng. for TElecomm. Switching Systems 27-29, June, 1978 IEE Pub. No. 164 pp 18-24

BRAE 79 BRAEK R.
"Functional Specification and Description Languages as a Practical Tool for Improved System Quality"
ITU Telecom. 79 Forum, Geneva, Spet. 1979 pp 1.3.1.1-1.3.1.9

BRIL 77 BRILEY B.E. and TOY W.N.
"Telecommunications Processors"
Proc. IEEE Vol 65, No.9 Sept. 1977 pp 1305-1313

BROW 78 BROWN J.G.
"Developing an Electronic Local Exchange Register Translator for the British Post Office"
Electronics and Power June 1978 pp 448-450

BRUC 79 BRUCE R.A., GILOTH P.K. & SPIEGEL E.H.
"No. 4 ESS: A Continuing Evolution"
Bell Labs. Records, Vol57, No.6 1979 pp 155-162

BUCH 69 BUCHOLZ W.
"A Synthetic Job for Measuring System Performance"
IBM System Journal, Vol 8, No.4, 1969 pp 309-318

BUSS 68 BUSSARD D.L.
"Communication Network Design Using Message Flow Simulation"
Proc. 2nd Conf. on Applications of Simulation N.Y. Dec 1968
pp 198-201

BUXT 62 BUXTON J.N. and LASKI J.G.
"Control and Simulation Language"
Computer Journal Vol.5 April, 1962 - Jan. 1963 pp 194-199

BUXT 66 BUXTON J.N.
"Writing Simulations in CSL"
Computer Journal, 1966 Vol 9, pp 137-143

CALI 67 CALINGERT P.
"System Performance Evaluation: Survey & Appraisal"
Comm. of ACM, Vol.10, No.1, Jan.1967 pp 12-18

CASY 77 CASY L.M.
"Computer Structures for Distributed Systems"
Ph.D. Thesis, University of Edinburgh 1977

CCIT 77 CCITT (Consultative Committee for Internation Telephony & Telegraphy)
"Functional Specification and Description Language (SDL)"
CCITT Recommendations Z-101-Z103, Orange Book Vol. VI.4
ITU Geneva, 1977

CCIT 77A Consultative Committee for International Telephony & Telegraphy
"Programming Languages for Stored-Programme Control Exchanges"
CCITT Organge Book Vol VI.4 Geneva, 1977

CHAN 78 CHANDY K.M. & YEH R.T. (Editors)
"The Regenerative Method for Simulation Analysis"
in 'Current Trends in Programming Methodology Vol. III : Software Modelling'
Prentice-Hall Inc. 1978

CHAR 78 CHARLES E.H. and CHARLES P.P.
"ASSIST -V: An Evironment Simulator for IBM/360
Systems Software Development"
IEEE Trans. on Software Engineering Vol SE-4 No.6,
Nov. 1978 pp 526-530

CHEN 69 CHENG P.S.
"Trace Driven System Modelling"
IBM System Journal, Vol.8, No.4, 1969
pp 280-289

CIES 79 CIESLAK T.J. and HICKSON P.J.
"Analysis and Simulation of No.4 ESS Network Performance"
IEEE Trans. on Comm., Vol COM-27, No.1, Jan. 1979 pp.1-9

COHE 61 COHEN K.J. and CYERT R.M.
"Computer Models in Dynamic Economics"
The Quarterly Journal of Economics, LXXV, Feb. 1961
pp 112-127

COHE 68 COHEN L.J.
"S3, The System and Software Simulator"
Proc. 2nd Conf on Applications of Simulation,
2-4 Dec. 1968 NY pp 282-285

COLE 64 COLE A.C., THOMSON W.E. and WILLIAMS J.W.J.
"Digital Computer Simulation of Switching Systems
Using the Algol Automatic Programming Language"
4th International Teletraffic Congress, London
1964 Session 5.

CONW 59 CONWAY R.W., JOHNSON B.M. and MAXWELL W. L.
"Some Problems of Digital Systems Simulation"
Management Science, Vol.6, 1959 pp 92-110

CRAN 74 CRANE M.A. and IGLEHART D.L.
"Statistical Analysis of Discrete-Event Simulations"
Proc. Winter Simulation Conf. 1974 Washington D.C.
pp513-521

CSL 69 "1900 CSL Users Manual"
ICL 1900 Series, Technical Pub. 3386, July, 1969

CULL 79 CULLEN A.
"The TRL Multimicroprocessor Operating System : First
Design"
GEC Hirst Research Centre Memo No. TRL/814 Nov. 1979

CUNN 76 CUNNINGHAM R.J. PARR F.N. and SIM R.J.W.
"An Introduction to Combined Continuous System and
Discrete Event Process Simulation in SIMULA"
Pub. No. 76/11, Dept. of Computing and Control,
Imperial College, London, April, 1976

CUNN 81 CUNNINGHAM R. J. and SALIH A.M.
"The Use of Verification-Oriented Software Specification
in Telecommunication Engineering"
To be presented at the 4th International Conference
on Software Engineering for Telecommunications Switching
Systems, Warwick University, July, 1981, England.

