

A GRAPHICAL SIMULATION SYSTEM

Behzad Zamanzadeh

**June 1986
CSD-860036**

UNIVERSITY OF CALIFORNIA

Los Angeles

A Graphical Simulation System
for Manufacturing Systems

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Computer Science

by

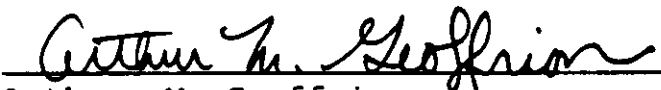
Behzad Zamanzadeh

1986

The dissertation of Behzad Zamanzadeh is approved



Robert. B. Andrews




Authur. M. Geoffrion



Lawrence. P. McNamee



Edward. C. Russell



Russell. A. Westmann



Michael. A. Melkanoff, Committee Chair

University of California, Los Angeles

1986

To
my father Habibollah Zamanzadeh
and my mother Iran Yadidi

CONTENTS

	LIST OF FIGURES	vi
	LIST OF TABLES	ix
	ACKNOWLEDGEMENTS	x
	VITA	xi
	ABSTRACT	xii
1	INTRODUCTION	1
1.1	Purpose And Accomplishments	1
1.2	Rationale And Background	4
1.3	Survey Of Existing Simulation Languages	13
1.3.1	Selection Of The Base Languages	17
1.4	Graphical Man Machine Interface For Simulation	23
2	GRAPHICAL SIMULATION SYSTEM	28
2.1	Overview Of Graphical Simulation System	30
2.1.1	User Interface	33
2.1.2	Block-Command Symbolic Language Compiler	37
2.1.2.1	Pass 1 Of The BCSL Compiler	43
2.1.2.2	Pass 2 Of The BCSL Compiler	43
2.2	Block Graphic Symbolic Language	45
2.2.1	BGSL Elements And Programming Concept	48
2.2.2	Block Command Symbolic Language	61
2.3	Methodology Of Translation To SIMSCRIPT	63
2.4	Detail Description Of The BCSL Compiler Architecture	69
2.5	Description Of Key Block Processor Routines	78
2.6	Differences Between GPSS And BCSL	88
2.6.1	ADVANCE Block	90
2.6.2	GENERATE Block	93
2.6.3	FUNCTION Commands	96
2.6.4	Floating Point And Arthmetic Statements As Operands	98
2.7	Output Generator And Output Files	99
3	CLASSIC APPLICATIONS AND CASE STUDIES	102
3.1	One-Line, One-Server Queuing System	104
3.1.1	Statement Of The Problem	104
3.1.2	Discussion Of The Model	104
3.1.3	Discussion Of SIMSCRIPT Equivalent	109
3.1.4	Discussion Of Results	110
3.2	Simulation Of The Production Shop	123
3.2.1	Statement Of The Problem	123
3.2.2	Discussion Of The Model	125

3.2.3	Discussion Of SIMSCRIPT Equivalent	136
3.2.4	Discussion Of The Results	143
3.3	A Bus Stop Simulation	157
3.3.1	Statement Of The Problem	157
3.3.2	Discussion Of The Model	158
3.3.3	Discussion Of SIMSCRIPT Equivalent Of The Bus Model	165
3.3.4	Discussion Of Results	168
4	ADVANCED FEATURES OF THE GRAPHICAL SIMULATION SYSTEM	179
4.1	Modeling Flexible Manufacturing Systems	180
4.1.1	Statement Of The Problem	184
4.1.2	Discussion Of The Model	187
4.1.3	Discussion Of SIMSCRIPT Equivalent For The FMS Model	196
4.1.4	Discussion Of The Results Of The FMS Simulation	209
4.2	Description Of New FMS Blocks And Their Translation	214
4.2.1	Define Transporter (DEF TRAN)	214
4.2.2	Define Workstation (DEFWSTAT)	217
4.2.3	Request Workstation (REQWSTAT)	219
4.2.4	Request Transport (REQTRANS)	222
4.2.5	Enter Station (ENTERSTAT)	225
4.2.6	Leave Station (LEAVESTAT)	228
4.3	User Written SIMSCRIPT	228
4.3.1	SIMSCRIPT Line (SIMSLINE) Block	229
4.3.2	Append File	229
4.4	Generation Of Customized Simulation Systems	234
4.4.1	Generation Of New Blocks In BCSL	234
4.4.2	WILDCARD And Expansion Of BCSL	237
4.4.3	MACRO Blocks	240
4.5	AN ALTERNATIVE SOLUTION TO THE PRODUCTION SHOP MODEL	243
5	CONCLUSIONS AND RECOMMENDATIONS	252
5.1	SUMMARY	252
5.2	Performance	256
5.3	Future Research Possibilities	258
5.4	Limitations And Potential Improvements	265
	BIBLIOGRAPHY	268
	APPENDIX	
A	SUMMARY OF BCSL MODEL BLOCKS	272
B	SUMMARY OF BCSL DEFINITION AND CONTROL BLOCKS	280
C	SUMMARY OF BCSL STANDARD NUMERICAL ATTRIBUTES	285
D	SYSTEM CONFIGURATION AND USERS GUIDE	287

LIST OF FIGURES

FIGRUE	PAGE
1-1. Languages Used in the Development of GSS.....	10
2-1. Overview of Graphical Simulation System.....	32
2-2. Overview of User Interface.....	36
2-3. Block Command Symbolic Language Compiler.....	38
2-4. Data Flow in Graphical Simulation System.....	41
2-5. Adding user written BCSL.....	42
2-6. Silhoutte of a Typical Block Command Symbolic Language.....	47
2-7. "Build Model" Selection Menu Display.....	49
2-8. "Add Block" Selection Menu Display.....	50
2-9. GENERATE Block Parameter Query Display.....	51
2-10. ADVANCE Block Parameter Query Display.....	52
2-11. Model Segments.....	55
2-12. Segment with Two TERMINATE Blocks.....	56
2-13. Data Flow in Graphical Simulation System.....	57
2-14. A BGS� Model and its BCSL Translation.....	60
2-15. Translation of a Model Segment.....	68
2-16. Primitive Elements in Raw SIMSCRIPT Text.....	73
2-17. Calling Sequence of BCSL Compiler	77
2-18. Translation of a GENERATE Block.....	81
2-19a. Translation of TERMINATE Block.....	82
2-19b. Translation of ADVANCE Block.....	83
2-20a. Translation of SEIZE Block.....	84
2-20b. Translation of a RELEASE Block.....	85

2-21.	Translation of a TEST Block	87
2-22.	Output Generator.....	101
3-1.	BGSL Model for the Barber Shop.....	107
3-2.	BCSL Model for the Barber Shop.....	108
3-3.	SIMSCRIPT Equivalent Model for the Barber Shop.....	112
3-4.	Simulation Results for the Barber Shop Model	120
3-5.	BGSL Model for the Production Shop.....	129
3-6.	BCSL Model for the Procution Shop	134
3-7.	SIMSCRIPT Model for the Production Shop.....	145
3-8.	Simulation Run Results for Production Shop.....	155
3-9.	BGSL Model for the Bus Stop.....	160
3-10.	BCSL Model for the Bus Stop.....	163
3-11.	SIMSCRIPT Model for the Bus Stop.....	170
3-12.	Simulation Run Results for the Bus Stop Model.....	178
4-1.	A Flexible Manufacturing System Floor Plan and Job Routes.....	182
4-2.	Unfolded Visitation Sequence.....	183
4-3.	BGSL Model for the Flexible Manufacturing System.....	189
4-4.	BCSL Model for the Flexible Manufacturing System.....	194
4-5.	SIMSCRIPT Model for the Flexible Manufacturing System.....	199
4-6.	Simulation Results for FMS Model.....	211
4-7.	Pictorial Equivalence of the REQWSTAT Block.....	221

4-8.	Equivalent of REQTRANS Block.....	224
4-9.	Equivalent of ENTERSTAT Block.....	227
4-10.	Append File Concatenation.....	230
4-11.	Usage of SIMSLINE Block.....	233
4-12.	Calling Sequence for a New Block.....	236
4-13.	WILDCARD Block Translation.....	239
4-14.	Equivalent of the "M" Block	241
4-15.	BGSL Model Representing JOB1 Transaction....	245
4-16.	Improved BCSL Model for the Production Shop.....	246
4-17.	SIMSCRIPT Equivalent for JOB1 Transaction...	248
4-18.	Simulation Results for the Improved Production Shop Model.....	250
5-1.	Animation Of Simulation Run.....	260
5-2.	Animation Routines and Block Commands.....	262
5-3.	An Expert Simulation System.....	264
D-1	GSS Operations Environment.....	288
D-2.	Main Menu.....	291
D-3.	System Utilities.....	292
D-4.	Simulate Model.....	293
D-5.	Model Settings.....	294

LIST OF TABLES

TABLE		PAGE
2-1.	ADVANCE Block.....	90
2-2.	Statistical Distribution Functions.....	92
2-3.	Operands of GENERATE Block.....	94
2-4.	Function Command.....	94
3-1.	Column Definition for Facility Report	121
3-2.	Column Definition for Queue Report	122
3-3.	Composition of Machine Groups in Production Shop.....	124
3-4.	Visitation Sequences and Mean Operation Times for the Three Job Types.....	124
A-1.	Explanation of the Abbreviations	273

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Professor Michael A. Melkanoff, for his constant guidance, encouragement and instructions during the course of this research and my graduate work. I am also grateful to Dr. Edward C. Russell for his support, guidance and many valuable suggestions that were indispensable for my research.

I am thankful to my committee members, Professors McNamee, Westmann, Geoffrion and Andrews for their time and valuable discussions.

Special thanks is also extended to Teledyne Controls executive management for their encouragement and allowing flexible working hours during the performance of this research.

VITA

March 29,1955 Born, Tehran, Iran

1973-1976 Programmer,
Material and Energy Research Center

1976 BS in Electrical Engineering,
Arya Mehr University of Technology

1977-1979 Systems Programmer, CompuCorp

1980 Masters in Computer Science,
University of California at Los Angeles

1979-1983 Software Engineer, Teledyne Controls

1983-1984 Communication Group Supervisor,
Teledyne Controls

1984-1986 Software Project Manager,
Teledyne Controls

1986 Engineering Section Manager,
Teledyne Controls

ABSTRACT OF THE DISSERTATION

A Graphical Simulation System
for Manufacturing Systems

by

Behzad Zamanzadeh

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1986

Professor Michael A. Melkanoff, Chair

An on line graphical simulation system has been developed based on an iconic simulation language. This system facilitates the simulation of manufacturing systems and Flexible Manufacturing Systems. Ease of use by unsophisticated users, expandability of the language and flexible output generation are the main characteristics of the Graphical Simulation System.

It has been demonstrated that iconic languages are practical and convenient for unsophisticated users. Several general purpose models in addition to a manufacturing job shop model and a flexible manufacturing model were built and are discussed in details.

Expandability of the underlying iconic language has made it possible to develop custom-made Flexible Manufacturing Simulation System, which raises a new concept in software engineering. Where software tools for development of customized systems based on very high level languages (graphical or natural) are the way of the future.

1 INTRODUCTION

1.1 Purpose And Accomplishments

The purpose of this research has been to develop an on-line Graphical Simulation System for manufacturing systems. This system is based on a block oriented simulation language. It permits the user to describe the system model, which in turn is translated into a computer program that simulates the manufacturing system. Ease of use by unsophisticated users, capability of on-line graphical interface, expandability of language and flexible output generation have been the major goals of this system.

During the course of this research the proposed Graphical Simulation System has been developed, tested and demonstrated with capabilities and potential beyond the original expectations. The system is constructed via three major building blocks; the graphical user interface, the BCSL language compiler and the simulation results output generator. The graphical user interface for this system (including the BCSL generator) has been developed by Edmond Mesrobian [28] as part of his masters thesis. The present disertation has produced the following results

1. Design of an iconic graphical simulation language based on GPSS.
2. Definition of the equivalent SIMSCRIPT code for each block in the proposed language and development of the methodology for translating the icons (block-commands) into SIMSCRIPT.
3. Development of the compiler which translates the BCSL language into SIMSCRIPT (up to 40 blocks have been implemented so far).
4. Demonstration of successful operation of the proposed system by running several simulation tests (with known results).
5. Simulation of a manufacturing job-shop model as well as several complex general purpose models.
6. Proof of the expandability of the language by defining new blocks for simulation of Flexible Manufacturing Systems and testing them by running a FMS simulation model.
7. Experimentation with the "macro block" concept while implementing the FMS new blocks. In addition provisions for guidelines and techniques for further expansion of the language.

The major achievement of this research is that we have demonstrated that graphical languages are practical and that systems designed for the unsophisticated end users with menu driven, interactive iconic interfaces are more convenient to use than to those based on procedural and command driven languages. In addition the expandability of the Graphical Simulation system has made it possible to use it as a tool in the development of custom made simulation systems. This raises a new concept in software engineering, where the development of translators for very high level programming languages as a tool for development of customized systems will be more and more common.

1.2 Rationale And Background

Simulation is one of the most powerful analysis tools available for the design and operation of complex processes or systems, including manufacturing systems.

A modern manufacturing system usually consists of a complex integration of machines, material handling equipment, and operators, which produce a variety of parts. In such systems, operations are performed at stations, each containing a group of similar machines. Material handling equipment is used to transport parts between stations and operators who may be required to operate machines or material handling equipment.

Manufacturing systems have many of the same types of problems. These problems can result in such symptoms as insufficient throughput, late production, high inventories, or highly variable work loads, all of which are undesirable. The problems which cause these symptoms are complex and are often difficult to identify and alleviate.

Through the use of simulation tools, techniques and methodologies the system analyst can accurately predict the behavior and operational characteristics of complex manufacturing systems before they are actually

installed. Simulation makes it possible to study the effects of informational, organizational, and environmental changes on the operation of a manufacturing system by making alterations in the simulation model rather than experimenting directly on the system itself. However, as manufacturing systems have become more complex, shortcomings in existing simulation modeling languages have become increasingly apparent.

Usually a simulation study starts by defining the system to be modeled and describing it in terms of logic flow diagrams and functional relationships. But eventually the analyst is faced with the problem of describing the model in a language acceptable to the computer which is used. Using conventional simulation languages requires programming knowledge and experience. In addition the user must become familiar with simulation concepts and internal techniques employed by the simulation language. As a result users have hesitated to use simulation in order to resolve design and operational problems in the manufacturing environment.

A survey of the aerospace industry, conducted by Professor M.A. Melkanoff shows that a major problem has been the fact that "simulation is done by someone else." Usually a separate group within the same company or outside specialists are used to do the modeling and simulation. As a result modifications to the model are inconvenient and often take a long time before results are in the hands of the analyst.

As pointed out by Oren and Zeigler [50], one of the deficiencies of simulation software is in the user interface. Lack of a good user interface makes simulation software difficult to learn and to operate, especially for someone without solid computer skills (Standridge [45]).

In recent years the ability to model complex real-world systems within a "no programming" environment has become more and more appealing. Users groups have shown a great desire for systems that the user can easily relate to, are convenient to use and provide natural languages and symbols to interact with the user. Therefore as graphics have become increasingly common as a major component of modern computer systems, the inherent naturalness of image-assisted communication has motivated the development of interactive and

iconic (1) graphic interfaces.

Iconic interfaces are easier to use, and, by taking advantage of our visual processing abilities, they can provide superior information carrying capabilities for user interaction.

As a result of the above observation we decided to develop an interactive graphical simulation system, which allows the user to build models using symbols and commands that he is familiar with and which are commonly used in his profession. The emphasis of this system is on manufacturing systems simulation as this area has the most demand. In the course of designing such a system the following goals were set:

1. To develop an intermediate language, based on an existing general purpose simulation language, which lends itself to graphical representation so that simulation results can be verified by comparing them with results from existing case studies. This provides the general purpose simulation capabilities within the system.

(1) Iconic communication concerns the use of images to convey ideas or information. The images are chosen to relate ideas either by resemblance (pictorial), by analogy (symbol), or by being selected from a previously defined and learned group of arbitrarily designed images (sign).

2. To select a powerful and widely used simulation language as the target language. The graphical model will be translated into the target language, then the equivalent model will be compiled and executed. This would eliminate the need for development of an internal simulation mechanism and other features which exist in all simulation system.
3. To provide iconic symbols within the graphical language which makes the modeling of manufacturing systems easier for the end user. Figure 1-1 shows the relationship between the principal languages used in the development of the Graphical Simulation System.

The General Purpose Simulation System (GPSS) has been selected as the basis for the intermediate simulation language (BCSL), and SIMSCRIPT as the target language.

This thesis describes the methodology and technical descriptions of the Graphical Simulation System. It is divided into five chapters and five appendices. The remainder of Chapter 1 presents a brief survey of existing simulation languages and the rationale behind the selection of the target simulation language and the

intermediate simulation language basis. Later this chapter provides a summary of current attempts to develop interactive Graphical Simulation Systems.

Iconic Graphic Model in
Block Graphic Symbolic Language



Intermediate Simulation Model in
Block Command Symbolic Language



Target Equivalent Simulation Model
in SIMSCRIPT



Machine Code

Figure 1-1. Languages Used in the Development of GSS

Chapter 2 contains an overview of the Graphical Simulation System followed by a detailed technical description of the Block Graphic Symbolic language, the Block Command Symbolic language and the BCSL compiler. Chapter 2 discusses the methodology behind the Translation of BCSL and the differences between GPSS and BCSL.

Three case studies have been selected to be studied in detail in Chapter 3. These problems were taken from the GPSS book by Schriber [41]. This makes it possible to compare the simulation model and simulation run results of the Graphical Simulation System with known results given in Schriber's book.

Chapter 4 discusses the advanced features of the Graphical Simulation System specifically aimed at the Block Command Symbolic language. This chapter describes how a model can be developed for Flexible Manufacturing systems and how GSS can be expanded by adding new blocks to the Block Graphic Symbolic language and Block Command Symbolic language. The use of MACRO blocks to develop new blocks and an alternative model for production shop simulation is also discussed.

Chapter 5 contains the conclusions drawn on the basis of the experience gained from development of the Graphical Simulation system and its usage to simulate general purpose and manufacturing systems. Performance and potential problems with the Graphical Simulation System are also discussed. Finally, future research possibilities based on this research are presented.

Appendix A,B and C contain a summary of all the blocks implemented in the GSS so far. Appendix D contains the GSS users guide and information about its operating environment.

1.3 Survey Of Existing Simulation Languages

Simulation languages are languages in the more general sense. They go beyond simply linking the user with the computer as a means of conversing. They afford the user an aid to problem formulation. Having a vocabulary and a syntax, simulation languages are descriptive, and consequently their users tend to think in them. Kiviat [22] believes that the two most important reasons for utilizing simulation languages as opposed to general purpose languages, are programming convenience and concept articulation. Concept articulation is important in the modeling phase and in the overall approach taken for system experimentation.

Shannon and Phillips [44] describe another advantage of simulation language, namely their use as communication and documentation devices. Emshoff and Sisson [14] believe that all simulations require certain common functions which make simulation languages different from general algebraic or business programming languages. Among those functions are the need to create random numbers, advance time (either by one unit or to the next event), record data for output, perform statistical analysis, arrange outputs in specified formats and detect inconsistencies and error conditions.

Specifically, simulation of discrete events requires specific operations to determine the type of events, to store and retrieve data from lists (including the event list), call subroutines to adjust the state variables as a result of the events, and identify specific state conditions.

Simulation languages are categorized according to the conceptual manner through which the host language represents real-world activities (world view of a simulation language). The implication of a language's world view is that when utilizing a particular language, the user is forced to view the world in the same manner (Shannon [43]), and accordingly this restricts the arena of language application. Three world views are prevalent: event orientation, activity scanning, and process orientation.

Using an event-oriented world view, the system to be modelled is described in terms of status disturbing events. The analyst constructs a simulation model by defining each event which can occur in the system, specifying the cause and effects of each event, creating mechanisms to execute event changes within the simulation model, logically linking each event to another, updating time and statistics at each status

disturbing event and collecting statistics of interest. SIMSCRIPT (Kiviat [23]), SPEED, MAP-1 (Miner [30]) and special options within SLAM II (Pritsker [38]) are basically event-oriented.

In some simulation models, events which are known to occur cannot be scheduled. However, it is usually possible to define the event mechanism in terms of those physical influences which trigger the event at an unknown time. In these models the simulation time is increased by units of time and all the activities in the model are scanned to find out if a change of state (event) has occurred. The mechanism by which events of this nature are monitored is called activity scanning and is normally used for continuous simulation modeling. Activity scanning is provided by MICRONET, SIMAN (Pegden [37]) and SLAM II.

Using a process orientation, the simulation language views the world as being composed of sequences of events which occur in a definite pattern. For example, a single channel queueing model possesses a waiting line, a service mechanism, and structured rules describing how items move through the service system. The entire sequence of activities can be combined into a single simulation "Command Module" which executes a

fixed set of processing rules.

Although queue disciplines and service time distribution times may vary, once they are specified, the sequence of events is uniquely determined. GPSS (Schriber [41]), Q-GERTS, GENS, SIMPL/1, SIMULA, GASP, and the new version of SIMSCRIPT II.5 (the latest version of SIMSCRIPT) contain process oriented world view. The fundamental and direct influence in this world view of these languages is the ability to model complex real-world systems within a "no programming" environment. GEMSII, Q-GERT, SIMAN, MICRONET, PSIM, Interactive, IDSS 2.0 (Yancey [49]) and portions of SLAM II all use network-type symbolism to specify system behavior and control.

A summary by Christy and Watson [11] of industry practices shows that 40% of their respondents used GPSS (this includes GPSS/H which is an improved implementation of GPSS) as their primary simulation language, 19% used SIMSCRIPT and 6% used SLAM.

1.3.1 Selection Of The Base Languages -

In order to develop the Graphical Simulation System we need to select a graphically representable simulation language as the base for the intermediate language and a powerful simulation language as the target language. The GSS first translates the graphical models into the intermediate language and then into the target simulation language. The resulting model is equivalent to the original graphical model. Finally, compilation and execution of the equivalent model in the target language will generate the simulation results. The following criteria have been established for selection of the target simulation language:

1. The simulation language must support a process-oriented world view because process-oriented simulation languages tend to be more structured and modular which makes it possible to build an iconic (symbol oriented) language on top of them.
2. The selected simulation language must be widely used and available on most commercial computers.
3. The selected simulation language must also support certain features of a popular general purpose language so that the user-written routines have

maximum flexibility and strength. These features must include arithmetic and list processing capabilities.

4. The language must provide flexible output generation.
5. It is desirable for the compiler to be written in the same language as the target language so that the need for an additional compiler is eliminated. For example, if SIMSCRIPT was selected as the Target language and FORTRAN selected to write the SIMSCRIPT Translator, both FORTRAN and SIMSCRIPT compilers would be needed to install the Graphical Simulation System. This makes the Graphical Simulation package more transferable across machines. Therefore, it is desirable for the selected language to support the features needed to build a compiler, like Text processing capability.

The most popular process oriented language is GPSS (General Purpose Simulation System). The principal appeal of GPSS is its modeling simplicity in a non-programming environment. A GPSS model is constructed by combining a set of standard blocks into a block diagram which defines the logical structure of the

system. For this reason, GPSS was selected as the basis for development of the intermediate language (BCSL). However GPSS suffers from the following shortcomings which make it undesirable as the target language.

1. GPSS does not provide the features of a general purpose programming language and does not support list or text processing features.
2. GPSS is limited in computing power and lacks the capability of floating point and real arithmetic. As a result, the GPSS simulation clock is integer valued.
3. GPSS does not provide a flexible report capability which makes it undesirable for generating individual tailor made outputs of simulation results (GPSS/H provides this capability).
4. To generate samples from distributions other than the uniform distribution, a user written table function must be included in the model. This is in disagreement with the general strategy of building models using symbols. (This has been improved in GPSS/H.)

5. It is difficult to trace the model parameters step-by-step.
6. Modeling complex systems usually requires knowledge of the internals of GPSS.

Another well known process-oriented simulation language is SIMULA which is a superset of ALGOL60. SIMULA has many statements that make it attractive for performing discrete event simulation, including advanced list processing capability. SIMULA is not widely used in the United States but it has received considerable attention in Europe. For more discussions of SIMULA refer to Hills [18].

SIMSCRIPT II.5 is one of the most widely used simulation languages and is available on most commercial computers. It satisfies all of the selection criteria established for the target language. One of the principal appeals of SIMSCRIPT as a programming and simulation language is its English-like and free-form syntax. Programs written in SIMSCRIPT are easy to read and tend to be self documenting. Recently a "process-oriented world view" support has been added to SIMSCRIPT II.5 (Russell [40]).

SIMSCRIPT II is a computer language developed by Kiviat[23] , Villanueva and Markowitz and the history of SIMSCRIPT development is given by Markowitz, [27]. The language had originally been divided into five levels. The first three levels provide a power comparable to FORTRAN, ALGOL and PL/1. The fourth level contains the statements that provide a structure for modeling using entity, attribute and set concepts (list processing). The fifth level contains the statements for time advance, event processing, generation of samples, and accumulation and analysis of simulation generated data.

The process-oriented SIMSCRIPT simulation model consists of a preamble, a main program and process subprograms. The preamble is not part of the executable program and is used to define the elements of a model. The preamble also includes declarative statements for defining all variable types, arrays and needed statistical data accumulation and analysis.

The main program is used for initializing variables, scheduling the activation of the process subprograms, and starting the simulation: The process routines are used to represent arrivals of new objects in the model, and a representation of what happens to the objects within the modelled system. The process

routines also provide the mechanism for termination of the simulation. The SIMSCRIPT simulation language provides text processing, list processing and flexible output generation capabilities which make it the most desirable simulation language to be used as the base language for development of Graphical Simulation Systems.

1.4 Graphical Man Machine Interface For Simulation

In recent years, the importance of improving the user interface for simulation systems has been recognized by the industry. Aids have been developed to specify computer readable forms of models, manage simulation inputs and results, analyze simulation results, present simulation results graphically and animate simulation runs. Finally, since early 1983 several projects have been directed towards development of graphical input capabilities.

In this section I will present a summary of existing simulation systems and languages that have been introduced or are being enhanced to provide a graphical user interface for simulation modeling. These include; the very recent extension of SLAM II (TESS) and a new version of SIMAN, both of which are directed toward graphic input support, although I have not yet seen a demonstration of their graphical input capabilities. Also, the Modelmaster (GEFSM) and a graphic interface developed for GPSS (in Rensselaer Polytechnic) are discussed. None of these systems provide the ease of use, expandability and flexibility of the GSS

TESS, The Extended Simulation System (Standridge [46]), provides an integrated framework for performing simulation modeling. TESS provides capabilities for animating runs without programming, and generating graphs of all simulation results. Report generation and the post-run analysis of simulation results are included. Its goal is to provide also capabilities for graphically building SLAM II networks and schematic models.

A nonprocedural command language provides access to the graphical builders and forms system as well as selects data for reports, graphs, analysis or animation. SLAM is a FORTRAN-based simulation language. It supports the three modeling world views in a single, integrated framework. SLAM II is the latest release of SLAM. It permits process-interaction, event-scheduling and continuous modeling perspectives, or any combination of the three. SLAM is the culmination of the family of languages developed by Pritsker and others, starting with GASP, GERT, and Q-GERT. Using TESS requires learning its command language and in addition SLAM II is rather a complicated language to learn.

SIMAN (Pegden [35]) is a general purpose simulation language which allows the adoption of an event-scheduling, process-interaction, or continuous approach, or a mixture of the three approaches. The language includes special features which facilitate the simulation of manufacturing and material handling systems. These special features include statements to simulation conveyors, robots, automated storages and retrieval systems, and manufacturing cells. The IBM-PC version of SIMAN will be augmentable by a menu driven support program which will allow the graphic and interactive construction of a model, rather than statement input. A post processor is available for the IBM-PC to capture the dynamic behavior of a system through the graphical animation of simulation results. SIMAN was developed by C. Dennis Pegden, a faculty member of Pennsylvania State University. The graphical version of SIMAN on IBM-PC (when it becomes available) represents the closest of these systems to our original goals. SIMAN is still rather difficult to learn and does not provide the flexibility of SIMSCRIPT user written subroutines in GSS.

Modelmaster (Graphically Enhanced Factory Modeling System) is a Manufacturing Simulation System developed by the General Electric Company (Duersch [12]). This system provides facilities for modeling manufacturing units, material handling, equipment and robots. It is designed specifically for Manufacturing Engineers or facility planning personnel. The user creates the graphic layout of workcenters and equipment on the video screen, followed by definition of job sequences and transporter paths. The software package then leads the user through a menu-driven question and answer session, based on the preceding graphic input, in a programming-free environment. The simulation model is configured automatically and runs in a manner transparent to the user. Simulation results are available in the form of summary statistics reports, graphs of queue contents and layout animation. This system is limited to manufacturing simulation and does not provide any flexible output generation.

A research activity in the Rensselaer Polytechnical institute, headed by Shin-Miao Chin [10], has developed a front end graphical interface for the GPSS. The block diagrams can be interactively created on a CRT screen and the necessary operand input for each GPSS command block is preceded by a description of the operand (the

user does not need to refer to the user manual). The diagram can be modified on the CRT screen by using interactive devices. The user can change, insert, or delete the components by using a light pen, cursor or keyboard. The capabilities of this system are limited to the GPSS language, which does not provide flexible output generation. It is slow and suffers from other GPSS shortcomings (refer to section 1.3.1 for comments on GPSS).

2 GRAPHICAL SIMULATION SYSTEM

This chapter contains a technical description of the Graphical Simulation System and its components. The Graphical Simulation System provides the user with an Iconic Graphical Simulation language and an interactive user interface. This system allows the user to graphically represent a simulation model, interactively build the model, translate the model into an equivalent SIMSCRIPT model and obtain the simulation run results.

Symbolic icons are the primary means for modeling this simulation system where the shape of each icon represents its function. The symbols are connected together via linear flow graphs which show the movement of entities through the simulation model. The sequencing is depicted by arrows which control the flow of the entities through the entire model. The entities represent "things" such as work-pieces, information or people which flow through the real system.

An overview of the Graphical Simulation System's architecture and its components including interactive user interface, BCSL language compiler and output generator is followed by a description of the Iconic Graphical Simulation language (Block Graphic Symbolic

Language) and the underlying Block-Command Symbolic Language (GPSS-like language). The iconic simulation language is expandable which makes it possible to develop special purpose simulation packages and generate customized output reports.

In this chapter we describe the translation methodology and techniques used in the development of the graphical simulation language. We demonstrate how the graphical representation of a model is translated into the equivalent SIMSCRIPT model and how simulation run results are generated by providing a detailed description of the Block-Command Symbolic language compiler, its components and outputs. Moreover, we present the SIMSCRIPT equivalent of several key blocks, thereby establishing a better understanding of the general translation strategy.

2.1 Overview Of Graphical Simulation System

The Graphical Simulation System consists of three major subsystems: frontend user interface including graphics and alphanumeric input capability, Block-Command Symbolic language compiler, and Output Display Generator (Figure 2-1 shows an overview of the system).

The user creates a graphical image of the simulation model, through the user interface of the Graphical Simulation System, which will be stored in "Block Graphic Files". In order to develop the graphic representation of the model the user will use a graphics terminal, joy stick, light pen (or cursor monitor) and a set of function keys.

The user interface (GRAPE) is a menu driven, user friendly tool, which allows the user to build the simulation models and interact with the rest of the simulation system. The user interface supports and guides the user in every step of development, compilation and execution.

The graphical and alphanumeric data representing the simulation model, including block-flow structure, relationship between building blocks and characteristics

of each block is converted to the Block-Command Symbolic language. The BCSL compiler then generates executable machine code to simulate the model behavior. This system is based on a general purpose simulation programming language called "Block-Command Symbolic Language". The Block-Command Symbolic Language is powerful, easy to use and expandable. This language allows the user to tailor fit the simulation system towards his own application by creating new special purpose Block-Commands.

The output generator executes the machine code, gathers required statistics, and later, generates and saves a hard copy of the simulation run. It is also possible to graphically display the animated state of the model as the simulation progresses using the output display generator.

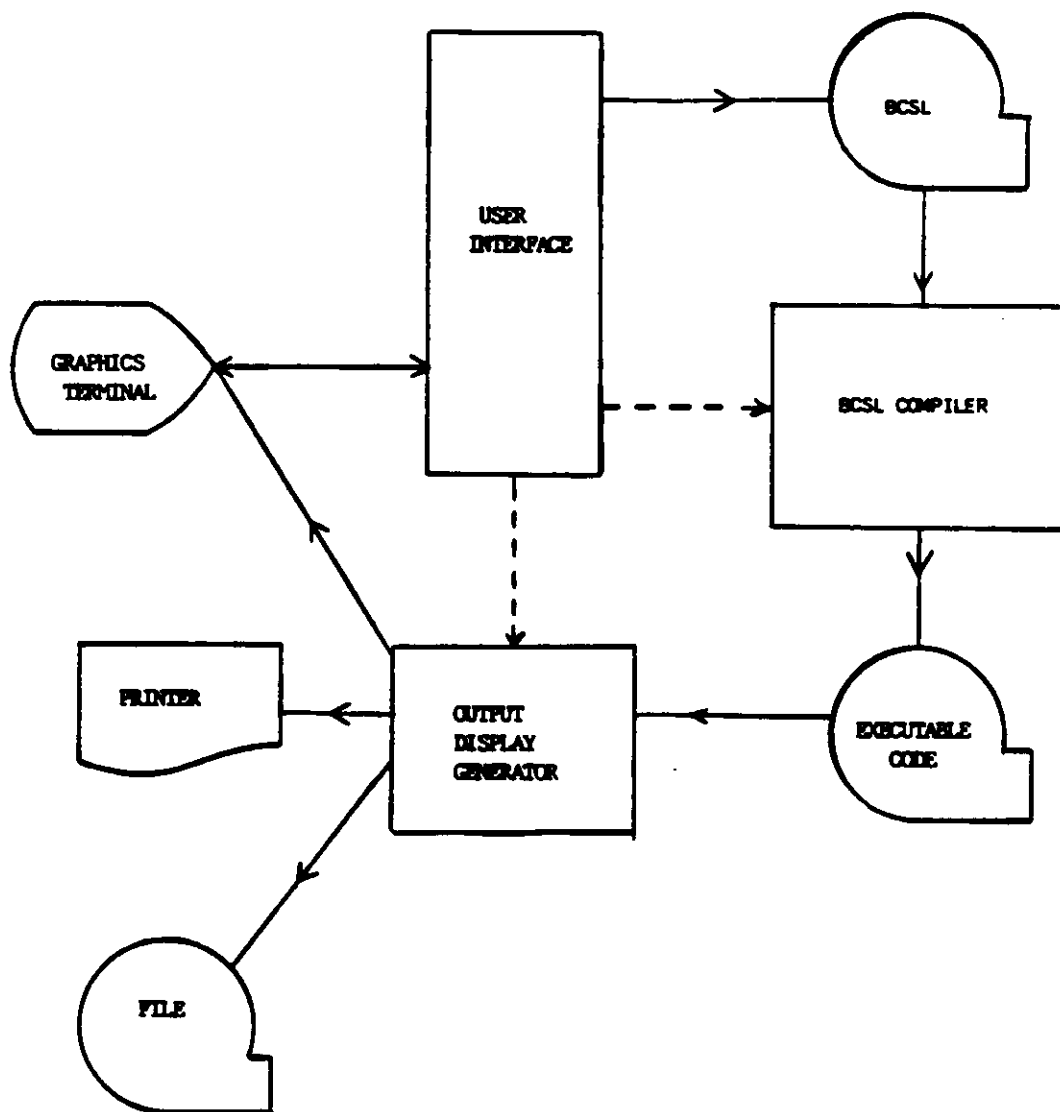


Figure 2-1. Overview of Graphical Simulation System

2.1.1 User Interface - The User Interface for the Graphical Simulation System (GRAPE) contains five basic building blocks; alphanumeric interface, graphic support, system utilities, BCSL generator, and help facility. Figure 2-2 shows an overview of the user interface architecture.

The alphanumeric interface is primarily incharge of all the non-graphic I/O processing. These include driving the menu, processing the user commands and data (from key board, light pen or joystick), and scheduling activities within the system. The alphanumeric interface interacts with other parts of the system, starts their activities and displays their messages on the screen.

The graphic support is in charge of building, modifying, displaying, and verifying the graphic model. The graphic support is built on top of the GDDM package which provides an interface for different IBM terminals (see Appendix D for further environmental information). The graphic support develops an online data base representing the model. This data base contains information about each block, their parameters and their connection path . The graphic representation of each block is saved in an ICON data base which is loaded to

the on-line data base.

The on line data base consists of two doubly linked list structures. Each SYMDATA structure represents a block, and its parameters, where the shape of the block is selected by an index pointing to the ICON data base. Each LINDATA structure represents a connection line between two blocks.

The system utilities can display or print any file generated by GSS. These include the intermediate language (BCSL) translation of the model, final translation (SIMSCRIPT equivalent) of the model, the simulation run results, and the compilers error messages. In addition, a plot of the graphic model is available by using the laser printer. The system utility loads the online data base from the disk, and saves the online data base in the disk resident files(The disk resident data base also includes the header file, representing the general model characteristics).

The help facility in cooperation with the alphanumeric interface assists the user in every stage of the model development. Every function in the user interface has a corresponding information file that the help facility displays upon the users request. The BCSL

generator is developed as part of the user interface and it is described in more details within the BCSL compiler section.

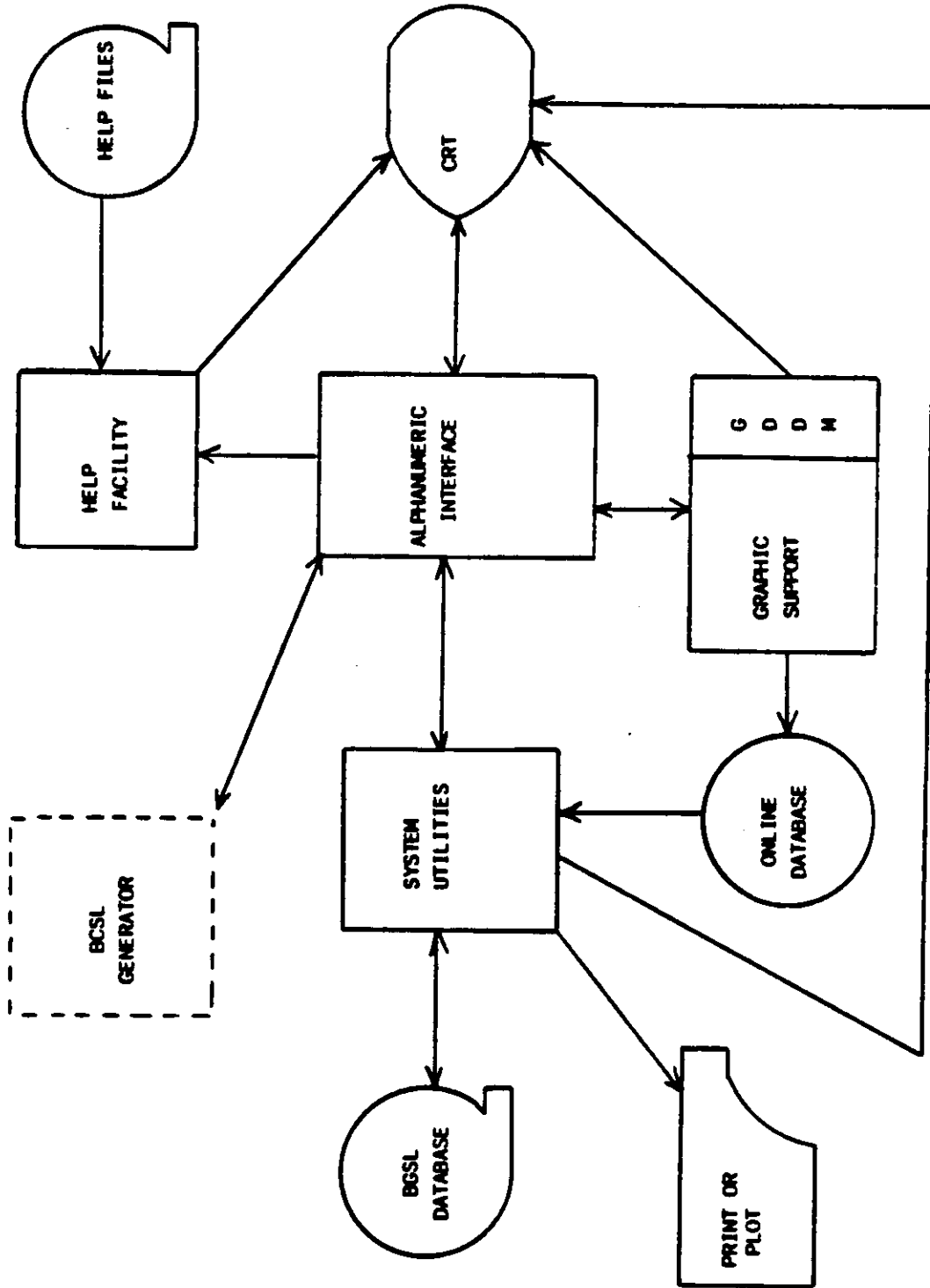


Figure 2-2. Overview of User Interface

2.1.2 Block-Command Symbolic Language Compiler - The Block-Command Symbolic Language Compiler (SIMSCRIPT Translator) consists of three components; BCSL Compiler Pass 1, BCSL Compiler Pass 2 and SIMSCRIPT Compiler. Figure 2-3 shows an overview of the Block-Command Compiler. The Block-Command Symbolic Code Generator receives the Block-Graphic Symbolic data from the "Block Flow Graph File", the "Data String File" and the "Block Connection File". From this data the BCSL code generator constructs the "Block-Command Symbolic File". The Block Command Symbolic Code generator merges the graphical and alphanumerical information, and checks for semantics, syntax and typographical errors.

There is a one-to-one relationship between each line of the "Block-Command Symbolic" Code and the blocks in the "Block Flow Graph". The BCSL compiler (SIMSCRIPT translator) generates the equivalent SIMSCRIPT code for each line of "Block-Command Symbolic" language. The BCSL compiler consists of two passes. Pass 1 generates SIMSCRIPT processes and routines for each block in the "Block-Command Symbolic" Code.

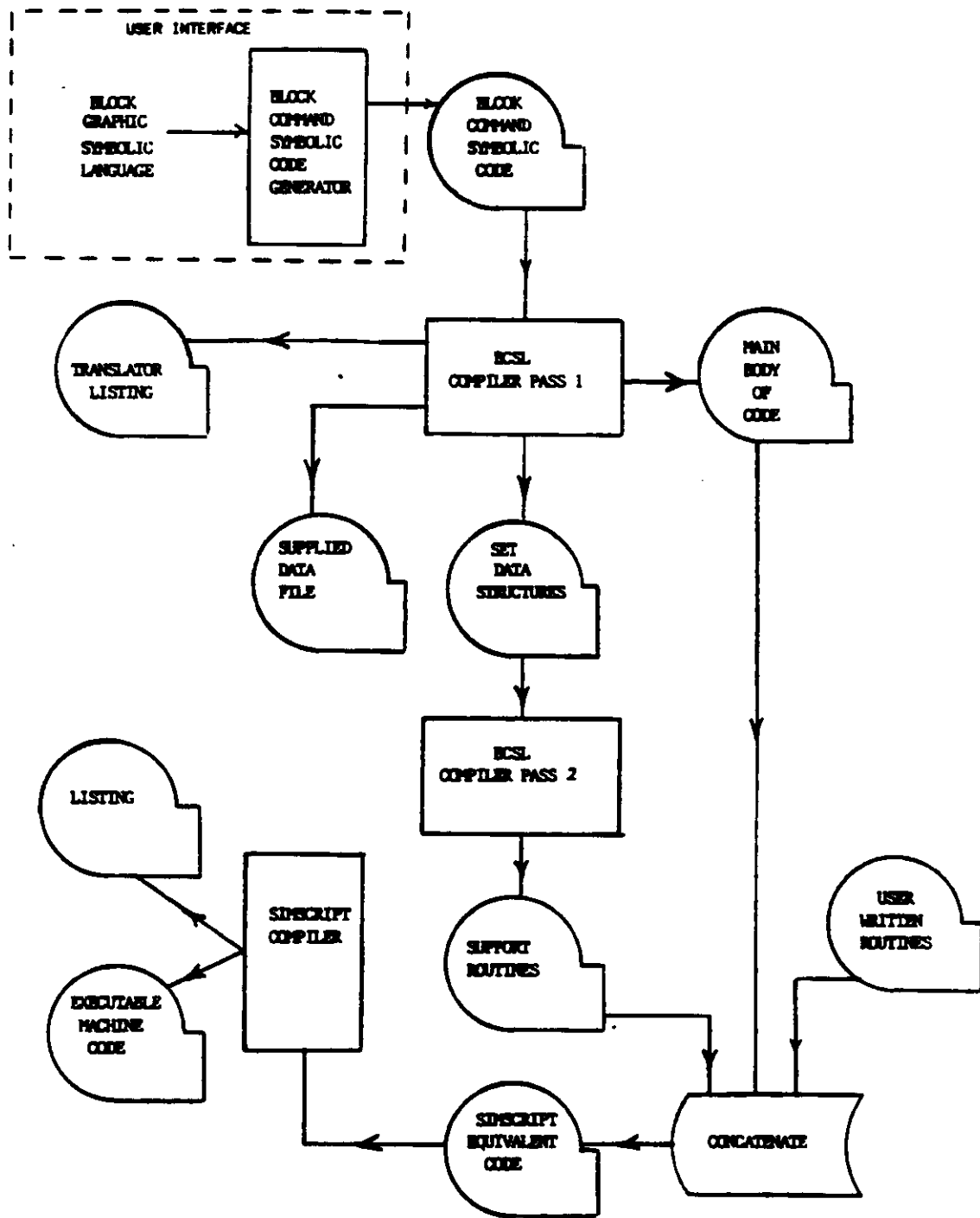


Figure 2-3. Block Command Symbolic Language Compiler

Considering the fact that SIMSCRIPT consists of several sections (i.e., preamble, main, processes and routines), each block-command affects several sections of SIMSCRIPT depending on the type of block. Pass 2 of the BCSL compiler generates the main, preamble, output generator, initialization and monitor routines.

Finally, all the SIMSCRIPT routines generated by Pass 1 and Pass 2 in addition to user supplied routines are concatenated in one file and passed to the SIMSCRIPT compiler. The SIMSCRIPT compiler uses this file to generate machine executable code which will simulate the model behavior in real time. Figure 2-4 shows the transformation of data in the Graphical Simulation System.

The BCSL compiler provides the system with SIMSCRIPT output generation capabilities, both as default and as user requested (tailor-made) outputs. The compiler also allows non-standard blocks to be added to the graphical language which creates customized special purpose simulation languages.

In addition, it is possible to generate the "Block Command Symbolic File" using editors or modify an existing one generated from Block Graphic Symbolic Language. Figure 2-5 displays a modification to the

compiler architecture of Figure 2-3, reflecting the user written BCSL input.

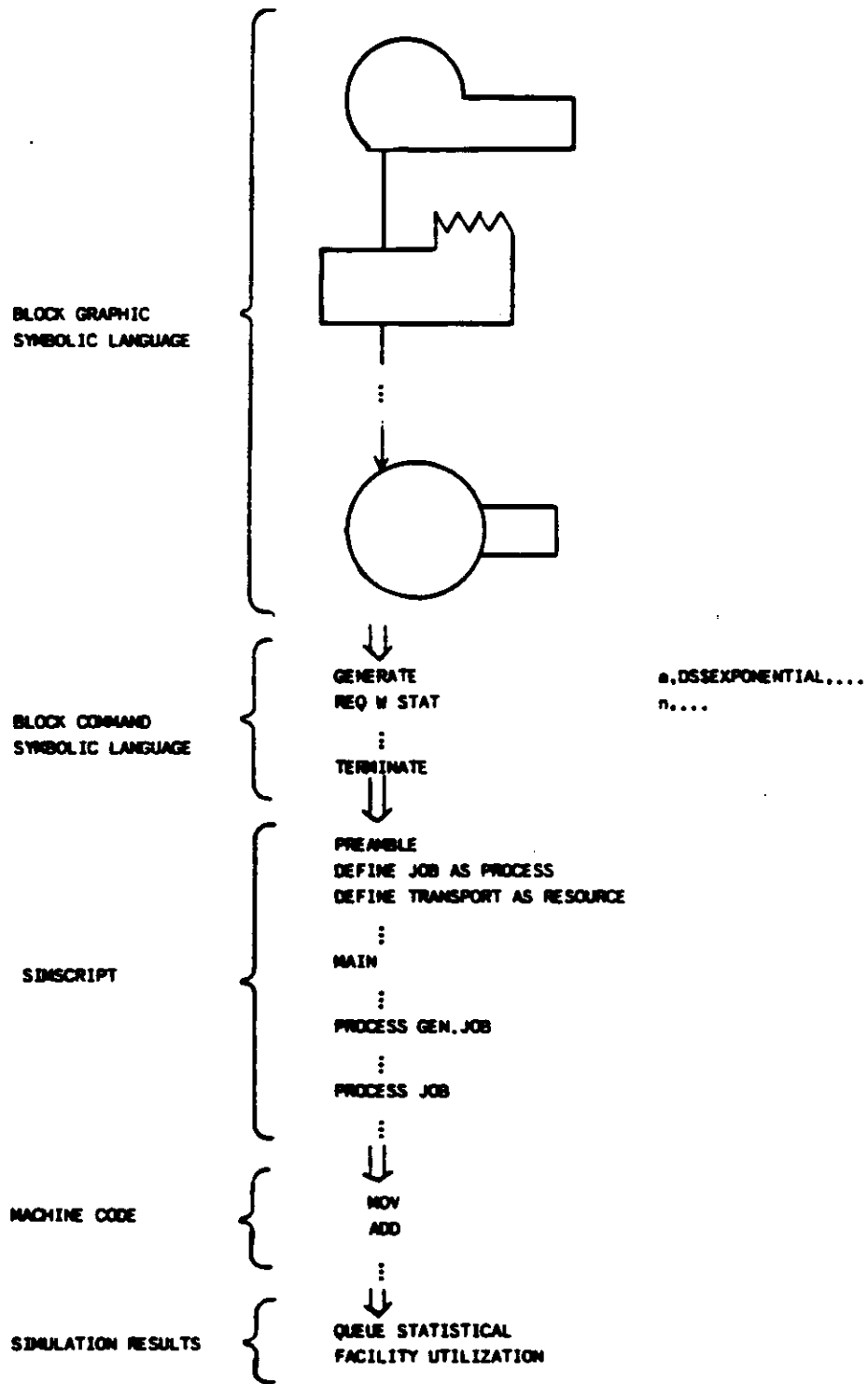


Figure 2-4. Data Flow in Graphical Simulation System

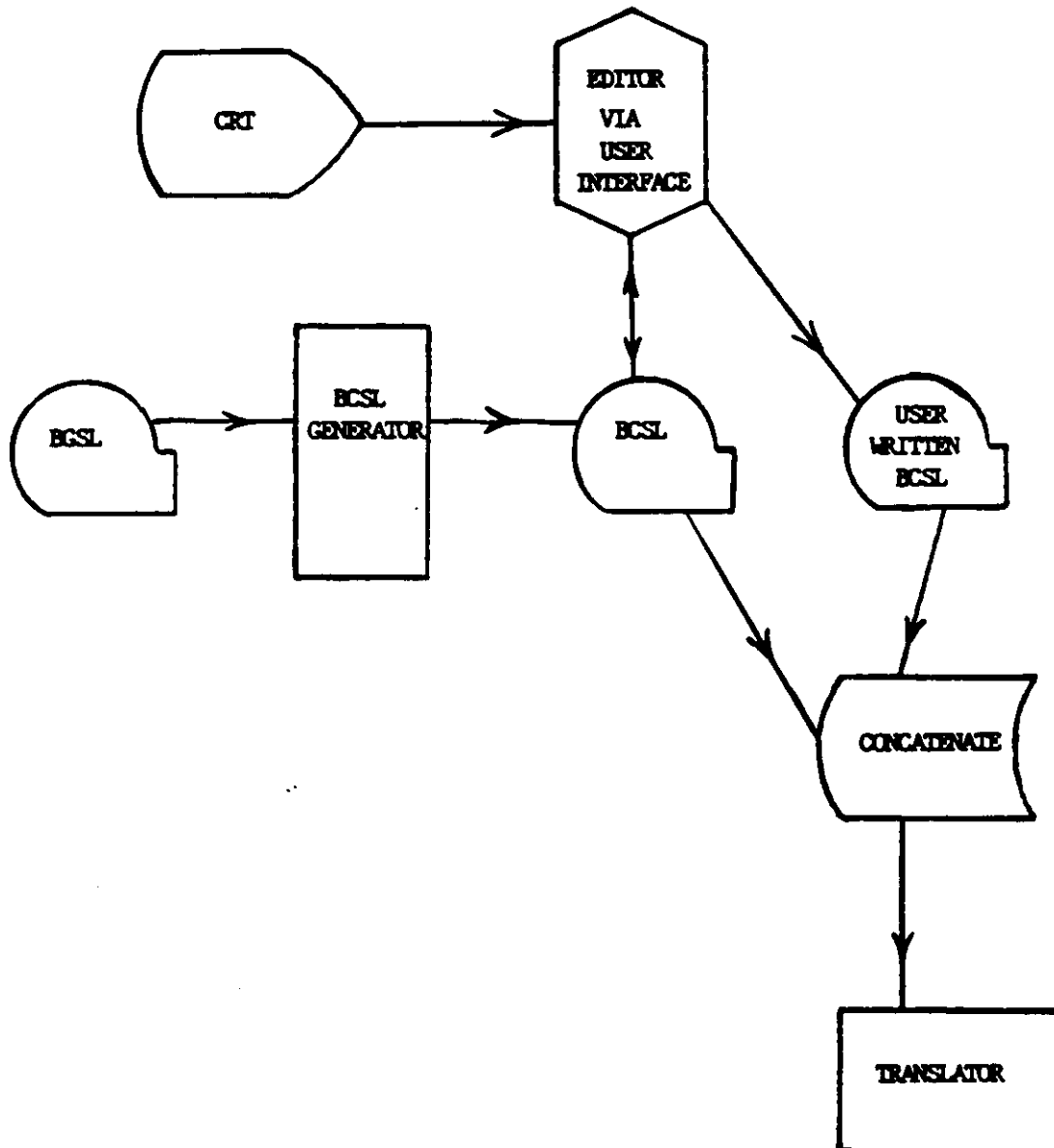


Figure 2-5. Adding user written BCSL

2.1.2.1 Pass 1 Of The BCSL Compiler - Pass 1 of the BCSL compiler reads one line of "Block Command Symbolic" code at a time. It parses each line by defining operation, operands and label, then it generates a SIMSCRIPT equivalent of the block. In addition pass 1 creates and adds new elements to appropriate set data structures. These elements contain the characteristics of each block in the model and will be used by pass 2 of translator. Usually each block translates into predetermined "raw" SIMSCRIPT code. The raw SIMSCRIPT code is converted to executable SIMSCRIPT code by the raw text processor. The raw text processor is the central part of the SIMSCRIPT code generator. It converts the raw text by replacing the undefined primitives by their substitutes in the block operands.