DAHL 66 DAHL O.J. and NYGAARD K.
"SIMULA - An ALGOL-based Simulation Language"
Comm. of ACM Vol 9, No.9 Sept 1966
pp 671-678

DAHL 67 DAHL O.J. and NYGAARD K.
"Class and Sub-Class Declarations"
Proc. IFIP Conf. on Simulation Programming Languages
May, 1967 Oslo Norway, pp 158-174

DAHL 68 DAHL O.J.
"Discrete Event Simulation Languages"
In 'Programming Languages' NATO Advanced Study
Institute; F. Gennys (ED) Academic Press, 1968
pp 349-395

DAHL 70 DAHL O.J., MYHRHAUG B. and NYGAARD K.
"Common Base Language"
Norwegian Computing Centre, Pub. No. S-22, May, 1970

DAVI 74 DAVIS M. and MILLER G.E.
"What Price Virtual Memory"
Proc. ACM-SIGSIM 4-6 June, 1974 pp 85-93

DENN 68 DENNING P.J.
"The Working Set Model for Program Behaviour"
Communications of ACM Vol 11, No.5, May 1968
pp 323-333

DENN 70 DENNING P.J.
"Virtual Memory"
Computing Surveys, Vol.2, No.3, Sept. 1970
pp 153-189

DENT 78A DENT G.R.
"Call Processing Subsystem - General Description"
Document 1 ADA 00003 AAD-BA. The General Electric
Co. Ltd., of England. 1978

DENT 78B DENT G.R.
"CALL Processing Subsystem - Design Text"
Document 1 ADA 00003 AAD-CA. The General Electric
Co. Ltd., of England 1978

DENT 78C DENT G.R.
"Calling Processing Subsystem - Interface Specification"
Document 1 ADA 00003 AAD-BF. The General Electric Co.
Ltd. of England 1978

DIET 75 DIETRICH G.
"Traffic Model of Common Control Switching Systems"
Electrical Communications (GB) Vol. 50 No.1, 1975
pp 28-34

DIJK 68 DIJKSTRA E.W.
"Co-operating Sequential Processes"
in F. Genunys (Ed.) : Programming Languages
Academic Press, London and N.Y. 1968

DIJK 72 DIJKSTRA E. W.
"Notes on Structured Programming" in 'Structured Programming', Academic Press. 1972

DGWE 73 DGWELL D.
"Processor Environment Simulation"
Proc. IEE Conf. on Software Eng. for Telecomm. Switching Systems 2-5 April, 1973 IEE Pub. No. 97

EBNE 79 EBNER G.C. & TOMKO L.A.
"CCIS : Signalling System for the Stored Program Controlled Network"
Bell Lab. Records Vol.57, No.2 Feb. 1979

EDGE 72 EDGE G.
"System 250 and Diagnostics"
IERE Conf. Proceedings No. 25 Oct 1972 pp 201-212

ELLI 78 ELLISON D.
"Need a Language - Where Shall I Turn?"
In SIMULATION Newsletter, Summ. 78 Centre in Simulation University of Lancaster, Lancaster U.K.

EMSH 70 EMSHOFF J.R. & SISSON R.L.
"Design and Use of Computer Simulation Models"
The Macmillan Co. 1970

FAGA FAGAN B.M. & KLEINE H.
"A Simulation Model of a Message Switching System"
Proc. ACM-SIGSIM, 3-6, June, 1974 pp 62-73

FISH 67 FISHMAN G.S. & KIVIAT P.J.
"Digital Computer Simulation : Statistical Considerations"
The Rand Corp. RM-5287-PR, 1967 Santa Monica, Calif.