2.1.2.2 Pass 2 Of The BCSL Compiler - During the processing of each block in Pass 1 of translator, several tables (set data structures) are prepared so that each block may cause several elements to be created and added to appropriate sets. These sets constitute the basis for Pass 2 of the translator. They save the characteristics of the processed block, the needed variables, statistic counters and required output definitions. Pass 2 processes all the set structures

prepared by Pass 1 and generates the definition statements in the preamble section, initialization commands in the initialize routine, output commands in the output generation routines and monitored routines for extra statistical analysis and flow-control of processes.

2.2 Block Graphic Symbolic Language

The Block Graphic Symbolic Language (BGSL) is a general purpose simulation language which uses graphic images to interact with the user. Block diagrams are the primary means for modeling discrete systems in this simulation language. These diagrams are linear top-down flow graphs which show the movement of entities through the simulated system. The shape of the individual blocks indicates their function. The sequence of events is represented by arrows which control the flow of entities from block to block through the entire diagram.

These entities are used to represent work-pieces, information or people which flow through the real system. Each entity may be individualized by assigning attributes to describe or characterize it. For example, an entity representing a work-piece might have attributes corresponding to the due date and processing times. As the entities flow from block to block, they may be delayed, destroyed, combined with other entities, etc., as determined by the function of each block. Figure 2-6 displays the silhouette of a typical "Block Graphic Symbolic Language" program.

BGSL is considered a very high level programming language, which isolates the programmer from many syntactical barriers and limitations. BGSL is an integral part of the Graphical Simulation System. This system interactively guides the user through each step of the construction via selection menus, help messages, guiding messages (what to do next) and error messages.

BGSL is an expandable language which makes it possible to define new blocks for the language as the need arises. This allows the language to grow and better meet the user requirements as time passes. Using this capability, new blocks have been added for simulation of Flexible Manufacturing Systems. Simplicity of model generation and system support throughout the model building and verification allow a non-sophisticated user (without programming background) to generate rather complicated simulation models within this graphical simulation system.

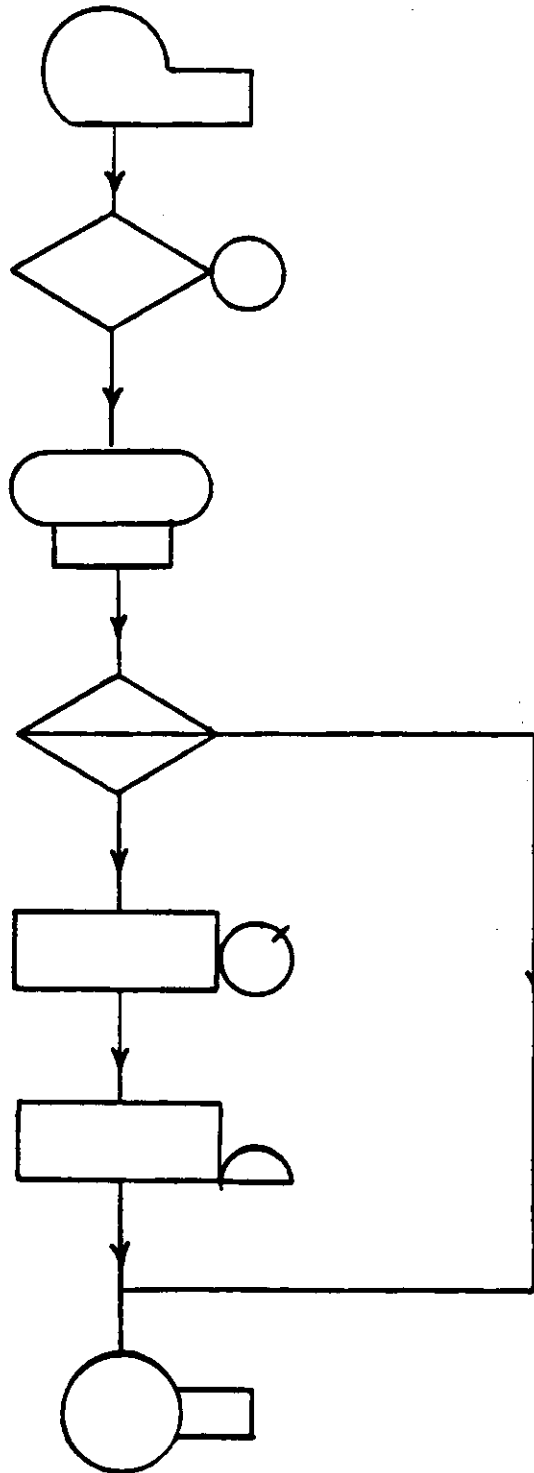


Figure 2-6. Silhouette of a Typical Block Command Symbolic Language

2.2.1 BGS L Elements And Programming Concept - The Block Graphic Symbolic Language consists of three major elements; Graphical Command blocks, block operands (attributes) and connection lines. A graphic terminal provides the user with a window to a two dimensional "Graphic domain". The user can move around this domain, zoom in, zoom out and select any element to add, delete or modify.

In this language the user builds a simulation model by selecting the blocks, adding them to the graphic domain, defining their attributes and connecting them together. Once the user starts building a model (selects "BUILD MODEL" from the menu), an option list will be displayed (Figure 2-7) which allows the user to add, delete, modify and search for any blocks. When the user selects ADD BLOCK, a list of available blocks is displayed (Figure 2-8) from which one block may be selected. Then the user will be asked to select a location in the graphic domain into which to insert the selected block. Later the operand query menu containing the name and description of all needed attributes for that block is displayed and the user will be asked to provide data for each item. Figure 2-9 shows the query menu displayed for the "GENERATE" block, and Figure 2-10 for the "ADVANCE" block.

USE THE LIGHT PEN TO MAKE A SELECTION FROM THE MENU:

CURRENT MENU PATH: /BUILD MODEL

ADD BLOCK	LOCATE BLOCK
CHANGE BLOCK INFO	MOVE BLOCK DOWN
CONNECT BLOCKS	MOVE BLOCK UP
DELETE BLOCK	ZOOM IN
DELETE LINE	ZOOP OUT
LABEL BLOCK	

PKF: 1 = HELP 3 = CANCEL/EXIT
 7 = SCROLL UP 8 = SCROLL DOWN
 10 = SCROLL LEFT 11 = SCROLL RIGHT

Figure 2-7. "Build Model" Selection Menu Display

USE THE LIGHT PEN TO MAKE A SELECTION FROM THE MENU:

CURRENT MENU PATH: /BUILD MODEL/ADD BLOCK

ADVANCE	GATESRG	PRIORITY	START
ASSEMBLE	GATHER	QTABLE	STORAGE
ASSIGN	GENERATE	QUEUE	STORAGES
BIVARIABLE	INITIAL	RELEASE	TABLE
BUFFER	LEAVE	REALLOCATE	TABULATE
CLEAR	LINK	RETURN	TERMINATE
DEPART	LOGIC	RMUT	TEST
ENTER	LOOP	SAVEVALUE	TRANSFERC
EQU	MARK	SEGMENT	TRANSFERS
FUNCTION	MATCH	SEIZE	TRANSFERU
FVARIABLE	MATRIX	SELECTLOG	UNLINK
GATELOG	MSAVEVALUE	SELECTMM	VARIABLE
GATELOC	PREEMPT	SELECTREL	WILDCARD
GATEFAC	PRINT	SPLIT	

PFK: 1 = HELP 3 = CANCEL /EXIT
 7 = SCROLL UP 8 = SCROLL DOWN 10 = SCROLL LEFT
 11 = SCROLL RIGHT

Figure 2-8. "Add Block" Selection Menu Display

SUPPLY THE FOLLOWING INFORMATION FOR THE COMMAND SELECTED,
THEN PRESS ENTER
GENERATE'S REQUIRED PARAMETERS

MEAN TIME	==>
SPREAD OR FNC MODIFIER	==>
OFFSET INTERVAL	==>
LIMIT COUNT	==>
PRIORITY LEVEL	==>
NO. OF PARAMETERS	==>
TYPE OF PARAMETERS	==>
TRANSACTION NAME	==>
2ND DISTRIBUTION FNC PARAMETER	==>

PFK: 1 - HELP

3 - CANCEL/EXIT

Figure 2-9. GENERATE Block Parameter Query Display

SUPPLY THE FOLLOWING INFORMATION FOR THE COMMAND SELECTED,
THEN PRESS ENTER
ADVANCE'S REQUIRED PARAMETERS

MEAN TIME

==>

SPREAD OR FUNCTION MODIFIER

==>

SECOND FUNCTION PARAMETER

==>

PFK: 1 - HELP

3 - CANCEL/EXIT

Figure 2-10. ADVANCE Block Parameter Query Display

Typically, the last step in building a model is to connect the blocks. Each connection line needs a source block and destination block. Each block can be entered only from the top and exited from the sides or bottom. Several entry lines can be connected to the top of each block while no more than three exit lines are possible; one from each of the right, left or bottom sides. The user can choose to connect several blocks in one step which speeds up building the model.

Each added block is assigned a default label except when the user decides to define his own label. In the latter case, the user can select to label any block or change the label on any block.

Models in the Block Graphic Symbolic Language are divided into several segments. There are three segment types; definition segments, control segments and model segments. Each model can have several definition, control or normal segments, even though usually there is only one definition and control segment and several normal segments in a model.

The order of blocks in the model segment represents the order in which events occur for each transaction and the flow of the transactions in the model. Typically, each GENERATE block requires a normal segment block

connected to it so that the connection source is the segment block and destination is the GENERATE block. This block is called "father of GENERATE" block. Similarly the next block connected to the GENERATE block is its "son". Each block in the model must have a father except the segment blocks. All the blocks graphically connected to each other and to a segment block constitute a segment (they all have the same ancestors).

Figure 2-11 shows a model with three segments. Notice that the normal segment block is the father of the GENERATE block and the GENERATE block is the father of the SEIZE block.

There could be any number of termination blocks in each normal segment. Figure 2-12 shows a segment with two terminate blocks.

Figure 2-13 illustrates that two or more segments can share the same terminate block; However this modeling technique is not recommended as it may lead to confusion.

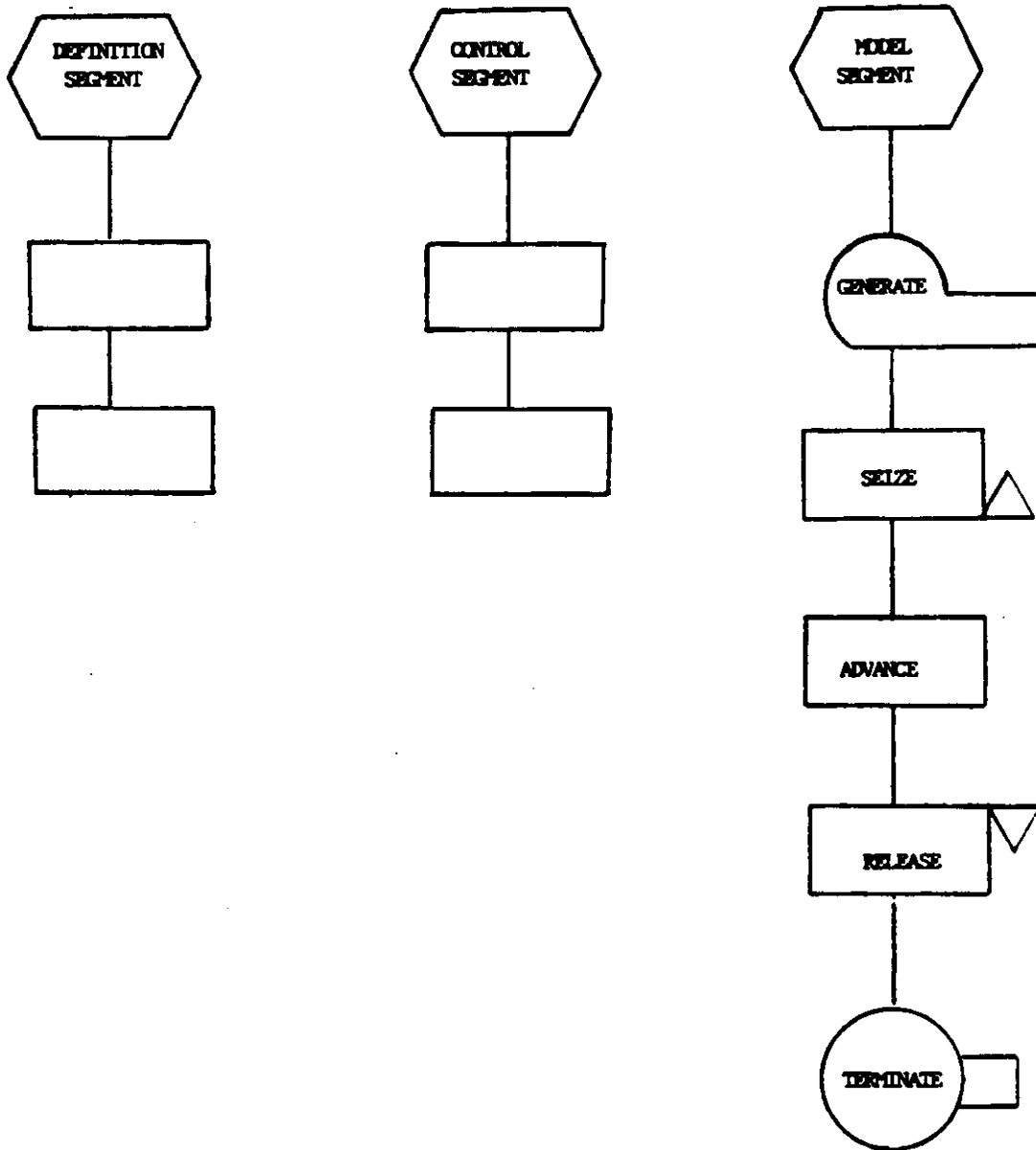


Figure 2-11. Model Segments

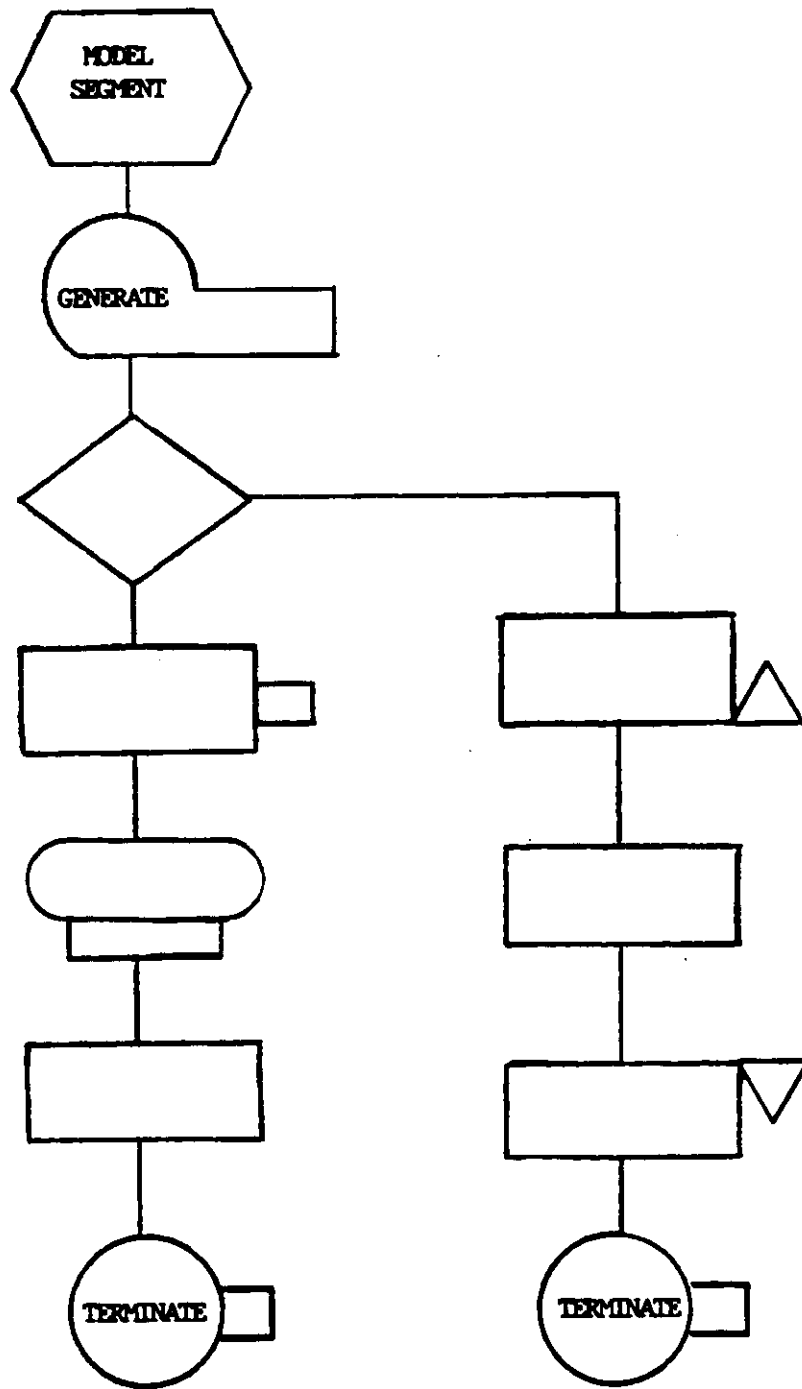


Figure 2-12. Segment with Two TERMINATE Blocks

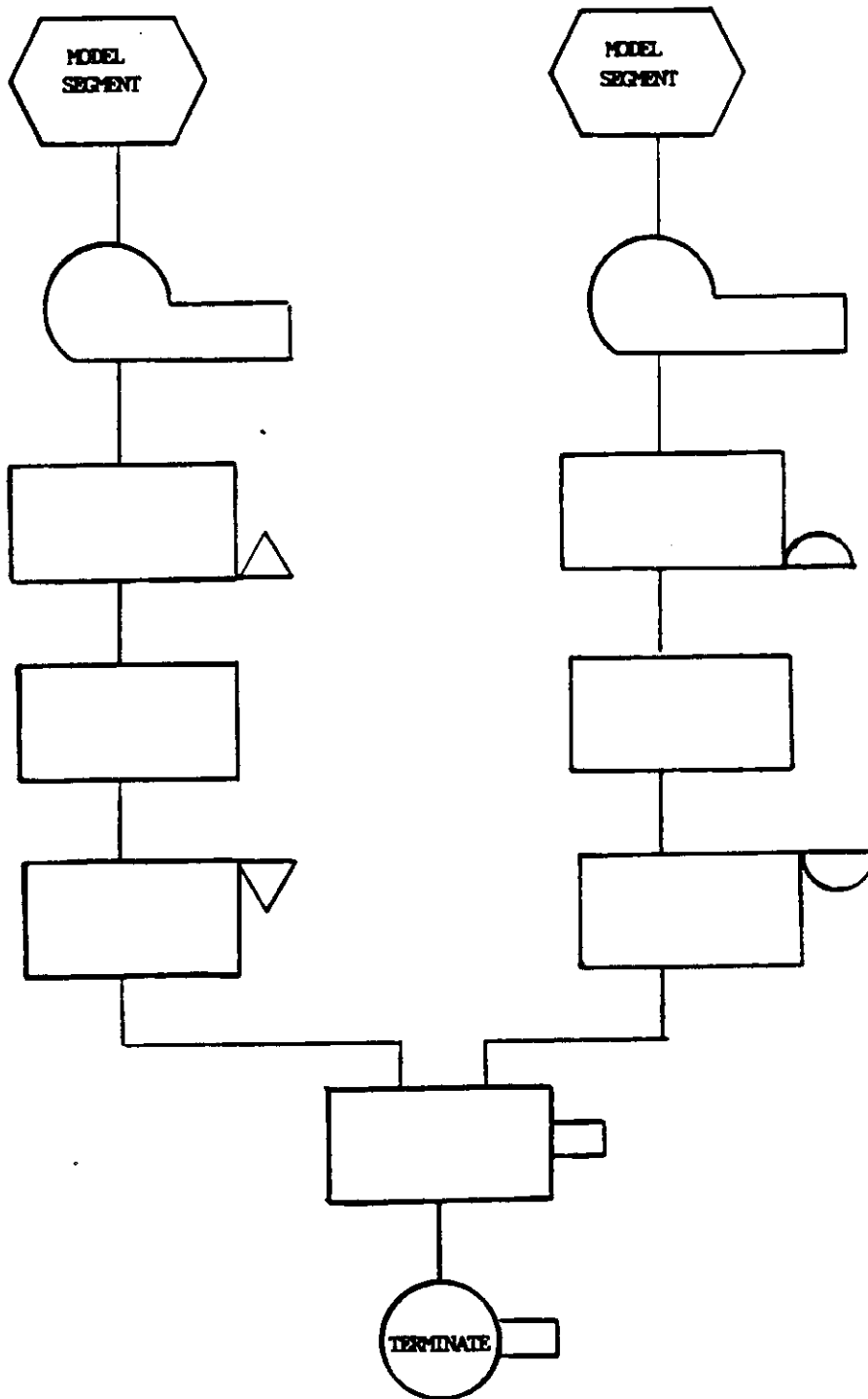


Figure 2-13. Data Flow in Graphical Simulation System

Definition segments are used to define and initialize distribution functions, tables, matrices, variables, storage, transporters and work stations. Control segments are used to start the simulation, reset statistics and rerun the model. Even though blocks in the definition and control segments are connected together, this does not imply any entity flow nor require any special order for blocks. Each block can only belong to one father even though it could belong to different segments. Father-son relationships in a BGS� model are used to define the order in which Block Command Symbolic Language statements are organized.

Usually, translation of a block in BCSL follows the translation of its father block. The only exception is when a block has more than one exit line (this block has several sons). In this case one of the son blocks is selected to follow the father's block translation in the main line of code (main branch), and the other sons are selected to START new sub-branches. Each sub-branch is labeled and it is connected to the main branch via a Transfer Command to its label. Figure 2-14 shows how a father block and its multiple sons are translated into BCSL. In this figure block "F" has one father (GF) and three sons (S1, S2, and S3). Each son has a grandson GS1, GS2 and GS3 each following their father's block

translation. Labels L1 and L2 start the sub-branches for each additional son. These labels are referenced by the father's block attributes.

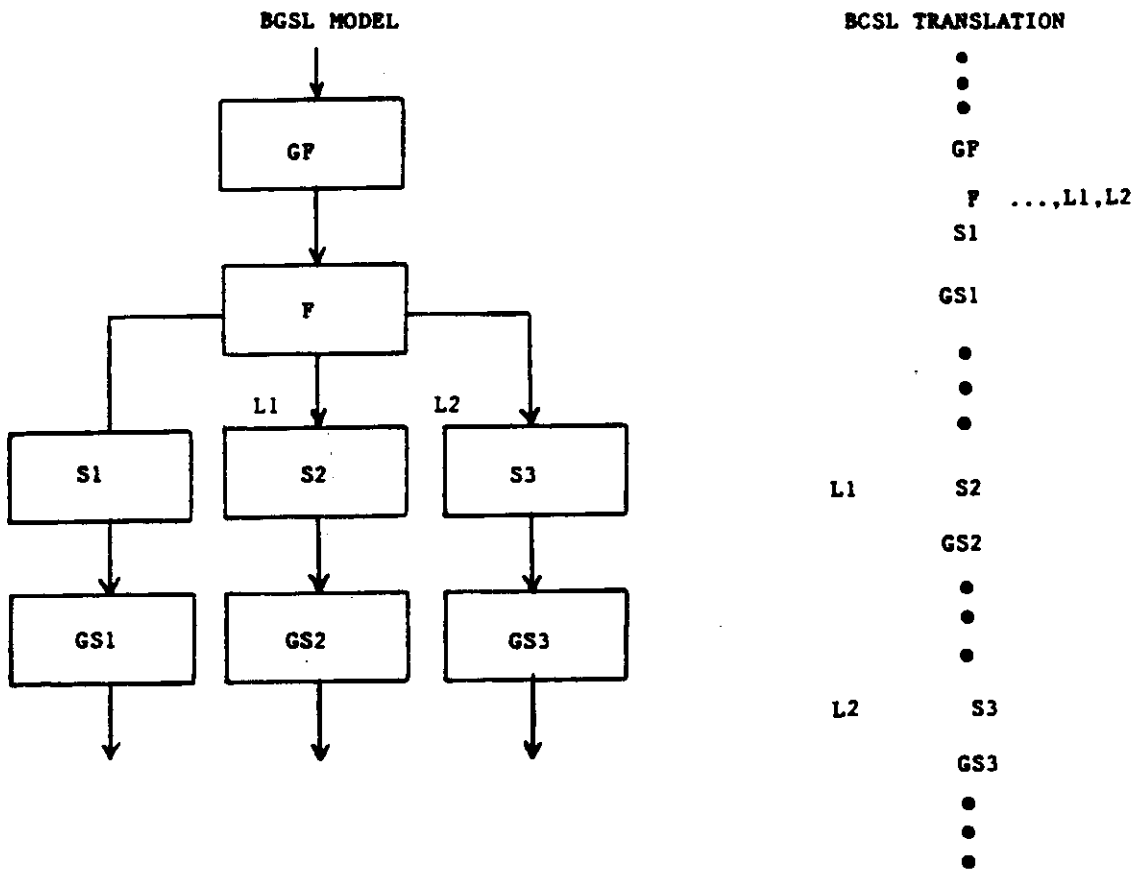


Figure 2-14. A BGS Model and its BCSL Translation

2.2.2 Block Command Symbolic Language -

The Block Graph Symbolic Language is based on an alphanumeric language called "Block Command Symbolic Language" (BCSL). BCSL is similar to and upward compatible with the GPSSV language. Usually each block in BGS� translates into a line of BCSL, where the attributes of the block provide the command operands. The format of BCSL is as follows;

Label Operation Operand1, Operand2, ...Operand8.

The order in which the commands will be located in BCSL are determined by the connectivity and topography of the graphic language and segments in which the blocks appear. The user can select BCSL as input to the system, representing a simulation model. This is done by using an editor and generating a file containing BCSL code (GPSS-like language).

There is a one-to-one relationship between the BGS� blocks and the BCSL commands. The same applies to the attributes of each block. Operands used in BCSL and BGS� generally follow the rules and conventions of GPSS operands. Except that, any algebraic statement is acceptable where GPSS only accepts integer values.

Standard GPSS numerical attributes are also available to the user.

Each statement in BCSL is represented by a block in BGSL, even for statements used in definition and control segments. (These statements are referred to as "non-block" statements in GPSS.) All of the blocks in the definition and control segments have the same pictorial representation in BGSL (a rectangular shape).

Appendices A and B contain a summary of all the commands in BCSL which have been implemented in the current state of the Graphical Simulation system. A picture of each block, name and attributes needed for it are provided and a list of standard numerical attributes are included in Appendix C.

A detailed description of GPSS compatible blocks can be found in Schriber [41]. New blocks are defined in "Advanced Features" section and the next section discusses the differences between the BCSL and GPSSV standard blocks.

In the remainder of this thesis we will use BGSL Blocks and BCSL Commands interchangeably and will not distinguish between them except where there are implementation differences.

2.3 Methodology Of Translation To SIMSCRIPT

The process oriented simulation world view for modeling of discrete systems has been used as a basis for development of SIMSCRIPT equivalent code. The SIMSCRIPT Translator provides the Block Command Symbolic Language with features that represent the dynamic and static objects of a system. The entities that represent dynamic objects of a system are called transactions. The static objects of a system are currently represented by several blocks; FACILITY, STORAGE, workstation and transporter. The FACILITY and transporter usually simulate a single resource object and the STORAGE and workstation simulate multiple-identical resource objects. During the execution of a BCSL Simulation model, the transactions move through the model (simulation system) from block to block. As they move, they interact with the static entities or resources of the system represented by various blocks. The main functions of most of these blocks are to create and destroy transactions, to alter their routings, and to delay their movements according to the logic of the model. The other blocks including control and definition blocks are provided to support the simulation features that are needed for building a simulation model and for performing simulation experiments.

The equivalent SIMSCRIPT program for each BCSL model is broken down into several standard and non standard sections. These include; preamble, main, initialization, main output generator, dedicated output generators, transaction generator processes, transaction processes, left monitored routines and user created routines. The generated SIMSCRIPT equivalent program is modular and well structured which facilitates future expansion of the source languages (BGS L and BCSL). These features also make the debugging of the SIMSCRIPT model easier by providing better traceability and readability for the generated code.

The basic idea in translation of BCSL commands to SIMSCRIPT language is based on the fact that each block is translated into a predefined set of SIMSCRIPT statements. The attributes of each block will supply the input data for their corresponding variables and parameters in SIMSCRIPT equivalent program.

Translation of the GENERATE block plays an important role in the development of the translation methodology for Block Command Symbolic language into SIMSCRIPT. The GENERATE block represents the mechanism for introducing arrivals of new transactions into the system from the external world. In other words, it is

considered a source of transactions. Each GENERATE block translates into SIMSCRIPT as a "Transaction Generator Process", which is in charge of creating and activating certain types of transactions and introducing them into the modeled system. This process can be characterized by describing the number of objects that arrive and specifying the time between arrivals.

The GENERATE block translation also initiates corresponding "Transaction Process" in generated SIMSCRIPT equivalent programs. A Transaction process is a representation of what happens to the transaction within the modeled system after it is introduced to the system by the Transaction generator. A process may be thought of as a sequence of interrelated events separated by lapses of time, either predetermined or indefinite.

In Block Graphic Symbolic Language the chain of blocks connected to a GENERATE block in the same segment, represents what happens to the transaction within the modeled system. Each block in the model segment represents an action, event or check point happening to the transaction. This block in turn translates into a section of the transaction process initiated by the GENERATE block in the same segment.

The flow of transactions can be nondeterministically suspended or permanently terminated as the transactions pass through segment blocks. The TEST and GATE blocks under certain conditions can cause a transaction process to suspend itself. These blocks will cause the generation of left-monitored routines. The left-monitored routine will resume the transaction process when the proper conditions are met.

The TERMINATE block represents a sink for the transactions where transactions finally leave the modelled system and disappears. This block translates into the End Statement for the transaction process which causes the transaction temporary entity to be deleted from the model database.

Figure 2-15 shows how a typical normal segment in Block Graphic Symbolic language translates into SIMSCRIPT statements.

In order to generate the preamble, initialization, output generation and left-monitored routines, a list of processes, resources, variables, and required statistical analysis is needed. Utilizing set data structures in SIMSCRIPT, the BCSL Compiler classifies the BCSL block characteristics into predetermined data structure elements. The BCSL compiler files each of

these elements into their corresponding sets (lists), including the set of processes (transactions and transaction generators), resources (facilities, storages, working stations, and transporters), queues, statistical variables, tabulated variables and left-monitored variables.

These sets are prepared while the BCSL compiler translates each block and generates their equivalent SIMSCRIPT statements for the transaction processes.

Analyzing each object in a model usually requires several variables to be monitored. Monitoring of these variables is accomplished either by adding monitor statements within the transaction process or utilizing the special preamble statements provided in SIMSCRIPT. Examples of translation of selected blocks are included in Section 2.4.1 (Description of Key Block Processor Routines).

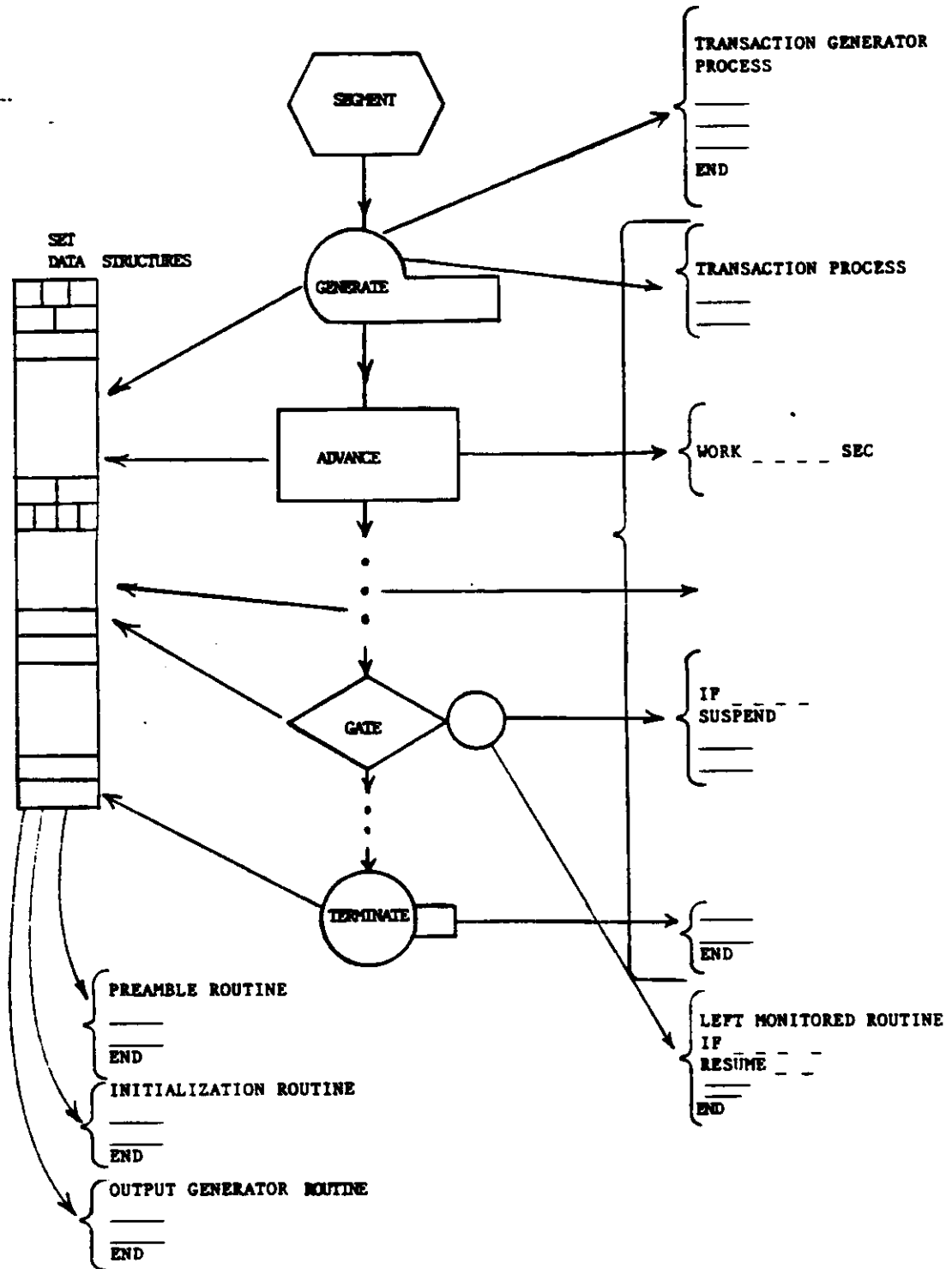


Figure 2-15. Translation of a Model Segment

2.4 Detail Description Of The BCSL Compiler Architecture

SIMSCRIPT has been selected as the language in which the BCSL compiler (SIMSCRIPT translator) is written. Availability of facilities needed for a compiler, like text processing, list processing and dynamic memory allocation have made it desirable to select SIMSCRIPT for development of the compiler. In addition, selection of the SIMSCRIPT language as a compiler eliminates an additional compiler for the translator code itself (e.g., if FORTRAN had been selected to write the BCSL compiler, a FORTRAN compiler would have been needed in addition to a SIMSCRIPT compiler). Therefore, in order to build simulation models in BCSL, the only compiler needed is SIMSCRIPT. This feature makes the BCSL compiler easily portable to all the computers which support SIMSCRIPT.

Given that the frontend user interface section of the Graphics Simulation System is written in PL/1, it is necessary to have both PL/1 and SIMSCRIPT compilers if the user wants to transfer BCSL to a new computer.

Structured programming and modular coding techniques have been used in addition to standard SIMSCRIPT programming conventions in the development of the BCSL compiler. The compiler includes the standard preamble section which contains all the needed data structures and corresponding data sets. The main section initializes the translator parameters. The Main section later calls Pass 1 of the compiler, calls Pass 2 of compiler, and finally returns the execution status of the compilation back to user interface.

The first pass of the translator opens the BCSL input file, the first pass output, the data file and the listing file. It then reads each line of BCSL code and copies it into an output listing file. Pass 1 also outputs the same input line as a SIMSCRIPT comment line into "pass 1" output file for debugging and documentation purposes. Given that each BCSL line carries the original block number of the corresponding block in BCSL, it makes it possible to trace SIMSCRIPT output code back to BCSL blocks.

While reading each line of BCSL, the first pass of the compiler parses the BCSL input line into table, operation and operand elements and calls the operation processor routine. The compiler' "Pass 1" finally

closes all the files and exits when the last line of the BCSL input file is reached.

The operation processor checks the validity of the operation command and calls the appropriate block processor routine. A typical block processor routine contains the equivalent raw SIMSCRIPT code embedded in the routine. For each line of raw SIMSCRIPT code, it calls the "Raw Text Processor" routine to generate and output executable SIMSCRIPT code.

The raw text processor routine parses the lines of raw SIMSCRIPT code and generates a token tree. It then replaces the primitive elements in the tree by their corresponding operands. Finally the raw text processor reassembles the executable SIMSCRIPT code by reversing the processed tree and writing it into the Pass 1 output file and the listing file.

The raw SIMSCRIPT text is a valid SIMSCRIPT statement which contains several unknowns or primitive elements. The primitive elements usually either refer to an operand of the original block or are derived from them. Each primitive element starts and ends with a colon and contains an integer number which represents an operand index number.

Notice that Pass 1 of the compiler originally parses the command-block and generates an array of operands. The block processor routine can generate additional operand array elements using existing elements in the array. Figure 2-16 illustrates how primitive elements and raw text are used to generate executable SIMSCRIPT statements. In this example, the primitive elements :1: and :2: are replaced by their corresponding operands var and integer.

The block processor routine creates the necessary Pass 2 data structure elements, defines their attributes and adds them to the proper sets as needed. It also generates the data file which are used during execution of generated SIMSCRIPT code as the input file.

The Second Pass of the compiler also follows the modular programming techniques by opening the Pass 2 output file and calling a series of special purpose routines. Each routine is responsible for creating certain section of the target SIMSCRIPT program. These routines include; preamble generator, main generator, initialization routine generator, output routine generator and left-monitored routine generator. The second compiler pass finally closes all the files and returns to the Main routine.

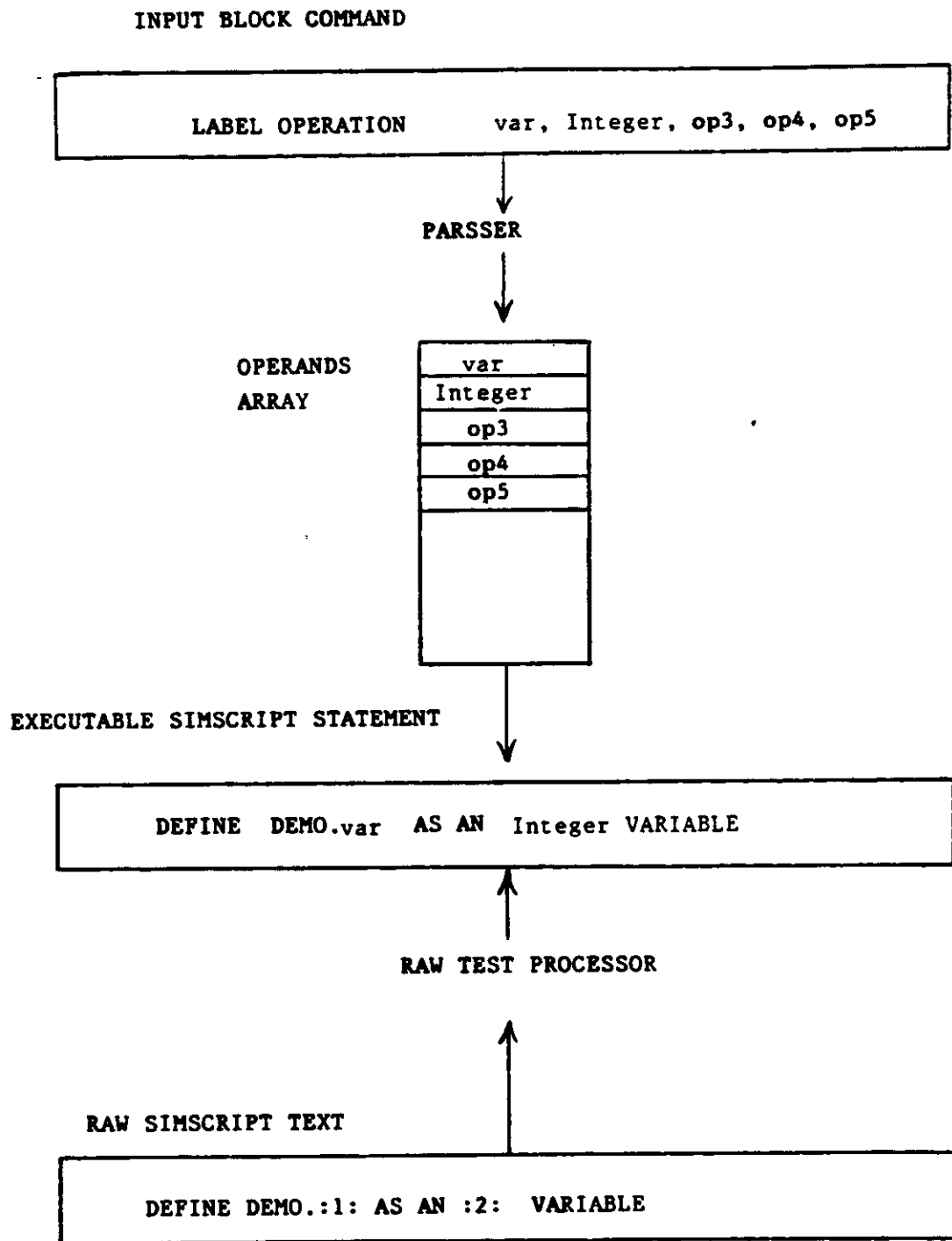


Figure 2-16. Primitive Elements in Raw SIMSCRIPT Text

The Preamble Generator searches the transaction list and defines two processes for each element. The transaction process and the corresponding transaction generator process. It then searches resource lists and generates a resource definition statement for each of the list elements. Using the variables list, left-monitored variables list, function list and matrix list, the preamble generator routine generates appropriate SIMSCRIPT definition statements for each of these list elements. The Preamble generator also generates ACCUMULATE and TALLY statements in order to gather statistics for each of the model parameters. These statistics include; maximum, standard deviation, average and histograms of appropriate parameters. Finally the Preamble generator routine generates statements to define "standard numerical attributes" used in GPSSV programs.

The main routine generator always generates a fixed SIMSCRIPT routine which calls the initialization routine, starts simulation and calls the output routine. The Initialization routine generator uses the matrix, resource and variable lists to generate SIMSCRIPT code which initializes them to given values as the simulation starts. In addition, the output routine generator uses the FUNCTION list to generate read statements to read

the input data file containing the distribution function.

The Output routine generator generates several routines in the target SIMSCRIPT program. These routines include a Central Output Routine which in turn calls all other specialized output generator routines. In addition, the resource report generator, queue report generator, transaction report generator, tables report generator and variable report generator routines are created.

The left-monitor routine generator searches the left-monitor variables list and, for each element in the list, it checks the reason for left-monitored variable and generates corresponding SIMSCRIPT code. Monitored variables can be generated as a result of "TEST" or "GATE" blocks which cause suspension of execution of a process whenever certain conditions are not met by the variable. The monitored variable is a model variable which is tested every time the variable changes and the corresponding process is reactivated if all conditions are met. Left-monitor variable list elements contain all the information regarding the reactivation conditions and suspended process.

Figure 2-17 shows the calling sequence of the BCSL compiler modules. As mentioned above, there is a compiler block processor routine for each command in BCSL (or Block in BGSL). These routines contain the SIMSCRIPT equivalent of each Block Command. In the next section several important block processor routines are described.

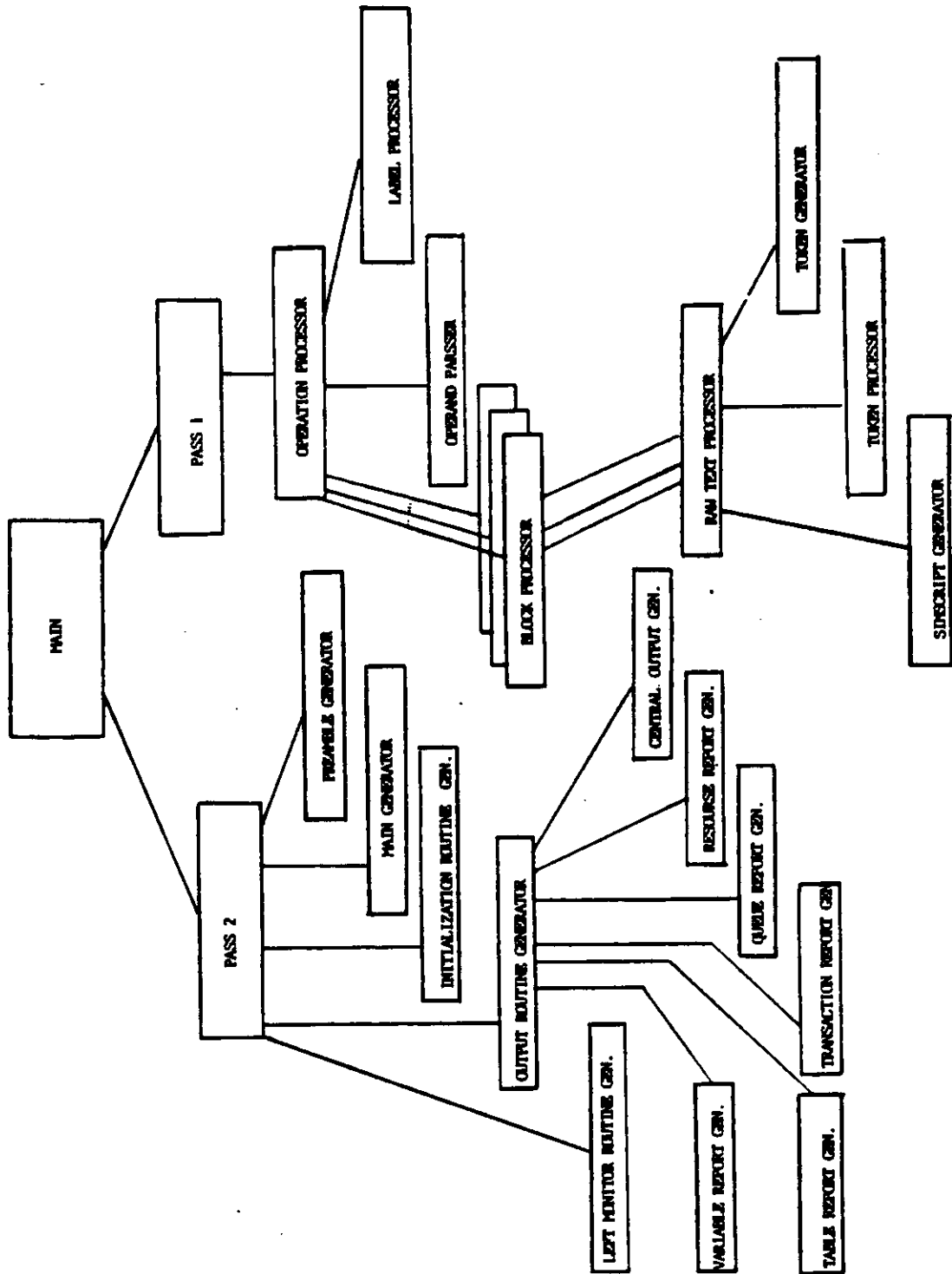


Figure 2-17. Calling Sequence of BCSL Compiler

2.5 Description Of Key Block Processor Routines

The goal of the this section is to develop an understanding of "block processor routines" logic and later in conjunction with SIMSCRIPT equivalent of each block, help provide a better understanding of the translation techniques used in this compiler. This section further clarifies the SIMSCRIPT Translation methodology by demonstrating the simplified equivalent of selected blocks in BCSL. For this reason the GENERATE, TERMINATE, ADVANCE, SEIZE, and RELEASE block processor routines have been selected as typical examples of block processor routines. It is assumed that readers are already familiar with the meaning and usage of these blocks in GPSSV.

The GENERATE block processor routine is of central importance to the BCSL compiler. It generates the "Transaction Generator Process" and starts the "Transaction Process". The first action taken by the GENERATE block processor is to include an "END" to the last transaction process because it is not possible for any of the blocks in the same segment to decide if the segment has ended; not even the TERMINATE block, because a segment can have more than one TERMINATE block in it.

One way to determine where a transaction process ends is to be aware of the graphical connectivity of the blocks (segment boundary). This data is available to the BCSL generator and is used to determine the order of BCSL statements. In this way the BCSL generator can guarantee the start of a new segment and mark the end of another one. In other words, all statements belonging to a segment are consecutive and the start of a new segment by the GENERATE block marks the end of the last segment. This makes it possible for the GENERATE block processor to end the last transaction process.

The Generate block processor creates a transaction element, sets the transaction name and priority attributes and files the element in the transaction set. This routine then investigates the type of distribution function used for inter-arrival time and generates the appropriate "Transaction Generator Routine". The transaction generator routine continuously activates the transaction processes. In between each activation it waits based on inter-arrival distribution function and parameters.

In addition the GENERATE block processor creates the top section of the "Transaction process". Figure 2-18 shows a simplified version of SIMSCRIPT equivalent

code generated for a selected GENERATE command block. The SIMSCRIPT equivalents of selected blocks are intended to help understand the reasoning behind and techniques used for the translation of each block. The convention used here for demonstration of SIMSCRIPT equivalents is as follows:

1. Capital letters represent the reserved and required SIMSCRIPT key words.
2. The primitives are shown in lower case. They represent values supplied by the block attributes.
3. The lower case statements within brackets contain the high level definition of several SIMSCRIPT statements (simplified for clarification).