FISH 68 FISHMAN G.S. & KIVIAT P.J.
"The Statistics of Discrete Event Simulation"
Simulation, April, 1968 pp 185-195

FISH 73 FISHMAN G.S.
"Concepts & Methods in Discrete Event Digital Simulation"
John Wiley & Sons Ltd., 1973

FLOO 75 FLOOD J.E. (Ed)
"Telecommunications Networks"
Peter Peregrinus Ltd., 1975

FLOO 76 FLOOD J.E.
"Alexander Graham Bell & the Invention of the Telephone"
IEE Proc. Vol. 123, No.12, Dec. 1976 pp 1387-1388

FONT 71 FONTAINE B.
"Real-Time Environment Simulation"
Electrical Communications (GB) Vol.46 No.3, pp188-190,1971

FRAN 73 FRANKLIN M. & LOONEY M.
"Simulation of a Computer System with Single and Dual Density Disks"
Proc. ACM Symp. on Simulation of Computer Systems, 19-20 June 1973 pp 259-267

FRAN 74 FRANTA W.R. and HOULE P.A.
"Comments on Models of Multiprocessor Multimemory Bank Computer Systems"
Proc. Winter Simulation Conf. Jan. 1974 Washington
pp87-97

FRAN 77 FRANTA W.R.
"The Process View of Simulation"
Elseveir North-Holland Inc., 1977

FREI 72 FRIEBERGER E. W. - Editor
"Statistical Computer Performance Evaluation"
Academic Press N.Y and London, 1972

GALE 75 GALE N .
"Application of State Transition Diagrams to System Independent Specification of Telephone Facilities and Systems"
ATR (Australia) Vol 9, No.2, 1975 pp 3-12

GEC 75A "GEC Mark II BL Communications Control Processor System"
GEC Document 950-0001-001 Sept, 1975 Coventry, U.K.

GEC 75B "GEC Mark II BL Interrupt and Timing Process Definition Text"
GEC Document 803-9094-103, Oct, 1975 Coventry U.K.

GEC 76A "POPUS - Process Allocator General Description"
GEC Document 1SAA-00-254-AAS/BA, 1976 Coventry, U.K.

GEC 76B "Process Allocator for Emulator"
GEC Document 803-9085-001, 1976, Coventry, U.K.

GEC 77 "Storage Allocator Definition Text"
GEC Document 1SAA 00012 AAP/BA, 1977 Coventry, U.K.

GEC 78 "GEC Mark II BL Thrash and Crash Processes Definition Text"
GEC Document 1SAA 00058 AAN/BA, 1978 Coventry U.K.

GERR 74 GERRAND P.H.
"Use of Call-State Transition Diagrams in the Analysis of Telephone Call Failure Probabilities"
A.T.R. (Australia), Vol 8, No.1, 1974 pp 7-12

GERR 79 GERRAND P.H. and PARK J.L.
"Development of a Processor Monitoring Instrument for SPC Exchanges: Facilities and Application Areas"
Telecomm. Journal of Australia, Vol 29, No.2, 1979
pp 113-120

GIBS 70 GIBSON J.C.
"The GIBSON Mix"
IBM TR 00-2043, June, 1970

GORD 78 GORDON G.
"System Simulation"
Prentice-Hall Inc., Englewood, Cliffs, N.J. 1978

GRAN 64 GRANTGES R.F. and SINOWITZ N.R.
"NEASIM : A General Purpose Computer Simulation Program for Load-loss Analysis of Multistage Central Office Switching Networks"
B.S.T.J. May, 1964 Vol.43, No.3 pp 965-1004

GREE 80 GREEN H.
"Design Suggestions for Overload Control"
GEC Telecommunications Ltd., Memo No. GEC/DMNSC/DEV/DP (80) 15, October, 1980

GRUS 76 GRUSZECKI M. and CORNELIS F.
"Environment Simulator for Traffic and Processor Capacity Studies in SPC Switching Systems" Elec. Comm.(GB) Vol.51 No.2 pp 107-111 1976

GUIT 76 GUITONNEAU G.T., LEMMONIER J.L. and SOULET J.J.
"A Simulator for Debugging and Evaluating the E.11 System"
Proc. IEE Conf. on Softwate Engineerings for Telecomm. Switching Systems 18-20, Feb. 1976 IEE Pub. No. 135 pp 108-111.

HALT 77 HALTON D.
"System Reliability - S250 Multiprocessor"
IEE Vac. School on 'Stored Program Control of Telephone Switching Systems' University of Essex, 11-16, Sept, 1977 pp 13/1 -13/13

HANS 73 HANSEN, Per Brinch
"Operating Systems Principle"
Prentice-Hall Inc., Englewood, Cliffs, N.J, 1973

HANS 77 HANSEN Per Brinch
"The Architecture of Concurrent Programs"
Prentice-Hall Inc., Englewood, Cliffs, 1977

HARR 79 HARRIS L.R.F. and DAVIS E.
"System X and the Evolving U.K. Telecommunications Network"
P.O.E.E.J. Vol, 72, April, 1979 pp 2-8