The TERMINATE block processor routine usually generates the last part of the transaction process which was started by the GENERATE block processor routine. This routine generates SIMSCRIPT code to calculate "M1" standard numerical attribute (transaction life time in the model), increment the transaction counter and decrements the "simulation stop counter".

Block Format:

```
GENERATE      a,DS$b,,,,,JOB1
```

where

a = mean integration time

b = distribution function name

JOB1 = transaction name

SIMSCRIPT equivalent

```
PROCESS      GEN.JOB1
LET STOP.FLAG.JOB1(GEN.JOB1)=1
WAIT 0 UNITS
LET SEED=1
UNTIL STOP.FLAG.JOB1(GEN.JOB1)>=0
DO
WAIT b (a, SEED) UNITS
{Calculate inter generation time}
ACTIVATE A TRAN.JOB1 NOW
{Increment transaction counters}
LOOP
END
```

```
PROCESS      TRAN.JOB1
{Define local variables}
{Set start time for M1 SNA}
```

Figure 2-18. Translation of a GENERATE Block

The TERMINATE block processor generates statements to test the "simulation stop counter", stops simulation if the stop counter becomes zero and activates output the generator routine. Transactions eventually leaves the simulation model when the "transaction process" ends. Figure 2-19a shows the SIMSCRIPT equivalent for a TERMINATE block.

Block Format:

```
TERMINATE      a
```

where

a = Termination Counter

SIMSCRIPT Equivalent:

```
{calculate M1 SNA}  
{increment transaction counter}  
{decrement simulation stop counter by a}  
{if simulation stop counter is exhausted}  
CALL OUTPUT.GENERATOR ROUTINE  
STOP THE SIMULATION  
{end if}
```

Figure 2-19a. Translation of TERMINATE Block

The ADVANCE block processor establishes the distribution function needed for delaying the simulated process. It then generates a SIMSCRIPT statement to wait using the given distribution function, and prepares the wait statement parameters. The ADVANCE block will advance the simulation time by a number drawn from the distribution function defined in the second parameter of the ADVANCE block. Figure 2-19b shows the SIMSCRIPT equivalent for an ADVANCE block.

Block Format

```
ADVANCE a,DS$b
```

where

a = mean time

b = distribution function name

SIMSCRIPT Equivalent

```
WORK b(a, SEED) units
```

Block Format

```
ADVANCE a,b
```

WHERE

a = mean time

b = spread

SIMSCRIPT Equivalent

```
WORK UNIFORM.F(a-b,a+b,SEED) units
```

Figure 2-19b. Translation of ADVANCE Block

The SEIZE block processor searches the resource list for the facility name and adds the name of the new facility to the list if it already does not exist in the list. The SEIZE block processor generates the SIMSCRIPT statement which requests the facility and locks it for certain periods of time. Figure 2-20a shows the SIMSCRIPT equivalent of a SEIZE command block.

BLOCK format

```
SEIZE  a
Where
      a = Facility Name
```

SIMSCRIPT equivalent

```
PREAMBLE
-
-
RESOURCES INCLUDE a
-
-
END
MAIN
-
  CREATE EVERY a
-
-
END
PROCESS TRAN.name
-
-
-
REQUEST 1 a
-
-
-
END
```

Figure 2-20a. Translation of SEIZE Block

In the same manner as above the RELEASE block processor searches the resource list to verify that the released facility exists and then issues a RELINQUISH facility command in SIMSCRIPT to unlock the facility for other transactions. Figure 2-20b shows the SIMSCRIPT generated equivalent of a RELEASE command block.

Block Format

```
RELEASE      a
WHERE
  a = facility
```

SIMSCRIPT equivalent

```
PROCESS TRAN.name
-
-
RELINQUISH  1 a
-
-
END
```

Figure 2-20b. Translation of a RELEASE Block

The test block processor generates the needed statement to check the condition of the test variable. If the process can be suspended by the test block, then it will add the test variable to the list of monitored variables. Later, in Pass 2, a left-monitored variable, and left-monitored routine is generated to reactivate the suspended process. Figure 2-21 shows the translation of a TEST command block.

For more information on the block processor routines, please refer to the copy of the BCSL compiler program. Each block processor routine name is the same as the corresponding block name concatenated with ".BLK" extension (for example, to find the compiler routine for the TEST block, look for the TEST.BLK routine).

Block Format

```
TEST E    a,b
```

Where

```
    a = left variable  
    b = right variable
```

SIMSCRIPT EQUIVALENT

```
PREAMBLE
```

```
-  
-
```

```
DEFINE a AS AN INTEGER VARIABLE MONITORED ON THE LEFT
```

```
-  
-
```

```
END
```

```
LEFT ROUTINE a
```

```
DEFINE N.a AS AN INTEGER VARIABLE
```

```
ENTER WITH N.a
```

```
IF N.a EQ b
```

```
FOR EACH TRANSACTION IN BLOCKED.
```

```
TRANSACTION.LIST
```

```
WITH MONITORED VARIABLE EQUAL a
```

```
DO
```

```
{remove transaction from blocked transaction list}
```

```
REACTIVATE THIS TRAN.name NOW
```

```
LOOP
```

```
ALWAYS
```

```
MOVE FROM N.a
```

```
RETURN
```

```
PROCESS TRAN.name
```

```
-  
-
```

```
IF a EQ b
```

```
ELSE
```

```
{let monitored variable of blocked transaction = a}
```

```
FILE TRAN.name in BLOCKED.TRANSACTION.LIST
```

```
SUSPEND
```

```
ALWAYS
```

```
-  
-
```

```
END
```

Figure 2-21. Translation of a TEST Block

2.6 Differences Between GPSS And BCSL

Compatibility of GPSS and BCSL has been a major design goal in the development of BCSL. This allowed the Block Command Symbolic Language to be based on a popular and widely used language. In addition, we can verify the translation of simulation models built in BCSL by comparing run results with existing results from simulation runs for the same models in GPSS.

However, in order to improve GPSS capabilities, additional options have been added to some of the blocks and several new blocks have been defined. In this section we describe the modified blocks and operands. Chapter 4 on advanced features of BCSL contains the description of new blocks and their operands.

These modifications were done to allow the user to have access to a pool of distribution functions, to avoid entering the distribution function tables every time a new model is build and to allow floating point numbers and arithmetic statements to be used instead of integer numbers as is currently the case in GPSSV.

In the following sub-sections, the ADVANCE block, GENERATE block and FUNCTION commands are described in detail. In addition, the way in which floating point

and arithmetic statement are used in place of integer operands will be discussed.

2.6.1 ADVANCE Block - The ADVANCE block is provided in GPSS and BCSL to accomplish the task of freezing a transaction's motion for a prescribed length of time. The prescribed length of time is usually a random variable and represents a service time of a server. Table 2-1 shows the ADVANCE block operands and their descriptions.

TABLE 2-1 ADVANCE Block

ADVANCE	A, B, C
OPERAND	DESCRIPTION
-----	-----
A	MEAN TIME
B	EITHER SPREAD MODIFIER OR A DISTRIBUTION FUNCTION
C	SECOND DISTRIBUTION FUNCTION ARGUMENT (OPTIONAL)

In GPSS the service time distribution is expressed through operands A and B only and two service time categories exist: uniform distribution and user-supplied distribution.

Uniform distribution service times are utilized when the "A" supplies the average time that transactions entering the block are held there. The B operand

provides the half-width of range over which the holding times are uniformly distributed.

The user supplied distributions are utilized when operand B starts with the letters "FN". The rest of the B operand represents a distribution function name or number that has been supplied in the definition segment. In this case Operand A represents the mean time for the distribution function.

In BCSL a third option has been added for selecting distribution functions. This option requires operand B to start with the letters "DS". The rest of B operand supplies a distribution function name from available distribution functions in SIMSCRIPT language. Table 2-2 contains a list of these distribution functions and their arguments. In this case, Operand A supplies the mean distribution for service time and operand C supplies the second argument (e.g., standard deviation). This argument is optional and is required only by a few distributed functions.

DISTRIBUTION FUNCTION NAME	ARGUMENTS
BETA.F	POWER OF x , POWER OF $(1-x)$, SEED
BINOMIAL.F	NUMBER OF TRIALS, PROBABILITY OF SUCCESS, SEED
ERLANG.F	MEAN, k , SEED
EXPONENTIAL.F	MEAN, SEED
GAMMA.F	MEAN, k , SEED
LOG.NORMAL.F	MEAN, STANDARD DEVIATION, SEED
NORMAL.F	MEAN, STANDARD DEVIATION, SEED
POISSON.F	MEAN, SEED
RANDI.F	LOW VALUE (INTEGER), HIGH VALUE, SEED
UNIFORM.F	LOW VALUE (REAL), HIGH VALUE, SEED
WEIBULL.F	SHAPE, SCALE, SEED

TABLE 2-2. STATISTICAL DISTRIBUTION FUNCTIONS

2.6.2 GENERATE Block - BCSL and GPSS transactions are created and enter the model by use of the GENERATE block. The A operand specifies the mean time between creations. If the B operand is a number, then the time between transactions is uniformly distributed in the range of A-B to A+B. If the B operand starts with "FN" it represents a function name or number supplied by the user. The C operand specifies the time of the first transaction creation and is referred to as the offset interval. The D operand prescribes a limit on the number of transactions which can enter the model through a given GENERATE block. Each transaction created by the GENERATE block has a priority specified by operand E. Operands F and G, supplying the number and type of other operands in GPSS, are not needed in BCSL simulation language. These operands are not used in BCSL in order to stay compatible with existing GPSSV code. Table 2.3 contains a list of GENERATE block operands and their description in BCSL.

TABLE 2-3. OPERANDS OF GENERATE BLOCK

GENERATE	A, B, C, D, E, F, G, H, I
<u>OPERAND</u>	<u>DEFINITION</u>
A	MEAN TIME
B	SPREAD MODIFIER OR DISTRIBUTION FUNCTION
C	OFF SET INTERNAL
D	LIMIT COUNT
E	PRIORITY LEVEL
F	NO. OF PARAMETERS (NOT USED)
G	TYPE OF PARAMETERS (NOT USED)
H	TRANSACTION NAME (OPTIONAL)
I	SECOND DISTRIBUTION FUNCTION ARGUMENT

The Block Command Symbolic language provides a third option for the "B" operand, where "B" can represent a distribution function available in SIMSCRIPT (the same as ADVANCE block). The I operand is used for the second argument of the distribution function if needed. Table 2-2 contains the list of available distribution functions and their arguments. The H operand is used to assign names to transactions generated by the GENERATE block. Transaction name is an optional operand and is mainly used to improve the graphical representation of the models by selecting meaningful names for actual tasks and entities that the transaction is representing. If this operand is not provided the SIMSCRIPT translator will assign a number to the transaction and will generate a name for the transaction based on the assigned number. Notice that each transaction type will be assigned a unique name but all of the transactions created by the same GENERATE block will have the same name. The SIMSCRIPT translator checks for duplicate transaction names when names are provided by the user.

2.6.3 FUNCTION Commands - The FUNCTION command defines a distribution function supplied by the programmer in BCSL and GPSS languages. Given appropriate information about a distribution, the processor automatically performs a table lookup procedure using a uniform random number generator each time a sample is needed to be drawn from the distribution. In the FUNCTION definition command, the location operand contains the name of the FUNCTION. This name will be referred to by function modifier operands of ADVANCE and GENERATE blocks. The A operand contains the source of random number generators (1 through 8) and the B operand defines the number of different values that are provided in order to specify the discretely distributed random variable. The values themselves and their corresponding cumulative frequencies are provided in the following lines of the FUNCTION statement in either GPSS or BCSL. Table 2.4 contains the list of operands and their definitions for FUNCTION commands.

TABLE 2-4 FUNCTION COMMAND

LOC FUNCTION A, B, C

OPERAND DESCRIPTION
 ----- -----

LOC FUNCTION NAME

- A RANDOM STREAM NUMBER
- B NUMBER OF DIFFERENT VALUES THAT ARE IN
 THE FOLLOWING LINES
- C VARIABLE TYPES

Follow up lines of the FUNCTION command have the following syntax:

$X_1, Y_1 / X_2, Y_2 / \dots / X_i, Y_i / \dots X_n, Y_n.$

Where X_i and Y_i are the i th cumulative probability and the associated random variable value respectively, and

$X_1 \gg X_2 \dots \gg X_i \dots \gg X_n.$

The C operand defines the type of Y values which in BCSL can be real or integer.

Notice that in BCSL all X values are real and the cumulative probability (X_n) is "1."

The Block Graphical Symbolic Language represents all the data needed for FUNCTION commands in one block, thus providing a better graphical representation of the data.

2.6.4 Floating Point And Arithmetic Statements As Operands - In GPSSV the only numerical input type is integer. This imposes a significant restriction on the user. BCSL not only allows floating point numbers and variables but it also accepts any arithmetic statement as a variable.

Each valid statement consists of numbers and variable names separated by the arithmetic operands "+", "-", "/", "*" and parentheses. BCSL arithmetic statements have the same syntax as FORTRAN or SIMSCRIPT arithmetic statements. This feature improves all the commands that use numerical operand (Appendices A and B refers to them as "K" operands).

2.7 Output Generator And Output Files

The output generator is in charge of the execution of the simulation model and generation of the simulation output results (program output). The generated output results are stored in a file named after the model with extension of RESULTS. This file is displayed on the user terminal, and in addition a hard copy can be obtained upon user request.

The Graphical Simulation System generates several files while the model development and compilation is in progress, including the block graph file, BCSL file, BCSL Compiler listing, and SIMSCRIPT equivalent file. The BCSL and SIMSCRIPT equivalent files are editable by the user, which makes it possible for the user to manually alter the model in different stages of translation or compilation. The BCSL compiler listing file contains every input line and its corresponding output statements. It allocates the input line number, output line number and specifies if the output line belongs to pass 1 or pass 2 of the compiler.

It is possible to display the intermediate state of the simulation model by graphically displaying the model and animating the movement of the transactions through the model blocks and their queues.

In Chapter 6 (Future Research Possibilities) we will describe how the Graphical Simulation System can be expanded to provide animation capabilities. Figure 2-22 shows the output generator, its input files, output files and how potentially the graphical animated output display can be added to the system.

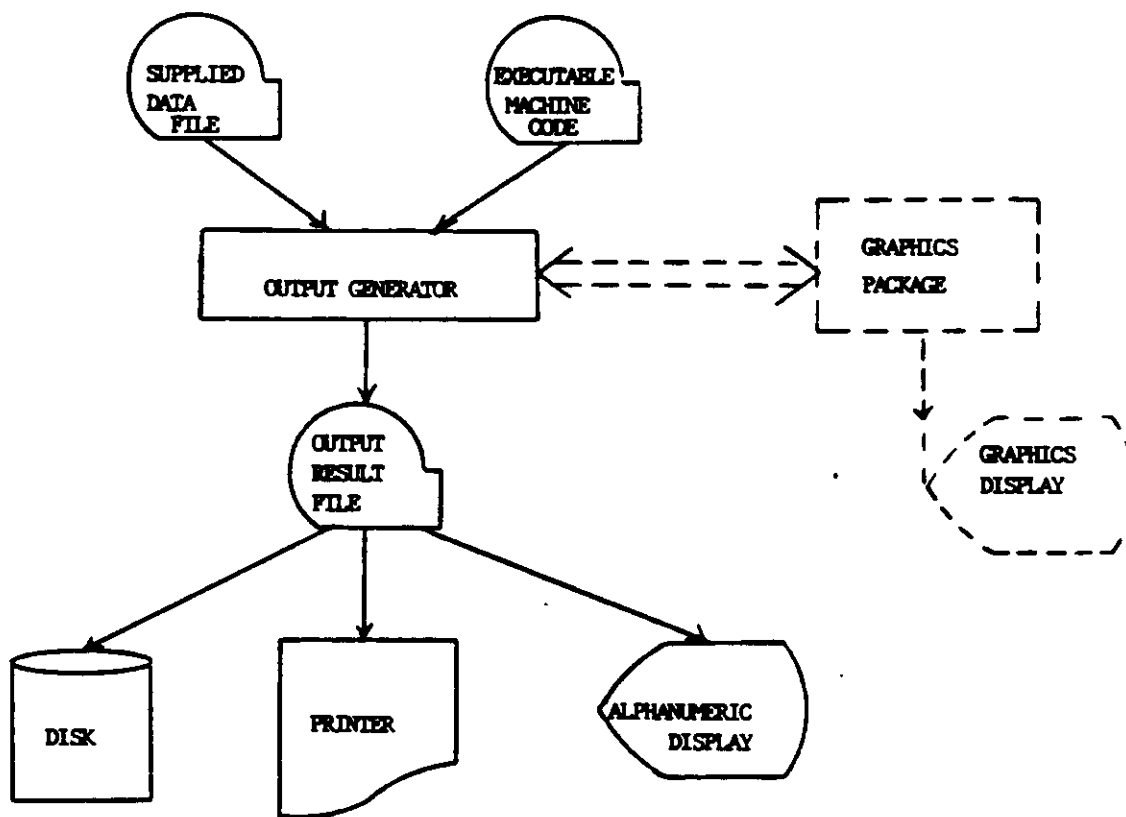


Figure 2-22. Output Generator

3 CLASSIC APPLICATIONS AND CASE STUDIES

In this chapter three problems were selected to be studied and discussed in detail. These problems were taken from Schriber, who presents a GPSS model and corresponding results of simulation runs for each system in his book "Simulation Using GPSS" (see reference ---).

The first problem represents the classic one-line, one-server queuing system. It is used to familiarize the reader with the translation philosophy and techniques which have been discussed in the previous chapters. This problem is intended to integrate all of the topics discussed in regards to translation methodology and implementation techniques used by BCSL compiler by studying a simple model.

The second problem represents the production shop model. The objective of this research has been to translate a graphical representation of a production shop model into SIMSCRIPT and successfully simulate such a system using a general purpose graphical simulation language. Detailed discussion of this problem and proof of correctness for the results obtained are essential to this dissertation.

The last problem selected is a more complicated model representing a BUS STOP. This model includes several important advanced blocks. Their translations demonstrate the usage of "left-monitored routines" and corresponding techniques. The ability to use the Block Command Symbolic language as a general purpose simulation language and the versatility of this language are demonstrated by this example.

3.1 One-Line, One-Server Queuing System

3.1.1 Statement Of The Problem - A simple one-line, one-server queuing system was selected to demonstrate how SIMSCRIPT translation is done and which SIMSCRIPT statements correspond to which blocks in the BGS L model. This model is described in Schriber's book as follows:

"The interarrival time of the customer at a one-chair barber shop is uniformly distributed over the range 18+6 minutes. Service time for hair cuts is 16+4 minutes. Customers coming to the shop get their hair cut on a first-come, first-served basis, then leave. Model the shop, making provisions to collect data on the waiting line. Then run the model through 8 hours of simulated time."

3.1.2 Discussion Of The Model - This model includes two model segments and a control segment. The main model segment is easily constructed as a single sequence of blocks, where the order of blocks corresponds to the stages through which customers move in the real system.

Customers arrive, they queue and wait if necessary for their turn. Then they engage the barber, get their hair cut, release the barber, and leave. A second normal segment known as the "timer segment", is used to

control the duration of the simulation run. It basically uses a GENERATE and a TERMINATE block which trigger a simulation STOP flag after a certain amount of time has passed. Figure 3-1 illustrates the Block Graphic solution for this problem which is the same as the GPSSV model.

Figure 3-2 shows the Block Command Symbolic language equivalent of this code. Each line of BCSL represents a block in BCSL and the corresponding block number is shown in column 72-80 of the same line in BCSL. Transaction names are selected for better pictorial understanding of the model. These are customer transactions and clock transactions which appear as the 7th attribute of the GENERATE block in order to stay compatible with existing GPSSV Code.

Notice that the mean and spread values for the distribution functions are in real format. Figures 3-3a through 3-3h contain the SIMSCRIPT equivalent of the model, which is generated by the BCSL compiler (SIMSCRIPT translator). The original BCSL command is added as a comment to the SIMSCRIPT code ahead of its corresponding SIMSCRIPT equivalent. Given that each BCSL command contains the block number of the original graphic block, it is possible to trace back from

SIMSCRIPT statements to the blocks in the Block Graphic
Symbolic language.

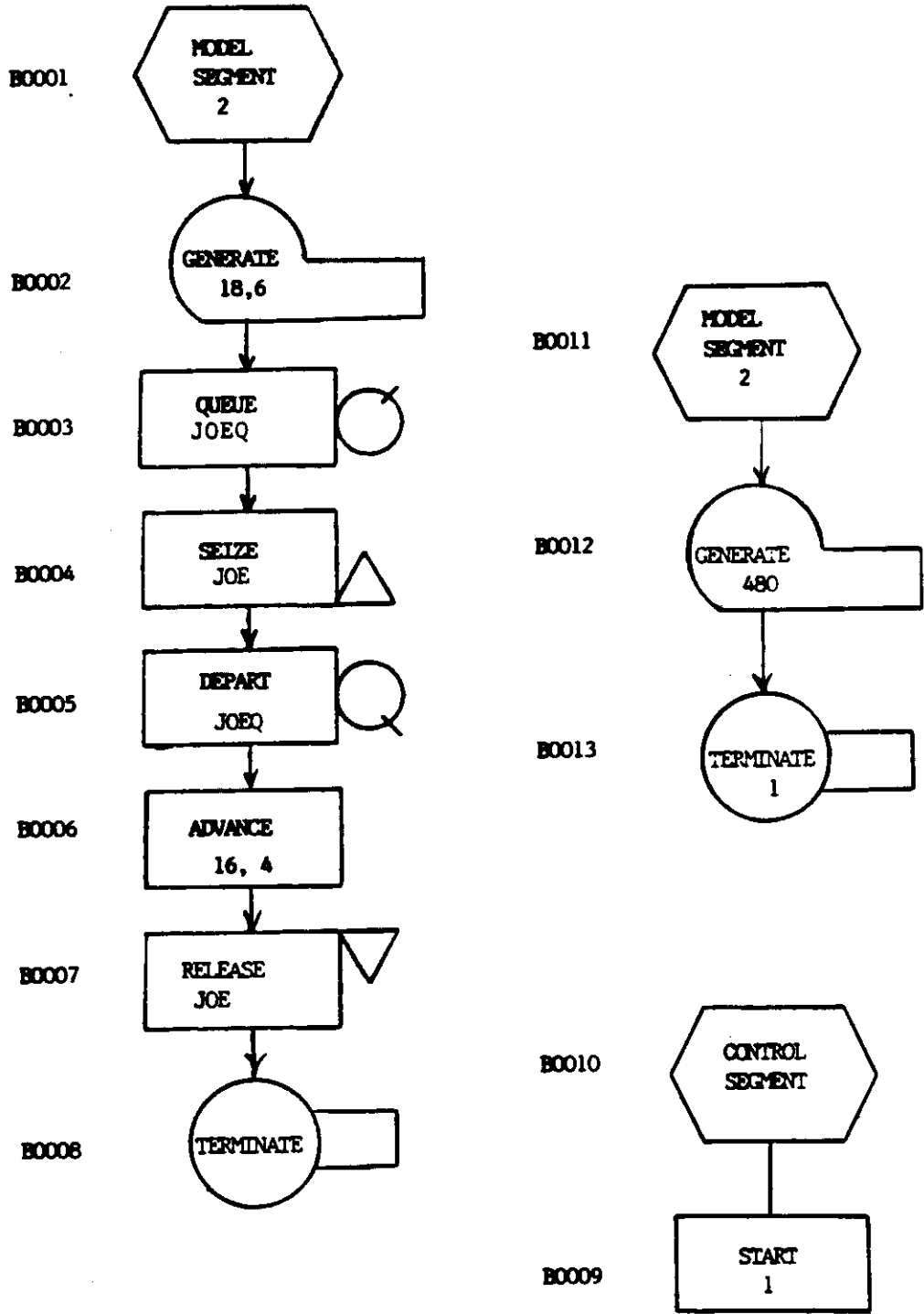


Figure 3-1. BGSL Model for the Barber Shop

```

*****
**
** PCSL CODE GENERATOR VERSION 1.0
**
*****
*
*
*   SIMULATE
*
*
*** DEFINITION SEGMENT
*
*
*   GENERATE 18.,0.,,,,,,CUSTOMER      B0002
*   QUEUE    JOEQ                      B0003
*   SEIZE    JOE                        B0004
*   DEPART   JOEQ                      B0005
*   ADVANCE  16.,4.                    B0006
*   LEAVE    JOE                        B0007
*   TERMINATE                                B0008
*
*
*** NEW SEGMENT
*
*
*   GENERATE 480.,,,,,,CLOCK          B0012
*   TERMINATE 1                        B0013
*
*
*** CONTROL SEGMENT
*
*
*   START    1                          B0009
*
*
*   END

```

Figure 3-2. BCSL Model for the Barber Shop

3.1.3 Discussion Of SIMSCRIPT Equivalent - The Preamble section (Figure 3-3a) defines four processes: customer transaction, clock transaction, customer transaction generator and clock transaction generator. The only resource in this model is the barber "joe". The queue block has caused the generation of standard numerical attribute definition statements. The P-array and X-array are defined globally for cross process communications which provide GPSS compatibility of P and X operands. Additional definition statements are for the transaction counter and the seeds used in the distribution functions. Accumulate and tally statements are generated for each resource and their corresponding queue.

The Initialize routine (Figure 3-3b) initializes the number of barbers in the model, transaction counters and stop flag. Finally the initialization routine activates the process generator routines. The main routine and output routine (Figure 3-3b) always call predefined routines. The output subroutines (Figure 3-3c and 3-3d) each generate appropriate reports for resources, queues, transactions, tables and variables in the model.

The customer generator process (Figure 3-3e) activates a customer transaction process using a uniform distribution function and accumulates the statistics regarding created transactions. The customer transaction process (Figure 3-3f) reflects the translation of each BCSL command into SIMSCRIPT and the sequence in which the translation is done. Each transaction process also keeps statistics on the time spent in the model (standard numerical attribute M1) for each occurrence of the transaction.

The clock generator process (Figure 3-3g) activates the clock transaction process (Figure 3-3g) which in turn will stop the simulation by decrementing the stop flag counter to zero and finally will call the output generator routine before termination.

3.1.4 Discussion Of Results - Figure 3-4 shows the results of the simulation run for the barber shop model. Tables 3-1 and 3-2 contain the definition for column headings in the report. In order to verify the correctness of translation of the model, these results are compared with the results of simulation run results from the same GPSS model in Schriber's book where the average utilization of the barber is 86 percent and average contents of the queue is 0.16 customers.

Facility utilization and average contents of the queue matches the analytical modeling results as well as Schriber's book results.

```

".....*.....
".....*.....
".....*..... SIMSCRIPT CODE GENERATED BY .....*
".....*..... BCSL TO SIMSCRIPT CROSS COMPILER .....*
".....*..... VERSION 1.2 .....*
".....*.....
".....*.....
1 PREAMBLE
2 NORMALLY MODP IS REAL
3 PROCESSES
4 EVERY TRAM.CUSTOMER HAS A MONITOR.V
5 AND MAY BELONG TO THE BLOCKED.CUSTOMER
6 EVERY TRAM.CLOCK HAS A MONITOR.V
7 AND MAY BELONG TO THE BLOCKED.CLOCK
8 THE SYSTEM OWNS THE BLOCKED.CUSTOMER
9 THE SYSTEM OWNS THE BLOCKED.CLOCK
10 EVERY GEN.CUSTOMER HAS A STOP.FLAG.CUSTOMER
11 EVERY GEN.CLOCK HAS A STOP.FLAG.CLOCK
12 RESOURCES
13 EVERY JOE HAS A NUM.OF.JOE
14 DEFINE TIMEI.P.JOE AS A REAL VARIABLE
15 DEFINE TIMEIN.Q.JOE AS A REAL VARIABLE
16 DEFINE NUM.ENT.Q.JOE AS AN INTEGER VARIABLE
17 DEFINE CAPACITY.JOE AS AN INTEGER VARIABLE
18 ''DEFINE VARIABLES AND LEFT MONITORED VARIABLES
19 DEFINE STOP.FLAG AS INTEGER VARIABLE
20 ''DEFINE SNA FOR QUEUES
21 DEFINE Q$JOEQ AS INTEGER VARIABLE
22 DEFINE QCSJOEQ AS INTEGER VARIABLE
23 DEFINE QCSJOEQ AS INTEGER VARIABLE
24 DEFINE QZ$JOEQ AS INTEGER VARIABLE
25 DEFINE Q.T.JOEQ AS INTEGER VARIABLE
26 DEFINE Q.I.JOEQ AS INTEGER VARIABLE
27 ''DEFINE TEST BLOCK FLAGS
28 ''DEFINE STATISTICAL VARIABLES
29 ''DEFINE RANDOM VARIABLES
30 ''DEFINE MATRICES
31 ''DEFINE GPSS GLOBAL VARIABLES
32 DEFINE P1 TO MEAN P_ARRAY(1)
33 DEFINE P2 TO MEAN P_ARRAY(2)
34 DEFINE P3 TO MEAN P_ARRAY(3)
35 DEFINE P4 TO MEAN P_ARRAY(4)
36 DEFINE P5 TO MEAN P_ARRAY(5)
37 DEFINE P6 TO MEAN P_ARRAY(6)
38 DEFINE P7 TO MEAN P_ARRAY(7)
39 DEFINE P8 TO MEAN P_ARRAY(8)
40 DEFINE P9 TO MEAN P_ARRAY(9)
41 DEFINE P10 TO MEAN P_ARRAY(10)
42 DEFINE X1 TO MEAN X_ARRAY(1)
43 DEFINE X2 TO MEAN X_ARRAY(2)
44 DEFINE X3 TO MEAN X_ARRAY(3)
45 DEFINE X4 TO MEAN X_ARRAY(4)
46 DEFINE X5 TO MEAN X_ARRAY(5)
47 DEFINE X6 TO MEAN X_ARRAY(6)
48 DEFINE X7 TO MEAN X_ARRAY(7)

```

Figure 3-3a. SIMSCRIPT Equivalent Model for the Barber Shop

```

49 DEFINE X8 TO MEAN X_ARRAY(8)
50 DEFINE X9 TO MEAN X_ARRAY(9)
51 DEFINE X10 TO MEAN X_ARRAY(10)
52 DEFINE UNITS TO MEAN MINUTES
53 DEFINE MONITOR.V AS TEXT VARIABLE
54 DEFINE H1 AS A REAL VARIABLE
55 DEFINE C1 TO MEAN TIME.V*HOURS.V*MINUTES.V
56 ''TRANSACTION DEFINITION
57 DEFINE TRAN.CNT.CUSTOMER AS AN INTEGER VARIABLE
58 DEFINE C.TRAN.CNT.CUSTOMER AS AN INTEGER VARIABLE
59 DEFINE SEED.TRAN.CUSTOMER AS AN INTEGER VARIABLE
60 DEFINE TRAN.CNT.CLOCK AS AN INTEGER VARIABLE
61 DEFINE C.TRAN.CNT.CLOCK AS AN INTEGER VARIABLE
62 DEFINE SEED.TRAN.CLOCK AS AN INTEGER VARIABLE
63 ''DEFINE TABLE VARIABLES
64 ''TALLY AND ACCUMULATE FOR RESOURCES
65 ACCUMULATE UTIL.JOE AS AVERAGE AND
66 MAX.NI.JOE AS MAXIMUM OF N.I.JOE
67 ACCUMULATE AVG.Q.LEN.JOE AS AVERAGE AND
68 MAX.Q.LEN.JOE AS MAXIMUM OF N.Q.JOE
69 TALLY AVG.TIMEIN.P.JOE AS MEAN OF TIMEIN.P.JOE
70 TALLY AVG.TIMEIN.Q.JOE AS MEAN OF TIMEIN.Q.JOE
71 ''TALLY AND ACCUMULATE FOR STATISTICAL VARIABLES
72 ''TALLY AND ACCUMULATE FOR QUEUES
73 ACCUMULATE QASJREQ AS AVERAGE AND QNSJREQ AS MAXIMUM
74 OF QNSJREQ
75 TALLY QISJREQ AS MEAN OF Q.I.JREQ
76 TALLY QISJREQ AS MEAN OF Q.I.JREQ
77 DEFINE GEN.TIME.CUSTOMER AS A REAL VARIABLE
78 TALLY AVG.GEN.TIME.CUSTOMER AS MEAN OF GEN.TIME.CUSTOMER
79 DEFINE GEN.TIME.CLOCK AS A REAL VARIABLE
80 TALLY AVG.GEN.TIME.CLOCK AS MEAN OF GEN.TIME.CLOCK
81 ''HISTOGRAM FOR TABLES
82 END

```

Figure 3-3a. SIMSCRIPT Equivalent Model for the Barber Shop (Con't)

CACI SIMSCRIPT II.5 IBM S/370 19.3 PAGE
17-NOV-1985 16:44 (1)

OPTIONS TRM,LOAD,LD,TRACE2,NOTERS,CHK,REN=REN

```

1  MAIN
2  CALL INITIALIZE
3  START SIMULATION
4  CALL OUTPUT_ROUTINE
5  END

```

CACI SIMSCRIPT II.5 IBM S/370 19.3 PAGE
17-NOV-1985 16:44 (1)

OPTIONS TRM,LOAD,LD,TRACE2,NOTERS,CHK,REN=REN

```

-1  ROUTINE INITIALIZE
2  NORMALLY MODE IS INTEGEL
3  LET CAPACITY.JOE = 1
4  CREATE EVENT JOE(1)
5  LET U.JOE(1) = 1
6  LET TRAN.CHTR.CUSTOMER = 0
7  LET TRAN.CHTR.CLOCK = 0
8  LET STOP.FLAG = 1
9  ACTIVATE A GEN.CUSTOMER NOW
10  ACTIVATE A GEN.CLOCK NOW
11  END

```

CACI SIMSCRIPT II.5 IBM S/370 19.3 PAGE
17-NOV-1985 16:44 (1)

OPTIONS TRM,LOAD,LD,TRACE2,NOTERS,CHK,REN=REN

```

1  ROUTINE OUTPUT_ROUTINE
2  WRITE AS //,/,B 20,
3  *****//,B 20,
4  ***,B 60,***,/,B 20,
5  *** GRAPHICAL SIMULATION SYSTEM OUTPUT ***//,B 20,
6  ***,B 60,***,/,B 20,
7  *****//,B 20,
8  USING 6
9  CALL RESOURCE_PRINT
10  CALL QUEUE_PRINT
11  CALL TRANSACTION_PRINT
12  CALL TABLE_PRINT
13  CALL STAT_PRINT
14  CALL IVAR_PRINT
15  END

```

Figure 3-3b.

```

1 ROUTINE RESOURCE_PRINT
2 DEFINE I AS AN INTEGER VARIABLE
3 WRITE AS //,/,/,/,B 2,"FACILITY",B 12,"CAPACITY",
4 B 23,"AVERAGE",B 37,"NUMBER OF",
5 E 46,"AVERAGE",B 59,"CURRENT",B 70,"MAXIMUM",
6 /,B 2,"NAME",B 23,"UTILIZATION",
7 E 37,"ENTRIES",
8 E 48,"TIME/TRAN",B 59,"CONTENT",B 70,"CONTENT",/
9 USING 6
10 WRITE CAPACITY.JOE,(UTIL.JOE/CAPACITY.JOE),
11 (NUM.ENT.Q.JOE - E.Q.JOE),
12 AVG.TIMEIN.F.JOE,E.I.JOE,MAX.NI.JOE AS /,B 2,
13 "JOE",B 12,I 6,B 23,D(9,3),B 37,I 6,B 48,D(9,3),
14 B 59,I 6,B 70,I 6
15 WRITE AS //,/,/,/,B 2,"STORAGE",B 13,"CAPACITY",
16 B 25,"AVERAGE",B 40,"NUMBER OF",
17 B 55,"AVERAGE",B 70,"CURRENT",B 85,"MAXIMUM",
18 /,B 2,"NAME",B 25,"UTILIZATION",
19 E 40,"ENTRIES",
20 E 55,"TIME/TRAN",B 70,"CONTENT",B 85,"CONTENT",/
21 USING 6
22 WRITE AS //,/,/,/,B 2,"RESOURCE",B 15,"AVERAGE",
23 E 28,"TOTAL",B 41,"AVERAGE",B 53,"CURRENT",
24 E 66,"MAXIMUM",/,B 2,"QUEUE",B 15,"CONTENT",
25 E 26,"ENTRIES",B 41,"TIME/TRAN",B 53,"CONTENT",
26 B 66,"CONTENTS",/
27 WRITE AVG.Q.LEN.JOE,NUM.ENT.Q.JOE,AVG.TIMEIN.Q.JOE,
28 E.Q.JOE,MAX.Q.LEN.JOE AS /,B 2,"JOE",B 14,D(8,3),
29 E 28,I 6,B 41,D(9,3),B 53,I 6,B 66,I 6 USING 6
30 END

```

Figure 3-3c.

```

1 ROUTINE QUEUE_PRINT
2 WRITE AS //,/,/,/,B 2,"QUEUE",B 12,"EALIBUH",
3 B 22,"AVERAGE",E 32,"TOTAL",
4 B 42,"ZELO",B 52,"AVERAGE",B 62,"SAVERAGE",
5 B 72,"CUMASMI",
6 /,B 2,"NAME",B 12,"CONTENTS",B 22,"CONTENTS",
7 B 32,"SERIES",
8 B 42,"SERIES",B 52,"TIME/TAN",
9 B 62,"TIME/TAN",B 72,"CONTENTS",/
10 USING 6
11 WRITE WFIJOEU,QAJJOEU,QCSJOEU,
12 QZFJOEU,QTBJOEU,QXSJOEU,QBJOEU AS /,B 2,
13 "JOE",B 12,I 6,E 22,D(9,3),B 32,I 6,E 42,I 6,
14 P 52,D(9,3),B 62,D(9,3),B 72,I 6
15 END
  
```

```

1 ROUTINE TRANSACTION_PRINT
2 DEFINE I AS AN INTEGER VARIABLE
3 WRITE AS //,/,/,/,P 10,"TRANSACTION",B 25,"NUMBER",
4 B 40,"AVERAGE",/,B 10,"NAME",B 25,"CREATED",
5 E 40,"CREATION TIME",/ USING 6
6 WRITE I12N.CM12.CUSTOMER,AVG.GEN.TIME.CUSTOMER AS /,B 10,"CUSTOMER",
7 P 25,I 6,B 40,D(9,3) USING 6
8 WRITE I12N.CM12.CLOCK,AVG.GEN.TIME.CLOCK AS /,B 10,"CLOCK",
9 B 25,I 6,E 40,D(9,3) USING 6
10 END
  
```

Figure 3-3d.

```

*****
****
**** BCSI CODE GENERATED VERSION 1.0
****
*****
***
***
**      SIMULATE
***
***
**** DEFINITION SEGMENT
***
***
**      GENERATE 18.,6.,,,,,,CUSTOMER                B0002

1  PROCESS GEN.CUSTOMER
2  LET STOP.FLAG.CUSTOMER(GEN.CUSTOMER) = 1
3  LET SEED.TRAN.CUSTOMER = 1
4  WAIT 0 UNITS
5  UNTIL STOP.FLAG.CUSTOMER(GEN.CUSTOMER) <= 0
6  DO
7  DEFINE GEN1.CUSTOMER AS A REAL VARIABLE
8  LET GEN1.CUSTOMER = TIME.V
9  WAIT UNIFORM.F((18.-6.)/1.0,(18.+6.)/1.0,
10 SEED.TRAN.CUSTOMER) UNITS
11 LET GEN.TIME.CUSTOMER=(TIME.V - GEN1.CUSTOMER)*HOURS.V*MINUTES.V
12 ACTIVATE A TRAN.CUSTOMER NO#
13 LET TRAN.CNTL.CUSTOMER=TRAN.CNTL.CUSTOMER + 1
14 LET C.TRAN.CNTL.CUSTOMER=C.TRAN.CNTL.CUSTOMER + 1
15 LOOP
16 END

```

Figure 3-3e.

```

1  PROCESS TRAN.CUSTOMER
2  DEFINE P_ARRAY AS INTEGER,1-DIMENSIONAL ARRAY
3  RESERVE P_ARRAY AS 10
4  DEFINE TIN.N1 AS A REAL VARIABLE
5  TIN.N1 = TIME.V
6  "      QUEUE      JOEQ                               B0003
7
8  LET TINQ.JOEQ = TIME.V
9  LET QCSJOEQ = QCSJOEQ + 1
10 LET QSJOEQ = QSJOEQ + 1
11 LET QUSJOEQ = QUSJOEQ + 1
12 "      SEIZE      JOE                               B0004
13
14 LET TIN.Q.JOE = TIME.V
15 LET NUM.ENT.Q.JOE = NUM.ENT.Q.JOE + 1
16 REQUEST 1 JOE
17 LET TIMEIN.Q.JOE=(TIME.V - TIN.Q.JOE)*HOURS.V*MINUTES.V
18 LET TIN.P.JOE = TIME.V
19 "      DEPART     JOEQ                               B0005
20
21 LET Q.T.JOEQ = (TIME.V - TINQ.JOEQ)*HOURS.V*MINUTES.V
22 IF Q.T.JOEQ LE 0
23 LET QZSJOEQ = QZSJOEQ + 1
24 ELSE
25 LET Q.I.JOEQ = (TIME.V - TINQ.JOEQ)*HOURS.V*MINUTES.V
26 ALWAYS
27 LET Q.SJOEQ = QSJOEQ - 1
28 LET QUSJOEQ = QUSJOEQ - 1
29 "      ADVANCE   16.,4.                               B0006
30
31 WORK UNIFORM.P((16.-4.)/1.0,(16.+4.)/1.0,
32 S&D.INAN.CUSTOMER) UNITS
33 "      LEAVE     JOE                               B0007
34
35 RELINQUISH 1 JOE
36 LET TIMEIN.P.JOE=(TIME.V - TIN.P.JOE)*HOURS.V*MINUTES.V
37 "      TERMINATE
38
39 LET N1 = (TIME.V - TIN.N1)*HOURS.V*MINUTES.V
40 LET C.TRAN.CNT.CUSTOMER = C.TRAN.CNT.CUSTOMER - 1
41 RETURN
42 ***
43
44 ***
45
46 ***** NEW SEGMENT
47
48 ***
49
50 ***
51
52 "      GENERATE  480.,,,,,,CLOCK                       B0012
53
54 END

```

Figure 3-3f.


```

1 PROCESS GEN.CLOCK
2 LET STOP.FLAG.CLOCK(GEN.CLOCK) = 1
3 LET SEED.TIME.CLOCK = 1
4 WAIT 0 UNITS
5 UNTIL STOP.FLAG.CLOCK(GEN.CLOCK) <= 0
6 DO
7 DEFINE GEN1.CLOCK AS A REAL VARIABLE
8 LET GEN1.CLOCK = TIME.V
9 WAIT UNIFORM.F((490.-0)/1.0,(480.+0)/1.0,
10 SEED.TIME.CLOCK)UNITS
11 LET GEN.TIME.CLOCK=(TIME.V - GEN1.CLOCK)*HOURS.V*MINUTES.V
12 ACTIVATE 1 TRAN.CLOCK NOW
13 LET TRAN.CNT.CLOCK=TRAN.CNTM.CLOCK + 1
14 LET C.TRAN.CNT.CLOCK=C.TRAN.CNT.CLOCK + 1
15 LOOP
16 END
    
```

```

1 PROCESS TRAN.CLOCK
2 DEFINE P_ARRAY AS INTEGER,1-DIMENSIONAL ARRAY
3 RESERVE P_ARRAY AS 10
4 DEFINE TIM.N1 AS A REAL VARIABLE
5 TIM.N1 = TIME.V
6 ** TERMINATE 1 B0013
7
8 LET N1 = (TIME.V - TIM.N1)*HOURS.V*MINUTES.V
9 LET C.TRAN.CNT.CLOCK = C.TRAN.CNT.CLOCK - 1
10 LET STOP.FLAG = STOP.FLAG - 1
11 IF STOP.FLAG LE 0
12 CALL OUTPUT_ROUTINE
13 STOP
14 ALWAYS
15 RETURN
16 ***
17 ***
18 ***
19
20 ***** CONTROL SECTION1
21
22 ***
23
24 ***
25
26 ** START 1 B0009
27
28 ***
29
30 ***
31
32 ** END
    
```

Figure 3-3g.

```

*****
*
** GRAPHICAL SIMULATION SYSTEM OUTPUT **
*
*****

```

FACILITY NAME	CAPACITY	AVERAGE UTILIZATION	NUMBER OF ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENT	MAXIMUM CONTENT
JOE	1	.853	25	16.581	1	1

STORAGE NAME	CAPACITY	AVERAGE UTILIZATION	NUMBER OF ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENT
--------------	----------	---------------------	-------------------	-------------------	-----------------

RESOURCE QUEUE	AVERAGE CONTENT	TOTAL ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENT	MAXIMUM CONTENTS
JOE	.095	25	1.831	0	1

QUEUE NAME	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	AVERAGE TIME/TRAN	AVERAGE CURRENT CONTENTS
JOE	1	.095	25	14	1.920	4.364

TRANSACTION NAME	NUMBER CREATED	AVERAGE CREATION TIME
CUSTOMER	25	18.745
CLOCK	1	480.000

Figure 3-4. Simulation Results for the Barber Shop Model

TABLE 3-1. COLUMN DEFINITION FOR FACILITY REPORT

COLUMN	DESCRIPTION
FACILITY NAME	Names of the various facilities used in the model
CAPACITY	The number of units within the facility
AVERAGE UTILIZATION	Fraction of the time that the corresponding facilities were in a state of capture during the simulation
NUMBER OF ENTRIES	Number of transactions entering the facility
AVERAGE TIME/TRAN	Average service time per transaction
CURRENT CONTENT	The number of transactions in the facility at the end of the simulation run
MAXIMUM CONTENT	The maximum number of transactions in the facility reached during the simulation run

TABLE 3-2. COLUMN DEFINITION FOR QUEUE REPORT

COLUMN	DESCRIPTION
QUEUE NAME	Name of the various queues used in the model
MAXIMUM CONTENTS	Largest number of Transactions in the queue during the simulation
AVERAGE CONTENTS	Average value of queue contents
TOTAL ENTRIES	Total number of entries to the queue
ZERO ENTRIES	Total number of entries to the queue which did not have to wait
AVERAGE TIME/TRAN	Average time that each entry spent waiting in the queue (Zero entries are included in this average)
\$AVERAGE TIME/TRAN	Average time that each queue entry spent waiting in the queue (Zero entries are excluded from this average)
CURRENT CONTENTS	The contents of queues when simulation stops

3.2 Simulation Of The Production Shop

3.2.1 Statement Of The Problem - A production shop is composed of six different groups of machines. Each group consists of a certain number of machines of a given kind, as indicated in Table 3-1. For example, group 1 consists of 14 casting units. Within any single group, the machines are identical. It does not matter, then, which particular casting unit is used to perform a casting operation, or which particular shaper is used to perform a shaping operation, etc. Three different types of jobs move through the production shop. The job-types are designated as type 1, type 2, and type 3. Each job-type requires that operations be performed at a specified kind of machines in a specified sequence. The total number of kinds of machines each job-type must visit, and the correspondig visitation sequence, are shown in Table 3-2.

For example, jobs of type 1 must visit a total of four machines. The machines themselves, listed in the sequence in which they must be visited, are casting unit, planer, lathe, and polishing machine. The table also shows the mean time required by each job-type for each operation that must be performed on it.

TABLE 3-3. COMPOSITION OF MACHINE GROUPS IN PRODUCTION SHOP

GROUP NUMBER	MACHINES IN GROUP	
	KIND	NUMBER
1	CASTING UNITS	14
2	LATHES	5
3	PLANNERS	4
4	DRILL PRESSES	8
5	SHAPERS	16
6	POLISHING MACHINES	4

TABLE 3-4. VISITATION SEQUENCES AND MEAN OPERATION TIMES FOR THE THREE JOB TYPES

JOB TYPE	TOTAL NO. OF MACHINES TO BE VISITED	MACHINE VISITATION SEQUENCE	MEAN OPERATION TIME (MINUTES)
1	4	CASTING UNIT	125
		PLANNER	35
		LATHE	20
		POLISHING MACHINE	60
2	3	SHAPER	105
		DRILL PRESS	90
		LATHE	65
3	5	CASTING UNIT	235
		SHAPER	250
		DRILL PRESS	50
		PLANNER	30
		POLISHING UNIT	25

For example, the casting unit operation job- Type 1 requires 125 minutes, on the average. The operation times are all exponentially distributed.

Jobs arrive at the shop in a Poisson stream at a mean rate of 50 jobs per 8-hour day. Twenty-four percent of the jobs in this stream are of type 1, 44 percent are of type 2, and the rest are of type 3. Whether an arriving job is of type 1, 2 or 3 is independent of the job type of the preceeding arrival.

We are to build a model which simulates the operation of the production shop. At the end of the week print out the distribution of job-type, and the distribution of the total number of jobs in the shop, based on observations made at the end of each day during the week. Assume that the queue discipline used at each machine group is first-come first-served, independent of the job type. Assume there are no discontinuities in moving between consecutive 8-hour work days.

3.2.2 Discussion Of The Model - The general approach in building the job shop model is to be able to represent the model in graphical form. This is made possible by eliminating tables and matrices, avoiding the alphanumeric representation of data, using built-in

distribution functions and selecting meaningful names and labels so that the graphical model becomes self-explanatory. Figures 3-5a through 3-5e contain the BGSL model representing the job shop.

Six segments are used to build this model. The first segment is the definition segment, the second segment the control segment, and the remainder are model segments. One of the model segments represents the stop simulation (time out) clock, which terminates the simulation at the end of the week. The rest of the model segments represent the job types.

Figure 3-5a shows the definition segment and control segment. In this model storages are used to represent the machines in each workstation. Each storage command defines the number of machines in the corresponding workstation. Table commands define four tables which hold statistical data regarding the time spent in the job shop for each job type (T1, T2, T3), and the total number of jobs in the model at the end of the work day (Tjobs). In order to calculate the number of jobs in the model the "COUNT" variable is used. This variable calculates the number of jobs in the system by subtracting the number of jobs remaining in the model from the number of jobs that entered the model. For

this reason, the "N\$" Standard Numerical Attribute is used ("N\$"SNA counts the number of transactions passed through a labeled block) in conjunction with IN and OUT labels used at the representative blocks for entrance and exit of the job shop. The control segment contains the START command which initiates the termination counter to 5 (allowing 5 days of simulated operation).

Each job type has a separate GENERATE block and a separate model segment. These model segments represent the scheduled sequences which jobs will go through. Figure 3-5b shows the model segment which simulates the movement of JOB1 job type (workpiece) through the production shop. Figure 3-5c and 3-5d represent job2 and job3 job types. This segment uses a sequence of ENTER, ADVANCE and LEAVE blocks to simulate each workstation visited by the work piece.

In order to calculate the mean interarrival time for each job type, the mean interarrival time (for all of the job types) is multiplied by the percentages of each job type. This is used to define the mean time for GENERATE blocks. The exponential distribution function for job interarrival times and service times is implemented using the "DS\$EXPONENTIALF" attribute as the function modifier for the GENERATE blocks. Finally the

TABULATE block saves the residence time of each workpiece in the production shop.

Figure 3-6 contains the Block Command Symbolic representation of the production shop model. Each segment of the graphical model has a corresponding segment in BCSL and each line has the corresponding block number on it.

All blocks have default block numbers except IN and OUT blocks which have labels. In the section on advanced features of BCSL, the job shop model is built using a new block called "Request Work Station" which simplifies the model. This block is a macro block containing ENTER-ADVANCE-LEAVE blocks which makes the model smaller and more understandable.