HART 75 HARTLEY M.G. (ED)
"Digital Simulation Methods"
IEE Monograph No. 15, Peter Peregrinus Ltd, 1975

HILL 73A HILLS P.R.
"An Introduction to Simulation Using SIMULA"
Norwegian Computing Centre Pb. No. S55, Oslo, 1973

HILL 73B HILLS P.R.
"Simulation Model Structures"
SIMULA Newsletter, Vol. 1 No.2, Aug, 1973 pp 3-4

HILL 76 HILLS P.R. and BIRTWHISTLE G.
"SIMON 75 : Reference Manual"
Robin Hills (Consultants) Ltd., 48, High Street, Exeter U.K. 1976

HILL 76A HILLS M.T. and KANO S.
"Programming of Electronic Switching Systems".
Peter Peregrinus Ltd, 1976

HORN 72 HORN R.V.
"Validation"
In 'The Design of Computer Simulation Experiments'
Edited by T.H. Naylor, Duke University Press, Durham
1972 pp 232-251

HUES 67 HUESMAN L.R. and GOLDBERG R.P.
"Evaluating-Computer Systems Through Simulation"
Computer Journal, Vol.10 No.2, Aug. 1967
pp 150-155

HUTC 67 HUTCHINSON G.K.
"Some Problems in the Simulation of Multiprocessor
Computer Systems"
Proc. IFIP Conf. on Simulation Programming Languages,
Oslo, May, 1967 pp 305-318

HUTC 73 HUTCHINSON G.K.
"The Use of Micro-level Simulation in the Design of
a Computer Supervisory System"
Proc. ACM-SIGSIM Symp. on Simulation of Computer Systems
19-20 June, 1973 Gaithersburg, U.S.A. pp 242-254

IGLE 78 IGLEHART D.L. and SHELDER G.S.
"Regenerative Simulation of Response Times
in Networks of Queues"
Journal of ACM Vol25, No.3 July, 1978 pp 449-460

JAME 78 JAMES Y.
"Simulation of a Business Communication System Switching
Network"
Proc. Winter Simulation Conf., 4-6, Dec, 1978 pp 751-757

JONE 79 JONES W.G.T., KIRTLAND J.P. and MOORE W.B.
"Principles of System X"
P.O.E.E.J. Vol 72, July, 1979 pp 75-80

JOUB 78 JOUBERT T.and MILLET C.
"Modelling of Stored Program Controlled Switching
Systems: Simulation versus Analytical Methods"
International Conf. on Software Engineering for Telecommunication
Switching Systems 27-29 June, 1978 IEE Pub. No. 164 pp 25-30

KALV 72 KALVENES S.
"An Introductory Course in Statistics for Simulation"
Norwegian Computing Centre, Pub. No. S-49, 1972

KASP 74 KASP H.and MILLER D.S.
"Job and Job Stream Modelling in the TRW Timeshare
System Simulation"
Proc. ACM-SIGSIM 4-6, June, 1974 pp 94-121

KATZ 66 KATZ J.H.
"Simulation of a Multiprocessor Computer System"
Proc. AFIPS-SJCC Vol28, 1966 pp 127-139

KAWA 71 KAWASHIMA H., FUTAMI K. and KANO S.
"Functional Specification of Call Processing by State
Transition Diagrams"
IEEE Trans. on Communication, Vol. COM-19 No.5, Oct. 1971
pp 581-587

KAWA 79 KAWASHIMA H., ARIMA T. and SHIMIZU Y.
"Software Structure for Today and Tomorrow"
Proc. TElecomm. Forum, Geneva, Sept. 1979
pp 1.3.2.1-1.3.2.8

KAY 72 KAY I.M.
"An Over-the-Shoulder Look at Discrete Simulation Languages"
AFIPS Conf. Proc. Vol.40, 1972 pp 791-798

KELL 62 KELLEY D.H. and BUXTON J.N.
"Montecode - an Interpretive Program for Monte Carlo Simulations"
Computer Journal, Vol.5 1962 pp 88-93

KITZ 67 KITZ B.
"Problems Encountered in the Programming of Games and Simulations"
Proc. 'Digital Simulation in Operational Research' Conf.,Hamburg 6-10th Sept, 1965, English Univ. Press, 1967 pp 125-130

KIVI 67 KIVIAT P.J.
"Digital Computer Simulation : Modelling Conepts"
The Rand Corp. RM-5378-PR Santa Monica Cal. 1967

KIVI 69A KIVIAT P.J.
"Digital Computer Simulation : Computer Programming Languages"
The Rand Corp. Santa Monica, Cal., 1969

KIVI 69B KIVIAT P.J., VILLANEUVA R., and MARKOWITZ H.
"The SIMSCRIPT II Programming Language"
Prentice-Hall Inc. Englewood, Cliffs, N.J. 1969

KLEI 70 KLEINE H.
"A Survey of Users' Views of Discrete Simulation Languages"
Simulation Vol. 14, May, 1970 pp 225-229

KLEI 71 KLEINE H.
"A Second Survery of Users' Views of Discrete Simulation Languages"
Simulation August, 1971 pp 89-94

KLEI 74 KLEIJNEN J.P.C.
"Statistical Techniques in Simulation" Part 1.
Marcel Dekker Inc. N.Y. 1974

KNUT 64 KNUTH D.E. and MCNELEY J.L.
"SOL - A Symbolic Language for General-Purpose System Simulation"
IEEE Trans. on Electronic Computers, Aug.1964 pp 401-408

KOBA 78 KOBAYASHI H.
"Modelling and Analysis:An Introduction to System Performance Evaluation Methodology" - Addison-Wesley, 1978

KOBU 72 KOBUS S., TRUITHOF A. & VIELLEVOYE L.
"Central Control Philosophy for the METACONTAL Switching System"
Electrical Communications Vol.47 No.3, 1972 pp 159-162

KOST 70 KOSTEN L.
"Simulation in Traffic Theory"
Sixth Internat. Teletraffic Congress, Munich
9-15 Sept. 1970 pp 441/1-411/8

KOSY 73 KOSY D.W.
"An Interm Empirical Evaluation of ECSS for Computer
System Simulation Development"
Symp. for the Simulation of Computer Sys. 19-20 June 1973
ACM-SIGSIM pp 79-90

KRAS 67 KRASNOW H.S.
"The Process View and Management Systems"
Proc. IFIP Conf. on Simulation Programming Languages
May 1967 Oslo, Norway pp 1-12

KUCH 75 KUCHIBHOTLA T.S. and RICHARD Y.K.
"On the Performance of Certain Multiprocessor Computer Organisations"
IEEE Trans. on Computers, Vol.C-24 No.11 1975 pp 1066-1074

LAMP 68 LAMPSON B.W.
"A Scheduling Philosophy of Multiprocessing Systems"
Comm. of ACM Vol.11 No.5 May, 1968 pp 347-360

LAMP 79 LAMPREABE M.A. DEL COSO
"Environment Simulation for Real-Time Software Processes"
Electrical Communication (GB) Vol.54 No.2 1979 pp 143-148

LASK 65 LASKI J.G.
"On Time Structure in 'Monte Carlo' Simulations"
Operational Res. Qutly. Vol.16 No.3 Sept.1965 pp 329-339

LAVE 67 LAVE R.E.
"Time Keeping for Simulations"
Jrnl. of Industrial Eng. Vol. XV111, Part 7 1967 pp 389-394

LAVE 75 LAVENBERG S.S. and SULTZ D.R.
"Regenerative Simulation of a Queuing Model of An Automated
Tape Library"
IBM Jrnl. of Res.& Dev., Vol. 19 No.5, Sept.1975 pp 463-475

LAWR 72 LAWRENCE G.N. and HYDE P.J.
"The Mark 1 Processor : The System and its Applications"
GEC Telecomm. Journal No.39, 1972 pp 6-12

LAWR 77 LAWRENCE G.N.
"Program Organisation : Application Programs"
2nd IEE Vac. Sch. on Stored Program Control of Telephone
Switching Sys. Univ. of Essex, 11-16 Sept. 77

LAWS 79 LAWSER J.J. and SHEINBEIN D.
"Realising the Potential of the Stored Program Controlled Network"
Bell Labs. Records, Vol.57, No.3, March, 1979 pp 85-89

LEAK 72 LEAKEY D.M. and SIGRIST P.
"The Role of SPC in Telephone Switching Systems"
GEC Telecomm. Journal No.39, 1972 pp 4-5

LEAK 77 LEAKEY D.M.
"Switching in Space and Time"
Proc. IEE Vol.124, No.1, Jan. 1977 pp 17-24

LEHM 68 LEHMAN M. and ROSENFIELD J.L.
"Performance of a Simulated Multiprogramming System"
Proc. AFIPS-FJCC Vol 33, 1968 pp 1431-1442

LEO 68 LEO J.C.
"S3, The System and Software Simulation"
Proc. 2nd Conf. on App. of Simulation Dec.1968 N.Y. pp 282-285

LOKE 74 LOKEN B.
"TETRASIM Model Description"
Norwegian Computing Centre. Oslo, Norway Pub.S-26 Aug, 1974