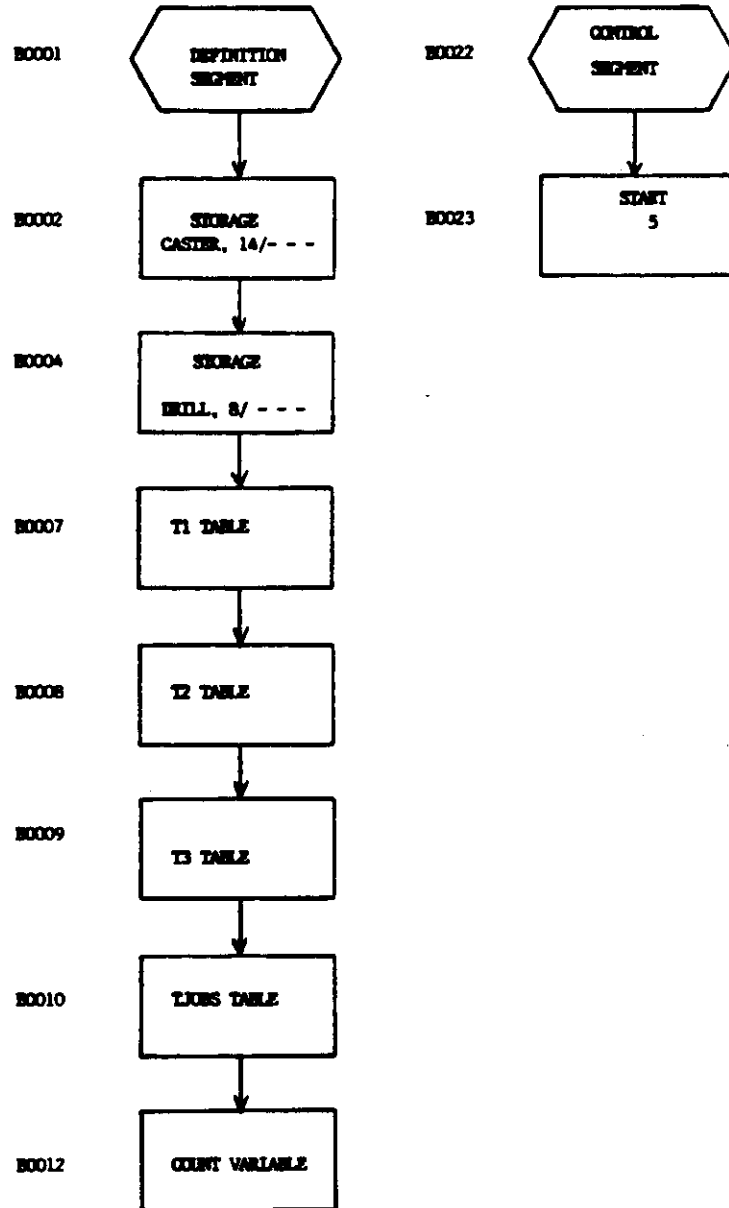


Figure 3-5a. BGSL Model for the Production Shop

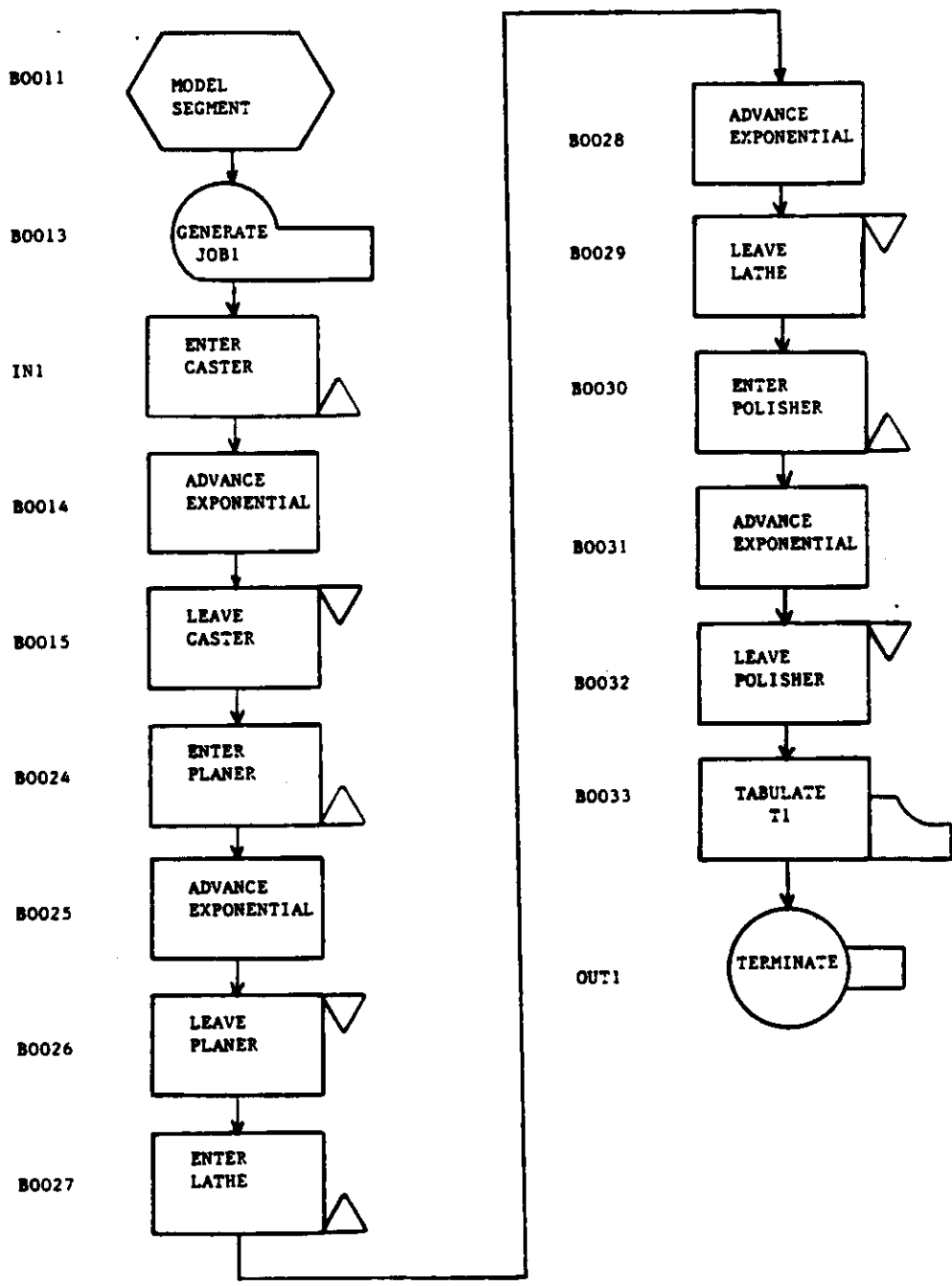


Figure 3-5b. BGSL Model for the Production Shop (Cont'd)

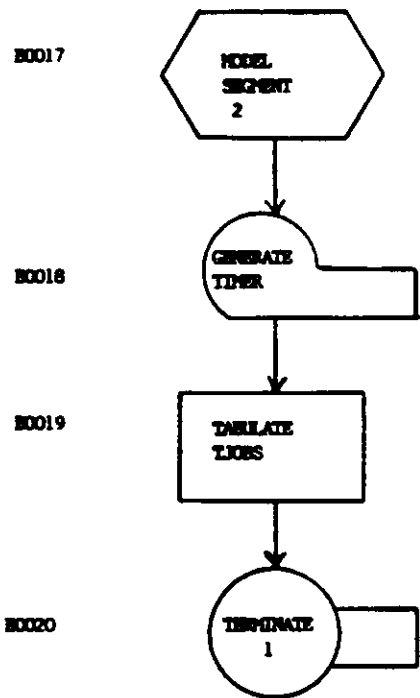


Figure 3-5c. BGSJ Model for the Production Shop (Cont'd)

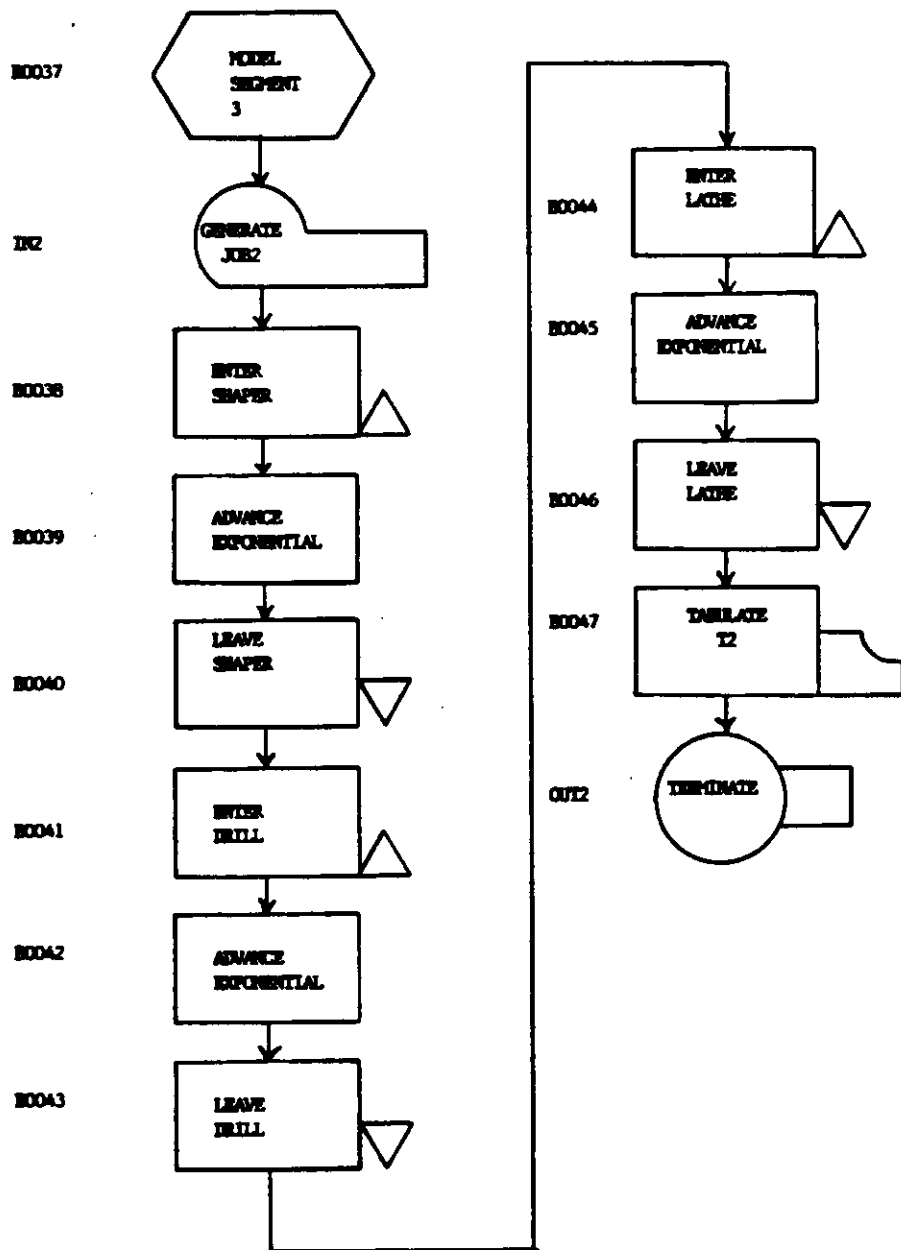


Figure 3-5d. BGS L Model for the Production Shop (Cont'd)

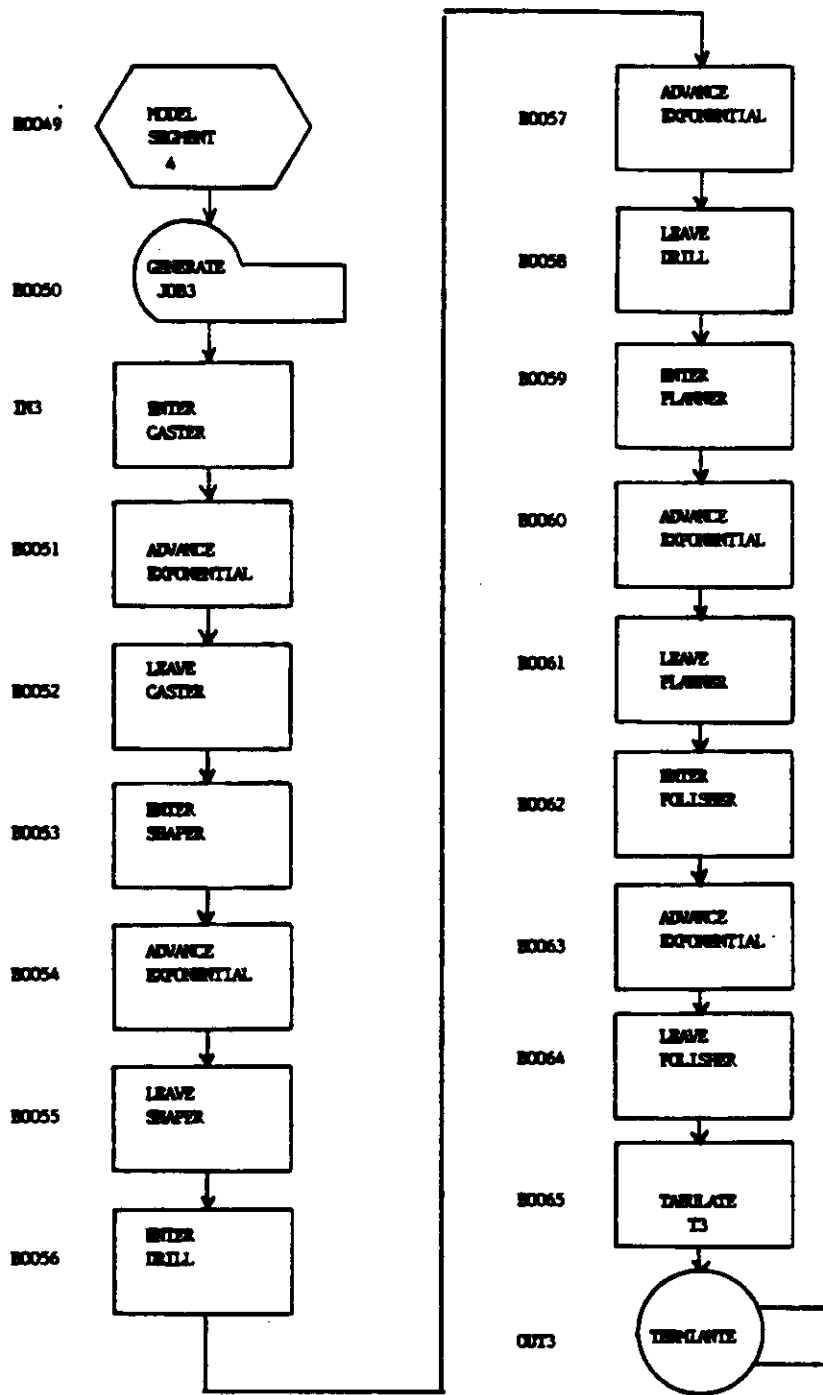


Figure 3-5e. BGS Model for the Production Shop (Cont'd)

```

*****
**
** BCSL CODE GENERATOR VERSION 1.0
**
*****
*
*      SIMULATE
*
*** DEFINITION SEGMENT
*
*      STORAGE      CASTER,14/LATHE,5/PLANEM,4      B0002
*      STORAGE      DRILL,8/SHAPE,16/POLISHER,4     B0004
11  TABLE          M1,1200,1200,10                 B0007
12  TABLE          M1,1200,1200,10                 B0008
13  TABLE          M1,1200,1200,10                 B0009
1JOBS TABLE       VSCOUNT,10,10,0                 B0010
CGUNT VARIABLE     MSIN1-MSOUT1+MSIN2-MSOUT2+MSIN3-MSOUT3
*
*
*** ME1 SEGMENT
*
*      GENERATE     300.,DSSEXPOENTIAL.F,,,,,JOB1
1E1  ENTER         CASTER,1                          IN1
      ADVANCE      1250.,DSSEXPOENTIAL.F             B0014
      LEAVE        CASTER,1                          B0015
      ENTER        PLANEM                             B0024
      ADVANCE      350.,DSSEXPOENTIAL.F             B0025
      LEAVE        PLANEM                             B0026
      ENTER        LATHE                              B0027
      ADVANCE      200.,DSSEXPOENTIAL.F             B0028
      LEAVE        LATHE                              B0029
      ENTER        POLISHER                           B0030
      ADVANCE      600.,DSSEXPOENTIAL.F             B0031
      LEAVE        POLISHER                           B0032
      TABULATE     T1                                 B0033
OUT1 TERMINATE    T1                                OB11
*
*
*** ME2 SEGMENT
*
*      GENERATE     4800.,,,,,,TIREM                B0016
      TABULATE     TJOBS                             B0019
      TERMINATE    1                                 B0020
*
*
*** ME3 SEGMENT
*
*      GENERATE     220.,DSSEXPOENTIAL.F,,,,,JOB2
IN2  ENTER        SHAPER                             IN2

```

Figure 3-6a. BCSL Model for the Procution Shop


```

      ADVANCE 1050.,DSSEXPNENTIAL.F      80038
      LEAVE  SHAPER      80039
      ENTER  DRILL      80040
      ADVANCE 900.,CSSEXPNENTIAL.F      80041
      LEAVE  DRILL      80042
      ENTER  LATHE      80043
      ADVANCE 650.,LSSEXPNENTIAL.F      80044
      LEAVE  LATHE      80045
      TABULATE 12      80046
OUT1 TERMINATE      0012
*
*
*** NEW SEGMENT
*
*
      IN3  GENERATE 286.,DSSEXPNENTIAL.F,,,,,JOB3
      ENTER  CASTER      IN3
      ADVANCE 2350.,DSSEXPNENTIAL.F      80051
      LEAVE  CASTER      80052
      ENTER  SHAPER      80053
      ADVANCE 2500.,DSSEXPNENTIAL.F      80054
      LEAVE  SHAPER      80055
      ENTER  DRILL      80056
      ADVANCE 500.,CSSEXPNENTIAL.F      80057
      LEAVE  DRILL      80058
      ENTER  PLANE      80059
      ADVANCE 300.,DSSEXPNENTIAL.F      80060
      LEAVE  PLANE      80061
      ENTER  POLISHER      80062
      ADVANCE 250.,CSSEXPNENTIAL.F      80063
      LEAVE  POLISHER      80064
      TABULATE 13      80065
OUT3 TERMINATE      0013
*
*
*** CONTROL SEGMENT
*
*
      START 5      80023
*
*
      END

```

Figure 3-6b. BCSL Model for the Procution Shop (Cont'd)

3.2.3 Discussion Of SIMSCRIPT Equivalent - The SIMSCRIPT equivalent model for the production shop consists of preamble, main, initialization, V\$COUNT (function) routine, generator processes and transaction processes. All the sections of the SIMSCRIPT program are discussed in the order in which they appear in the generated output file. As described in the description of the BGS L model, the ENTER, ADVANCE, and LEAVE block chain is used extensively to represent usage of machines. Translation of these blocks is similar to the translation of the SEIZE, ADVANCE, and RELEASE block chain discussed in the barber shop model. Figure 3-7 contains selected sections of the SIMSCRIPT equivalent program for the production shop model.

The preamble section defines all the processes where there is a generator process and a transaction process per job type. The transaction process includes TRAN.job1, TRAN.job2, TRAN.job3 and GEN.TIMER. Each transaction can be blocked by a GATE or TEST block; therefore transaction names could be added to the blocked transaction list for its corresponding transaction type. There is a set for each transaction types called BLOCKED.job1, BLOCKED.job2, BLOCKED.job3 and BLOCKED.TIMER.

The preamble also defines all of the resources in the model which include; CASTER, LATHE, PLANNER, DRILL, SHAPER and POLISHER work-stations. A series of definition statements define the variables representing the time spent in the queue, service time, number of people entering the queue and capacity of each resource.

If the user selects a label for a block, the current number of the transaction in the corresponding block is calculated. The job1 segment has IN1 and OUT1 as labels, used to calculate the number of jobs in the segment. The total number of transactions which pass the labeled block is collected using the integer variables N\$IN1, N\$OUT1 and the current number of transactions in the labeled block is calculated using the integer variable W\$IN1, W\$OUT1 for job1.

The stop flag is defined as an integer variable and V\$COUNT is defined as a function. The Pn and Xn variables are replaced by elements of P and X arrays. For example if the user selects the P1 parameter in the SIMSCRIPT model, it refers to P-ARRAY (1), the first element of P-ARRAY. The unit of time is defined as minutes. Each transaction type in the model has a transaction counter which is defined as an integer variable. The table entry variables T1, T2, T3 and

Tjobs are defined as real variables.

M1, the time spent in the model by a transaction is defined as a real variable and is calculated within the Transaction process. C1 represents the current time and is equivalent to the current time in days multiplied by the number of minutes in an hour and the number of hours in a day.

The ACCUMULATE and TALLY statements are generated to calculate average utilization of resources, maximum number of people in service, average number of jobs in queue and the maximum number of jobs in the queue for each resource. Each HISTOGRAM statement collects the distribution of the number of times that table entries fall within certain limits. The histogram limits are defined for T1, T2, T3 and Tjobs tables; where T1, T2 and T3 represent time spent in the model and tjobs collects the total number of jobs in the system at the end of each day.

The Main and Output routines consist of standard calls to other SIMSCRIPT routines. These routines have been discussed in previous sections. The initialization routine, defines the capacity of the resources and creates all the resources. These resources represent the work-stations where the number of machines in each

group defines the capacity of the corresponding resource. The initialize routine sets the transaction counter to zero and activates all of the process generator routines GEN.job1, GEN.job2, GEN.job3 and GEN.timer.

The special purpose output generation routines RESOURCE-PRINT, QUEUE-PRINT, TRANSACTION-PRINT, TABLE-PRINT, STAX-PRINT and XVAR-PRINT generate the output results using a series of WRITE statements with predefined formats.

The storage and table block commands do not generate any SIMSCRIPT code in Pass 1; therefore there is no corresponding SIMSCRIPT statement following each block in the SIMSCRIPT equivalent model. These blocks generate the data element for the set structures which is used in Pass 2 of the compiler.

The variable block command has caused the generation of a V\$COUNT routine to calculate corresponding variables. This routine acts like a function routine in FORTRAN and calculates the total number of jobs in the system at the end of the day by adding the number of jobs in each model segment. The number of jobs in each segment is calculated by subtracting the total number of jobs left in the segment

from the number of jobs entering the same segment.

A GENERATE block is translated into the generator process. The JOB1 generator uses an exponential distribution function with a mean time of 384 units for interarrival time of the jobs into the system. The generator process also collects the statistics on actual interarrival times and increments the transaction counter every time it activates a transaction process.

The Transaction process for JOB1 is called TRAN.job1 and follows the generator process in the SIMSCRIPT model. It defines the P-ARRAY as a local array and saves the arrival time of the transaction. The IN1 label causes the total number of transactions passed to the ENTER block (N\$IN1()) to be incremented and the current number of transactions in the ENTER block (W\$IN1) to be incremented. W\$IN1 will be decremented when the transaction reaches the next block (ADVANCE). The ENTER block is translated into a REQUEST statement for a casting unit. The ENTER block translation also collects statistics for casting units queue and service time. The ADVANCE block (block number 14) translates into a WAIT statement using the transaction seed defined earlier. Translation of the LEAVE block (block number 15) relinquishes the casting unit and calculates the

time in service for the casting unit.

The transaction will continue its move in the model by entering the next workstation, this process continues as the transaction passes through the chains of the ENTER-ADVANCE-LEAVE blocks.

The transaction finally enters the TABULATE block and leaves the model via the TERMINATE block.

The TABULATE block translation first calculates the "M1" Standard Numerical Attribute (representing the time spent in the model by a transaction), then includes its results in the corresponding table. The T1 table represents the time spent in the JOB1 process by the transactions, T2 represents JOB2 and T3 represents JOB 3. This way every time a transaction reaches the TABULATE block, the M1 SNA is calculated despite the fact that it is also calculated by the TERMINATE block in the same segment when a transaction leaves the model.

The reason for recalculating the "M1" when it is used by a TABULATE block is that the "M1" SNA, which is calculated by the TERMINATE block, belongs to a transaction which has already left the model (last transaction). Therefore, there is no calculated M1 available for the first transaction when it arrives at

the TABULATE block, because this transaction has not yet reached the TERMINATE block. As a result, if M12 is not calculated by the TABULATE block, the first M1 used in the table would always have a wrong value and the last M1 generated could never be included in the table.

The OUT1 label translation calculates the total number of transactions which have left the model. The process actually terminates by a RETURN statement which returns the control back to the generator process and will cause the temporary element representing this job transaction process to be deleted from SIMSCRIPT's internal data base.

The generator process and transaction process translation for JOB2 and JOB3 are similar to JOB1 transactions discussed above.

The TIMER generator process will activate a TIMER transaction after a week. The TIMER transaction tabulates the number of transactions remaining in the production shop at closing time by calculating the V\$COUNT variable using the V\$COUNT function and storing it in Tjobs Table. The TIMER transaction will decrement the STOP counter for this simulation run by one, call the output generator routine, and stop the simulation if the counter is zero.

3.2.4 Discussion Of The Results - The generated output report includes an analysis of each resource utilization and their corresponding queue statistics. The resources in the production shop model are the machines in each work station. In GPSS terminology a resource with capacity larger than one is referred to as a storage and in order to stay compatible with GPSSV, this terminology has been used for report generation.

Figure 3-8 shows the BGSL simulation run results for the production shop model. In this model, the unit of time is a 10th of a minute (six seconds). The generated report shows that even though the average number of jobs in the queues is small, the maximum number of jobs in the lathe and drill queues are relatively high. The number of created transactions and average transaction interarrival times are displayed for each job type. The collected statistics for time spent in the model for each job type is displayed for T1, T2 and T3 tables. The collected statistics for the number of jobs in the model is also included in this report in TJOBS table. In order to provide a comparison between the GPSSV and the BGSL models, the simulation run results for the same production shop developed in GPSS are included in Figure 3-8.

Given the fact that the simulation results are dependent on the interarrival times and service times which in turn are dependent on the random number generation method and the seeds used to generate the numbers, we can say that the simulation results for the BGSL and GPSS models are very close to each other.