LOVD 77 LOVDAL E. and BELSNES D.
"Use of Simulation Techniques in Actual Network Implementation"
Proc. Symp. on Simulation & Modelling of Data Networks,
Oslo, Sept. 1977 pp 1-16

LUCA 71 LUCAS H.C.
"Performance Evaluation and Monitoring"
Computing Surveys, Vol.3, No.3, Sept. 1971 pp 79-90

MACD 73A MACDOUGAL M. and McALPINE J.S.
"Computer System Simulation with ASPOL"
Proc. Symp. on the Simulation of Computer Sys. 1973 pp 93-103

MACD 73B MACDOUGAL M.
"Simulating the NASA Mass Data Storage Facility"
Symp. on the Simulation of Computer Sys. 4-6 June, 1974
pp 32-43 SIGSIM-ACM

MACH 74 MACHESON S.T.
"Simulation Program Generators"
Simulation, 23(6), Dec. 1974 pp 181-189

MAGU 72 MAGUIRE J.H.
"Discrete Computer Simulation-Technology and Applications
- the Next Ten Years"
AFIPS-SJCC Proc. Vol.40, 1972 pp 815-825

MAND 76 MANDIGO P.D.
"No. 2B ESS : New Features for a More Efficient Processor"
Bell Labs. Records. Vol.54 No.1 1976 pp 304-309

MARK 77 MARK A., EGGENBERGER O. and NEHMER J.
"Experiences in the Design and Implementation of a Structured
Real-time Operating System"
Proc. IFAC/IFIP Workshop on Real Time Program. Eindhoven,
Netherlands, 20-22, June, 1977 pp 133-137

MART 79 MARTIN J.
"System X"
P.O.E.E.J. Vol.71, Jan 1979 pp 221-224

MAY 80 MAY C.A.
"Communication"
Electroncis and Power, Vol.26. No.1 Jan 1980 pp 56-62

MCQU 81 McQUADE E. , GRAY H. and SALIH A.M.
"A Process Interaction Approach to the Simulation of a
Multiprocessor Computer System"
To be pub. in SIMULATION, the Technical Journal of the
Society of Computer Simulation, U.S.A.

MIHR 71 MIHRAM G.A.
"Simulation : Statistical Foundation and Methodology"
Academic Press, N.Y. 1971

MIHR 72 MIHRAM G.A.
"Some Practical Aspects of the Verification and Validation
of Simulation Models"
Oprs. Res. Qtrly. Vol.23, No.1, 1972 C, pp 17-29

MORA 79 MORALEE D.
"The System X Project"
Electronics and Power, Vol.25 No.8 August, 1979 pp 544-551

NAUR 63 NAUR P. (ED.)
"Revised Report on the Algorithmic Language ALGOL 60"
CACM Vol6. No.1, 1963, pp 1-17

NAYL 66 NAYLOR T.H., BLINTFY J.L. BURDICK D.S. and CHU K.
"Computer Simulation Techniques" - Chapter 8
John Wiley & Sons Inc. 1966

NAYL 67 NAYLOR T.H. and FINGER J.M.
"Verification of Computer Simulation Models"
Management Science, Vol.14, No.92 Oct 1967 pp 92-101

NIEL 67 NIELSEN N.R.
"The Simulation of Time-Sharing Systems"
Communications of the ACM Vol.10, No.7 July 1967 pp 397-412

NISS 79 NISSEN J.C.D. and GIEGER G.V.
"A Fault-Tolerant Multimicroprocessor : for Telecommunications
and General Applications"
GEC Telecommunications Ltd., Coventry England 1979

NOE 74 NOE J.D. and NUTT G.J.
"Validation of a Trace-Driven CDC 6400 Simulation"
Proc. AFIPS-FJCC 1974 Vol.40 pp 749-757

NUTT 74 NUTT G.J.
"The Simulation of Computer Systems in a University
Environment"
Proc. ACM-SIGSIM Symp. on Simultion of Computer Systems
Gaithersburg, U.S.A. 4-6, June, 1974 pp 25-29

OPPE 68 OPPENHEIMER G. and WEIZER N.
"Resource Management for a Medium Scale Time-Sharing
Operating System"
Communications of the ACM Vol.11, No.5 May, 1968 pp 313-322

OWEN 73 OWEN G.J.
"ROLLBACK - A Method for System Recovery"
Proc. IEE Conf. on Software Engineering for Telecommunications
Switching Systems, IEE Pub. No.97 1973 pp 118-124

OWEN 76 OWEN G.J.
"Hardware vs. Software : Techniques for Obtaining the
Optimum Balance"
Conf. on Software Eng for Telecomm. Switching Systems
18-20 Feb. 1976 IEE Pub. No.135 pp 77-80

PALM 71 PALME J.
"A Reader Comments on Henry Kleine's Initial Survey"
Simulation, August, 1971 pp 94

PASA 73 PASAHOW E.J.
"Testing Real-time Systems with Simulation"
Proc. ACM-SIGSIM Symp. on Simulation of Computer Systems
Gaithersburg, USA 19-20 June, 1973 pp 40-45

PETR 68 PETRONE L.
"On a Simulation Language Completely Defined onto the
Programming Language PL/1"
Proc. IFIP Conf. on Simulation Programming Languages Oslo
Norway, May, 1967 pp 61-85

PIEN 73 PIENE J.O.
"Simulation of Computer Systems - A Case Study on the
CDC 6000/SCOPE System"
Norwegian Computing Centre, Pub. No. S-46 April, 1973

POVE 65 POVEY J.A. and COLE A.C.
"The Use of Electronic Digital Computers for Telephone
Traffic Engineering"
P.O.E.E.J. NO. 58 1965 pp 203-209

PRIT 69 PRITSKER A.A. and KIVIAT P.J.
"Simulation with GASP II"
Prentice-Hall Inc., Englewood, Cliffs, N.J. 1969

RAND 68 RANDEL B. and KUEHNER C.J.
"Dynamic Storage Allocation Systems"
Communications of the ACM Vol 11 No. 5 May, 1968 pp 297-306

REIN 68 REINO A.M. and FRED C.D.
"Simulation Design of a Multiprocessing System"
Proc. AFIPS-FJCC, Vol.33, 1968 pp 1399-1410

REIT 71 REITMAN J.
"Computer Simulation Applications"
John Wiley & Sons Inc. 1971

SALI 79 SALIH A.M.
"Digital Computer Simulation as a Tool for Evaluating the
Performance of SPC Telephone Exchanges : Experience with a Model
for the GEC Mark II BL and Suggestions for Future Work."
Memo. No. TRL/809 GEC Hirst Research Centre, England, Oct. 1979

SALI 81 SALIH A. M.
"A Simulation Model for the DMNSC Exchange"
Memo. No. TRL/880 GEC Hirst Research Centre, England, Jan. 1981

SALI 81A SALIH A.M., McQUADE E., CUNNINGHAM R.J., GRAY H. and OWEN G.J.
"Multi-Level Process Simulation as a Design Aid for SPC Switching
Systems" - to be presented at the 9th Annual SIMULA Users'
Conference 9-11th Sept. 1981 Geneva, Switzerland

SALT 74 SALTZER J.H.
"A Simple Linear Model of Demand Paging Performance"
Comm. of the ACM Vol.17, No. 4, April, 1974 pp 181-186

SCHM 79 SCHMDIT C.E., RABINER L.R. and BERKLEY D.A.
"A Digital Simulation of the Telephone System"
BSTJ Vol. 58, No.4, 1979 pp 839-855

SEDG 70 SEDGEWICK R., STONE R. & MACDONALD J.N.
"SPY - A program to Monitor OS/360"
Proc. AFIPS-FJCC, Vol.35, 1970 pp 119-128

SHAN 73 SHANNON R.E. & WAYATT M.W.
"Discrete-Simulation Language Users' Survey Revisited"
Simulation Vol.19 May, 1973 pp 168-169

SHER 72 SHERMAN S., BASKET F. & BROWNE J.C.
"Trace-Driven Modelling and Analysis of CPU Scheduling
in a Multiprogramming System"
Comm. of the ACM, Vol.15, No.12 Dec. 1972 pp 1063-1069

SIMU 79 SIMULATION NEWSLETTER
"Language Comparison - State-of-the-Art"
No.2, Winter 1979, Publ by Centre in Simulation, University of
Lancaster, Lancaster, U.K.

SIMU 75 "SIMULA USERS GUIDE"
IBM/System/360 Revised Ed. 1975, NCC Pub. No.S-24-1

SINC 80 SINCLAIR T.M.
"PPU Model : Functional Description"
Report GEC/TGR/104 GEC Telecomm. Ltd. England Feb. 1980

SPIE 72 SPIEGEL M.R.
"Statistics"
Schaum's Outline Series, McGraw-Hill, 1972

STAN 68 STANLEY W.I & HERTEL H.F.
"Statistics Gathering and Simulation for the APOLLO Real-Time
Operating Syste" - IBM System Jour. Vol.7, No.2. 1968 pp 85-102

SVOB 76 SVOBODOVA L.
"Computer Performance Measurement & Evaluation Methods : Analysis
and Applications" -- American Elesevier Pub. Co. Inc. 1976

SWAM 78 SWAMINATHAN K.
"Computers in Telephone Districts"
Telecommunications (India) Vol.28, No.1, June, 1978 pp 5-7

TEIC 67 TEICKROW D., LUBIN J.F. & TRUIT T.D.
"Computer Simulation - Discussion of the Technique & Comparison
of Languages " - Simulation Vol.19, 1967 pp 181-190

TEOR 73 TEOREY J.T. & MERTEN A.G.
"Consideration of the Level of Detail in Simulation"
Proc. ACM-SIGSIM Symp. - Simulation of Computer Systems, June, 73

THOM 79 THOMAS J.C. & PAUL J.H.
"Analysis and Simulation of No.4 ESS Network Performance"
IEEE Trans. on Comm., Vol. COM-27, No.1 Jan, 1979 pp 1-9

TIPP 77 TIPPLER J.
"Review of Switching Requirements - Role of SPC"
IEE Vac. School on Stored Program Control of Telecomm. Switching

Systems. 11-16 Sept. 1977 University of Essex.

TIPP 79 TIPPLER J.
"Architecture of System X : Part 1 - An Introduction to the System X family"
POEEJ, Vol.72, Oct. 1979 pp 138-141

TOCK 60 TOCKER K.D. and OWEN D.G.
"The Automatic Programming of Simulations"
Proc. of 2nd International Conf. on Oper. Res. pp 50-68

TOCK 65 TOCKER K.D.
"Review of Simulation Languages"
Oper. Res. Quarterly, Vol.16, No.2, June, 1965 pp 189-217

TOGN 72 TOGNETTI K.P. and BRETT C.
"SIMSCRIPT II and SIMULA 67 - A Comparison"
The Australian Computer Journal, Vol.4, No.2, May, 1972 pp 50-57

TSAO 72 TSAO R.F. and MARGOLIN B.H.
"A Multi-Factor Paging Experiment : II Statistical Methodology"
In Statistical Computer Performance Evaluation, Academic Press 1972 pp 135-158

UNGE 72 UNGER B.W.
"A Simulation Model for Evaluating Computing Resource Architecture and Management"
Ph.D Thesis, University of California, San Diego, 1972

UNGE 75 UNGER B.W.
"Validation of a Computer Resource Allocation Model"
Summer Computer Simulation Conference. San Francisco July 1975

UNGE 77 UNGER B.W.
"A Computer Resource Allocation Model with some Measured and Simulation Results"
IEEE Trans. on Computers, Vol.C-26, No.3 March 77 pp 243-259

VAUC 71 VAUCHER J.G.
"Simulation Data Structures Using SIMULA 67"
Winter Simulation Conf. 1971 pp 255-260

VAUC 73 VAUCHER J.G.
'"WAIT-UNTIL' Algorithms for General Purpose Simulation Languages"
Proc. Winter Simulation Conf. 1973 pp 177-183

VIRJ 72 VIRJO A.
"A Comparison of Some Discrete Event Simulation Languages"
NORDATA 72 Conf., Helsinki, 1972

WARD 69 WARD M. & ACKROYD E.
"A Multiprocessor Control Unit for a Stored Programe Switching System"
In Conf. on Software Eng. for Telecom. Switching Systems 21-25 April, 1969 London, IEE Pub. No. 52

WARD 72A WARD M.
"The Mark II Communications Processor"
GEC Telecommunications Journal No.39 pp 13-15 1972

WARD 72B WARD M. and NISSEN J.C.D.
"Software Security in a Stored Program Controlled Switching System"
Proc. IEEE International Switching Symp. Record Boston, U.S.A 1972

WEAT 73 WEATHERBY J.E.
"Simulation in the Evaluation of an Executive System Design"
Proc. Symp. on Simulation of Computer Systems, Gaithersburg, Md. U.S.A. June, 1973 pp 226-232

WILL 74 WILLIS R.R.
"Structured Model Development Techniques"
Proc. ACM-SIGSIM Symp. on Simulation of Computer Systems 4-6, June, 1974, pp 132-135

WILS 78 WILSON J.C.T.
"Research in the Centre"
Simulation Newsletter, Centre In Simulation, University of Lancaster, Lancaster, U.K.

YAN 78 YAN J.
"Simulation of a Business Communication System Switching Network"
Proc. Winter Simulation Conf. 4-6 Dec. 1978 pp 751-757

ZURC 68 ZURCHER F.W. and RANDEL B.
"Iterative Multi-level Modelling - A Methodology for Computer System Design"
4th IFIP Conf. Proc. Edinburgh 1968 North Holland Inc.