The correctness of the BGSL model and the translation of this model have been confirmed by running the production shop model several times using different seeds and comparing the results with the models built in GPSS, SIMSCRIPT and SIMAN [36] for the same production shop. Furthermore, traceback routines and commands to print out intermediate states of the model have been used to prove the correctness of the generated SIMSCRIPT model for the production shop.

```

*.....*
*.....*
*.....* SIMSCRIPT CODE GENERATED BY .....*
*.....*..... BCSL TO SIMSCRIPT CROSS COMPILER .....*
*.....*..... VERSION 1.2 .....*
*.....*
1 REFABLE
2 NORMALLY MODE IS REAL
3 PROCESSES
4 EVERY TRAM.JOB1 HAS A MONITOR.V
5 AND MAY BELONG TO THE BLOCKED.JOB1
6 EVERY TRAM.TIMER HAS A MONITOR.V
7 AND MAY BELONG TO THE BLOCKED.TIMER
8 EVERY TRAM.JOB2 HAS A MONITOR.V
9 AND MAY BELONG TO THE BLOCKED.JOB2
10 EVERY TRAM.JOB3 HAS A MONITOR.V
11 AND MAY BELONG TO THE BLOCKED.JOB3
12 THE SYSTEM OWNS THE BLOCKED.JOB1
13 THE SYSTEM OWNS THE BLOCKED.TIMER
14 THE SYSTEM OWNS THE BLOCKED.JOB2
15 THE SYSTEM OWNS THE BLOCKED.JOB3
16 EVERY GEN.JOB1 HAS A STOP.FLAG.JOB1
17 EVERY GEN.TIMER HAS A STOP.FLAG.TIMER
18 EVERY GEN.JOB2 HAS A STOP.FLAG.JOB2
19 EVERY GEN.JOB3 HAS A STOP.FLAG.JOB3
20 RESOURCES
21 EVERY CASTER HAS A NUM.OF.CASTER
22 EVERY LATHE HAS A NUM.OF.LATHE
23 EVERY PLANE HAS A NUM.OF.PLANE
24 EVERY DRILL HAS A NUM.OF.DRILL
25 EVERY SHAPER HAS A NUM.OF.SHAPER
26 EVERY POLISHER HAS A NUM.OF.POLISHER
27 DEFINE TIMEIN.F.CASTER AS A REAL VARIABLE
28 DEFINE TIMEIN.Q.CASTER AS A REAL VARIABLE
29 DEFINE NUM.ENT.Q.CASTER AS AN INTEGER VARIABLE
30 DEFINE CAPACITY.CASTER AS AN INTEGER VARIABLE
31 DEFINE TIMEIN.F.LATHE AS A REAL VARIABLE
32 DEFINE TIMEIN.Q.LATHE AS A REAL VARIABLE
33 DEFINE NUM.ENT.Q.LATHE AS AN INTEGER VARIABLE
34 DEFINE CAPACITY.LATHE AS AN INTEGER VARIABLE
35 DEFINE TIMEIN.F.PLANE AS A REAL VARIABLE
36 DEFINE TIMEIN.Q.PLANE AS A REAL VARIABLE
37 DEFINE NUM.ENT.Q.PLANE AS AN INTEGER VARIABLE
38 DEFINE CAPACITY.PLANE AS AN INTEGER VARIABLE
39 DEFINE TIMEIN.F.DRILL AS A REAL VARIABLE
40 DEFINE TIMEIN.Q.DRILL AS A REAL VARIABLE
41 DEFINE NUM.ENT.Q.DRILL AS AN INTEGER VARIABLE
42 DEFINE CAPACITY.DRILL AS AN INTEGER VARIABLE
43 DEFINE TIMEIN.F.SHAPER AS A REAL VARIABLE
44 DEFINE TIMEIN.Q.SHAPER AS A REAL VARIABLE
45 DEFINE NUM.ENT.Q.SHAPER AS AN INTEGER VARIABLE
46 DEFINE CAPACITY.SHAPER AS AN INTEGER VARIABLE
47 DEFINE TIMEIN.F.POLISHER AS A REAL VARIABLE
48 DEFINE TIMEIN.Q.POLISHER AS A REAL VARIABLE
  
```

Figure 3-7a. SIMSCRIPT Model for the Production Shop

```

49 DEFINE NUM.EN1.L.POLISHER AS AN INTEGER VARIABLE
50 DEFINE CAPACITY.POLISHER AS AN INTEGER VARIABLE
51 **DEFAULT VARIABLES AND LEFT MONITORED VARIABLES
52 DEFINE NSIN1 AS INTEGER VARIABLE
53 DEFINE NSIN1 AS INTEGER VARIABLE
54 DEFINE NSOUT1 AS INTEGER VARIABLE
55 DEFINE NSOUT1 AS INTEGER VARIABLE
56 DEFINE NSIN2 AS INTEGER VARIABLE
57 DEFINE NSIN2 AS INTEGER VARIABLE
58 DEFINE NSOUT2 AS INTEGER VARIABLE
59 DEFINE NSOUT2 AS INTEGER VARIABLE
60 DEFINE NSIN3 AS INTEGER VARIABLE
61 DEFINE NSIN3 AS INTEGER VARIABLE
62 DEFINE NSOUT3 AS INTEGER VARIABLE
63 DEFINE NSOUT3 AS INTEGER VARIABLE
64 DEFINE STOP.FLAG AS INTEGER VARIABLE
65 DEFINE VSCOUNT AS INTEGER FUNCTION
66 **DEFINE SMA FOR WDEFES
67 **DEFINE TEST BLOCK FLAGS
68 **DEFINE STATISTICAL VARIABLES
69 **DEFINE HANDOFF VARIABLES
70 **DEFINE NATICES
71 **DEFINE GPSS GLOBAL VARIABLES
72 DEFINE P1 TO HEAD P_ARRAY(1)
73 DEFINE P2 TO HEAD P_ARRAY(2)
74 DEFINE P3 TO HEAD P_ARRAY(3)
75 DEFINE P4 TO HEAD P_ARRAY(4)
76 DEFINE P5 TO HEAD P_ARRAY(5)
77 DEFINE P6 TO HEAD P_ARRAY(6)
78 DEFINE P7 TO HEAD P_ARRAY(7)
79 DEFINE P8 TO HEAD P_ARRAY(8)
80 DEFINE P9 TO HEAD P_ARRAY(9)
81 DEFINE P10 TO HEAD P_ARRAY(10)
82 DEFINE X1 TO HEAD X_ARRAY(1)
83 DEFINE X2 TO HEAD X_ARRAY(2)
84 DEFINE X3 TO HEAD X_ARRAY(3)
85 DEFINE X4 TO HEAD X_ARRAY(4)
86 DEFINE X5 TO HEAD X_ARRAY(5)
87 DEFINE X6 TO HEAD X_ARRAY(6)
88 DEFINE X7 TO HEAD X_ARRAY(7)
89 DEFINE X8 TO HEAD X_ARRAY(8)
90 DEFINE X9 TO HEAD X_ARRAY(9)
91 DEFINE X10 TO HEAD X_ARRAY(10)
92 DEFINE UNITS TO HEAD MINUTES
93 DEFINE MONITOR.V AS TEXT VARIABLE
94 DEFINE M1 AS A REAL VARIABLE
95 DEFINE C1 TO HEAD TIME.V*HOURS.V*MINUTES.V
96 **TRANSACTION DEFINITION
97 DEFINE TRAN.CTR.JOB1 AS AN INTEGER VARIABLE
98 DEFINE C.TRAN.CNT.JOB1 AS AN INTEGER VARIABLE
99 DEFINE SEED.TRAN.JOB1 AS AN INTEGER VARIABLE
100 DEFINE TRAN.CTR.TIME1 AS AN INTEGER VARIABLE
101 DEFINE C.TRAN.CNT.TIME1 AS AN INTEGER VARIABLE
102 DEFINE SEED.TRAN.TIME1 AS AN INTEGER VARIABLE
103 DEFINE TRAN.CTR.JOB2 AS AN INTEGER VARIABLE
104 DEFINE C.TRAN.CNT.JOB2 AS AN INTEGER VARIABLE
  
```

Figure 3-7a. SIMSCRIPT Model for the Production Shop (Cont'd)

```

105 DEFINE SEED.TRAN.JOB2 AS AN INTEGER VARIABLE
106 DEFINE MAX.CNTR.JOB3 AS AN INTEGER VARIABLE
107 DEFINE C.TRAN.CMT.JOB3 AS AN INTEGER VARIABLE
108 DEFINE SEED.TRAN.JOB3 AS AN INTEGER VARIABLE
109 **DEFINE TABLE VARIABLES
110 DEFINE T1 AS REAL VARIABLE
111 DEFINE T2 AS REAL VARIABLE
112 DEFINE T3 AS REAL VARIABLE
113 DEFINE TJOBS AS REAL VARIABLE
114 **TALLY AND ACCUMULATE FOR RESOURCES
115 ACCUMULATE UTIL.CASTER AS AVERAGE AND
116 MAX.MX.CASTER AS MAXIMUM OF M.X.CASTER
117 ACCUMULATE AVG.Q.LEN.CASTER AS AVERAGE AND
118 MAX.Q.LEN.CASTER AS MAXIMUM OF M.Q.CASTER
119 TALLY AVG.TIMEIN.F.CASTER AS MEAN OF TIMEIN.F.CASTER
120 TALLY AVG.TIMEIN.Q.CASTER AS MEAN OF TIMEIN.Q.CASTER
121 ACCUMULATE UTIL.LATHE AS AVERAGE AND
122 MAX.MX.LATHE AS MAXIMUM OF M.X.LATHE
123 ACCUMULATE AVG.Q.LEN.LATHE AS AVERAGE AND
124 MAX.Q.LEN.LATHE AS MAXIMUM OF M.Q.LATHE
125 TALLY AVG.TIMEIN.F.LATHE AS MEAN OF TIMEIN.F.LATHE
126 TALLY AVG.TIMEIN.Q.LATHE AS MEAN OF TIMEIN.Q.LATHE
127 ACCUMULATE UTIL.PLANE AS AVERAGE AND
128 MAX.MX.PLANE AS MAXIMUM OF M.X.PLANE
129 ACCUMULATE AVG.Q.LEN.PLANE AS AVERAGE AND
130 MAX.Q.LEN.PLANE AS MAXIMUM OF M.Q.PLANE
131 TALLY AVG.TIMEIN.F.PLANE AS MEAN OF TIMEIN.F.PLANE
132 TALLY AVG.TIMEIN.Q.PLANE AS MEAN OF TIMEIN.Q.PLANE
133 ACCUMULATE UTIL.DRILL AS AVERAGE AND
134 MAX.MX.DRILL AS MAXIMUM OF M.X.DRILL
135 ACCUMULATE AVG.Q.LEN.DRILL AS AVERAGE AND
136 MAX.Q.LEN.DRILL AS MAXIMUM OF M.Q.DRILL
137 TALLY AVG.TIMEIN.F.DRILL AS MEAN OF TIMEIN.F.DRILL
138 TALLY AVG.TIMEIN.Q.DRILL AS MEAN OF TIMEIN.Q.DRILL
139 ACCUMULATE UTIL.SHAPE AS AVERAGE AND
140 MAX.MX.SHAPE AS MAXIMUM OF M.X.SHAPE
141 ACCUMULATE AVG.Q.LEN.SHAPE AS AVERAGE AND
142 MAX.Q.LEN.SHAPE AS MAXIMUM OF M.Q.SHAPE
143 TALLY AVG.TIMEIN.F.SHAPE AS MEAN OF TIMEIN.F.SHAPE
144 TALLY AVG.TIMEIN.Q.SHAPE AS MEAN OF TIMEIN.Q.SHAPE
145 ACCUMULATE UTIL.POLISH AS AVERAGE AND
146 MAX.MX.POLISH AS MAXIMUM OF M.X.POLISH
147 ACCUMULATE AVG.Q.LEN.POLISH AS AVERAGE AND
148 MAX.Q.LEN.POLISH AS MAXIMUM OF M.Q.POLISH
149 TALLY AVG.TIMEIN.F.POLISH AS MEAN OF TIMEIN.F.POLISH
150 TALLY AVG.TIMEIN.Q.POLISH AS MEAN OF TIMEIN.Q.POLISH
151 **TALLY AND ACCUMULATE FOR STATISTICAL VARIABLES
152 **TALLY AND ACCUMULATE FOR JOBS
153 DEFINE GEN.TIME.JOB1 AS A REAL VARIABLE
154 TALLY AVG.GEN.TIME.JOB1 AS MEAN OF GEN.TIME.JOB1
155 DEFINE GEN.TIME.TIMER AS A REAL VARIABLE
156 TALLY AVG.GEN.TIME.TIMER AS MEAN OF GEN.TIME.TIMER
157 DEFINE GEN.TIME.JOB2 AS A REAL VARIABLE
158 TALLY AVG.GEN.TIME.JOB2 AS MEAN OF GEN.TIME.JOB2
159 DEFINE GEN.TIME.JOB3 AS A REAL VARIABLE
160 TALLY AVG.GEN.TIME.JOB3 AS MEAN OF GEN.TIME.JOB3
  
```

Figure 3-7a. SIMSCRIPT Model for the Production Shop (Cont'd)

```

161 **HISTOGRAM FOR TABLES
162 TALLY T1.HISTO(1200 10 10000 BY 1200)
163 AS THE HISTOGRAM AND T1.AVG AS AVERAGE AND
164 T1.STD AS STD.DEV AND T1.NUM.EMT AS NUMBER OF T1
165 TALLY T2.HISTO(1200 10 10000 BY 1200)
166 AS THE HISTOGRAM AND T2.AVG AS AVERAGE AND
167 T2.STD AS STD.DEV AND T2.NUM.EMT AS NUMBER OF T2
168 TALLY T3.HISTO(1200 10 10000 BY 1200)
169 AS THE HISTOGRAM AND T3.AVG AS AVERAGE AND
170 T3.STD AS STD.DEV AND T3.NUM.EMT AS NUMBER OF T3
171 TALLY TJOBS.HISTO(10 10 50 BY 10)
172 AS THE HISTOGRAM AND TJOBS.AVG AS AVERAGE AND
173 TJOBS.STD AS STD.DEV AND TJOBS.NUM.EMT AS NUMBER OF TJOBS
174 END
  
```

C R O S S - R E F E R E N C E

NAME	TYPE	MODE
AVG.GEN.TIME.JOB1	ROUTINE	DOUBLE
AVG.GEN.TIME.JOB2	ROUTINE	DOUBLE
AVG.GEN.TIME.JOB3	ROUTINE	DOUBLE
AVG.GEN.TIME.TIMER	ROUTINE	DOUBLE
AVG.Q.LEN.CASTER	FUNCTION ATTRIBUTE	DOUBLE
AVG.Q.LEN.DRILL	FUNCTION ATTRIBUTE	DOUBLE
AVG.Q.LEN.LATHE	FUNCTION ATTRIBUTE	DOUBLE
AVG.Q.LEN.PLANER	FUNCTION ATTRIBUTE	DOUBLE
AVG.Q.LEN.POLISHER	FUNCTION ATTRIBUTE	DOUBLE
AVG.Q.LEN.SHAPE	FUNCTION ATTRIBUTE	DOUBLE
AVG.TIMEIN.F.CASTER	ROUTINE	DOUBLE
AVG.TIMEIN.F.DRILL	ROUTINE	DOUBLE
AVG.TIMEIN.F.LATHE	ROUTINE	DOUBLE
AVG.TIMEIN.F.PLANER	ROUTINE	DOUBLE
AVG.TIMEIN.F.POLISHER	ROUTINE	DOUBLE
AVG.TIMEIN.F.SHAPE	ROUTINE	DOUBLE
AVG.TIMEIN.Q.CASTER	ROUTINE	DOUBLE
AVG.TIMEIN.Q.DRILL	ROUTINE	DOUBLE
AVG.TIMEIN.Q.LATHE	ROUTINE	DOUBLE
AVG.TIMEIN.Q.PLANER	ROUTINE	DOUBLE
AVG.TIMEIN.Q.POLISHER	ROUTINE	DOUBLE
AVG.TIMEIN.Q.SHAPE	ROUTINE	DOUBLE
BLOCKED.JOB1	SET	
BLOCKED.JOB2	SET	
BLOCKED.JOB3	SET	
BLOCKED.TIME	SET	
C.TRAN.CNT.JOB1	GLOBAL VARIABLE	INTEGER
C.TRAN.CNT.JOB2	GLOBAL VARIABLE	INTEGER
C.TRAN.CNT.JOB3	GLOBAL VARIABLE	INTEGER
C.TRAN.CNT.LINEA	GLOBAL VARIABLE	INTEGER
CAPACITY.CASTER	GLOBAL VARIABLE	INTEGER
CAPACITY.DRILL	GLOBAL VARIABLE	INTEGER
CAPACITY.LATHE	GLOBAL VARIABLE	INTEGER
CAPACITY.PLANER	GLOBAL VARIABLE	INTEGER
CAPACITY.POLISHER	GLOBAL VARIABLE	INTEGER
CAPACITY.SHAPE	GLOBAL VARIABLE	INTEGER

Figure 3-7a. SIMSCRIPT Model for the Production Shop (Cont'd)

```

                                CACI SIMSCRIPT II.5 IBM S/370 49.3      PAGE
OPTIONS  TERM,LOAD,LD,TRACE2,NOTERS,CNK,REN=REN      19-NOV-1985 22:36 ( 1
1  MAIN
2  CALL INITIALIZE
3  START SIMULATION
4  CALL OUIPOL_ROUTINE
5  END

```

```

                                CACI SIMSCRIPT II.5 IBM S/370 49.3      PAGE
OPTIONS  TERM,LOAD,LD,TRACE2,NOTERS,CNK,REN=REN      19-NOV-1985 22:36 ( 1
1  ROUTINE INITIALIZE
2  NORMALLY MODE IS INTEGER
3  LET CAPACITY.CASTER = 14
4  CREATE EVERY CASTER(1)
5  LET U.CASTER(1) = 14
6  LET CAPACITY.LATHE = 5
7  CREATE EVERY LATHE(1)
8  LET U.LATHE(1) = 5
9  LET CAPACITY.PLANER = 4
10 CREATE EVERY PLANER(1)
11 LET U.PLANER(1) = 4
12 LET CAPACITY.DRILL = 8
13 CREATE EVERY DRILL(1)
14 LET U.DRILL(1) = 8
15 LET CAPACITY.SHAPER = 16
16 CREATE EVERY SHAPER(1)
17 LET U.SHAPER(1) = 16
18 LET CAPACITY.POLISHER = 4
19 CREATE EVERY POLISHER(1)
20 LET U.POLISHER(1) = 4
21 LET TRAN.CNTR.JOB1 = 0
22 LET TRAN.CNTR.TISER = 0
23 LET TRAN.CNTR.JOB2 = 0
24 LET TRAN.CNTR.JOB3 = 0
25 LET STOP.FLAG = 5
26 ACTIVATE A GEN.JOB1 NOW
27 ACTIVATE A GEN.TISER NOW
28 ACTIVATE A GEN.JOB2 NOW
29 ACTIVATE A GEN.JOB3 NOW
30 END

```

Figure 3-7b. SIMSCRIPT Model for the Production Shop (Cont'd)

```

*****
***
**** FCSL CODE GENERATOR VERSION 1.0
***
*****
***
***
**      SIMULATE
***
***
**** DEFINITION SEGMENT
***
***
**      STORAGE      CASTEL,14/LATHE,5/PLANNER,4          B0002
**      STORAGE      DRILL,8/SHAPE,16/POLISHER,4          B0004
** T1      TABLE    H1,1200,1200,10                      B0007
** I2      TABLE    H1,1200,1200,10                      B0008
** I3      TABLE    H1,1200,1200,10                      B0009
** TJOES TABLE    VSCOUNT,10,10,6                       B0010
** COUNT VARIABLE   NSIN1-NSOUT1+NSIN2-NSOUT2+NSIN3-NSOUT3
1  ROUTINE VSCOUNT
2  LEI VS_COUNT = NSIN1-NSOUT1+NSIN2-NSOUT2+NSIN3-NSOUT3
3  RETURN WITH VS_COUNT
4  END

```

Figure 3-7c. SIMSCRIPT Model for the Production Shop (Cont'd)


```

***
***
**** NEB SEGMENT
***
***
**      GENERATE 300.,DSSEXPONENTIAL.F,,,,,JOB1
1  PROCESS GEN.JOB1
2  LET STOP.FLAG.JOB1(GEN.JOB1) = 1
3  LET SEPD.TIME.JOB1 = 1
4  WAIT 0 UNITS
5  UNTIL STOP.FLAG.JOB1(GEN.JOB1) <= 0
6  DO
7  DEFINE GEN1.JOB1 AS A REAL VARIABLE
8  LET GEN1.JOB1 = TIME.V
9  WAIT EXPONENTIAL.F (300.,
10 SEPD.TIME.JOB1) UNITS
11 LET GEN.TIME.JOB1=(TIME.V - GEN1.JOB1)*HOURS.V*MINUTES.V
12 ACTIVATE A TRM.JOB1 NOW
13 LET TRM.CNT.JOB1=TRM.CNT.JOB1 + 1
14 LET C.TRM.CNT.JOB1=C.TRM.CNT.JOB1 + 1
15 LOOP
16 END

```

Figure 3-7d. SIMSCRIPT Model for the Production Shop (Cont'd)

```

1  PROCESS TRAN.JOB1
2  DEFINE P_ARRAY AS INTEGER,1-DIMENSIONAL ARRAY
3  RESERVE P_ARRAY AS 10
4  DEFINE TIM.M1 AS A REAL VARIABLE
5  TIME.M1 = TIME.V
6  ** IN1  EN1E1      CASIER,1
7
8  *IN1 *
9  LET NSIN1 = NSIN1 + 1
10 LET MSIN1 = MSIN1 + 1
11 LET TIM.Q.CASIER = TIME.V
12 LET NUM.ENT.Q.CASIER = NUM.ENT.Q.CASIER + 1
13 REQUEST 1 CASIER
14 LET TIMEIN.Q.CASIER=(TIME.V - TIM.Q.CASIER)*HOURS.V*MINUTES.V
15 LET TIM.F.CASIER = TIME.V
16 **      ADVANCE   1250.,DSSEXPOENTIAL.F
17
18 LET MSIN1 = MSIN1 - 1
19 WORK EXPONENTIAL.P(1250.,SEED.TRAN.JOB1) UNITS
20 **      LEAVE    CASIER,1
21
22 RELINQUISH 1 CASIER
23 LET TIMEIN.F.CASIER=(TIME.V - TIM.F.CASIER)*HOURS.V*MINUTES.V
24 **      EN1E1    PLANE1
25
26 LET TIM.Q.PLANE1 = TIME.V
27 LET NUM.ENT.Q.PLANE1 = NUM.ENT.Q.PLANE1 + 1
28 REQUEST 1 PLANE1
29 LET TIMEIN.Q.PLANE1=(TIME.V - TIM.Q.PLANE1)*HOURS.V*MINUTES.V
30 LET TIM.F.PLANE1 = TIME.V
31 **      ADVANCE   350.,DSSEXPOENTIAL.F
32
33 WORK EXPONENTIAL.P(350.,SEED.TRAN.JOB1) UNITS
34 **      LEAVE    PLANE1
35
36 RELINQUISH 1 PLANE1
37 LET TIMEIN.F.PLANE1=(TIME.V - TIM.F.PLANE1)*HOURS.V*MINUTES.V
38 **      EN1E1    LATHE
39
40 LET TIM.Q.LATHE = TIME.V
41 LET NUM.ENT.Q.LATHE = NUM.ENT.Q.LATHE + 1
42 REQUEST 1 LATHE
43 LET TIMEIN.Q.LATHE=(TIME.V - TIM.Q.LATHE)*HOURS.V*MINUTES.V
44 LET TIM.F.LATHE = TIME.V
45 **      ADVANCE   200.,DSSEXPOENTIAL.F
46
47 WORK EXPONENTIAL.F(200.,SEED.TRAN.JOB1) UNITS
48 **      LEAVE    LATHE
49
50 RELINQUISH 1 LATHE
51 LET TIMEIN.F.LATHE=(TIME.V - TIM.F.LATHE)*HOURS.V*MINUTES.V
52 **      EN1E1    POLISHER
53
54 LET TIM.Q.POLISHER = TIME.V
55 LET NUM.ENT.Q.POLISHER = NUM.ENT.Q.POLISHER + 1
56 REQUEST 1 POLISHER

```

Figure 3-7e. SIMSCRIPT Model for the Production Shop (Cont'd)

```
57 LET TIMEIN.W.POLISHER=(TIME.V - TIN.W.POLISHER)*HOURS.V*MINUTES.V
58 LET TIME.P.POLISHER = TIME.V
59 ** ADVANCE 600.,DSSEXPOENTIAL.F                      B0031
60
61 WORK EXPONENTIAL.F(600.,SEED.TRAN.JOB1) UNTILS
62 ** LEAVE POLISHER                      B0032
63
64 RELINQUISH 1 POLISHER
65 LET TIMEIN.P.POLISHER=(TIME.V - TIN.P.POLISHER)*HOURS.V*MINUTES.V
66 ** TABULATE 11                      B0033
67
68 LET M1=(TIME.V - TIN.M1)*HOURS.V*MINUTES.V
69 LET I1 = M1
70 ** OUT1 TERMINATE                      OUT1
71
72 'OUT1 '
73 LET NSOUT1 = NSOUT1 + 1
74 LET WSCOUT1 = WSCOUT1 + 1
75 LET M1=(TIME.V - TIN.M1)*HOURS.V*MINUTES.V
76 LET C.TRAN.CH1.JOB1 = C.TRAN.CH1.JOB1 - 1
77 RETURN
78 ***
79
80 ***
81
82 ***** MLL SEGMENT
83
84 ***
85
86 ***
87
88 ** GENERATE 4800.,.....,TIMER                      B0018
89
90 END
```

Figure 3-7f. SIMSCRIPT Model for the Production Shop (Cont'd)

CACI SIMSCRIPT II.5 IBM S/370 49.3 PAGE 2
 OPTIONS TERM,LOAD,LD,TRACE2,NOTEM,CHK,REN=NEW 19-NOV-1965 22:36 (1

```

1 PROCESS GEN.TIMER
2 LET STOP.FLAG.TIMER(GEN.TIMER) = 1
3 LET SEED.TRAN.TIMER = 1
4 WAIT 0 UNITS
5 UNTIL STOP.FLAG.TIMER(GEN.TIMER) <= 0
6 DO
7 DEFINE GEN1.TIMER AS A REAL VARIABLE
8 LET GEN1.TIMER = TIME.V
9 WAIT UNIFORM.F((4800.-6)/1.0,(4800.+0)/1.0,
10 SEED.TRAN.TIMER) UNITS
11 LET GEN.TIME.TIMER=(TIME.V - GEN1.TIMER)*HOURS.V*MINUTES.V
12 ACTIVATE A TRAN.LINE NOW
13 LET TRAN.CNT.TIMER=TRAN.CNT.TIMER + 1
14 LP C.TRAN.CNT.TIMER=C.TRAN.CNT.TIMER + 1
15 LOOP
16 END

```

CACI SIMSCRIPT II.5 IBM S/370 49.3 PAGE 2
 OPTIONS TERM,LOAD,LD,TRACE2,NOTEM,CHK,REN=NEW 19-NOV-1965 22:36 (1

```

1 PROCESS TRAN.TIMER
2 DEFINE P_ARRAY AS INTEGER,1-DIMENSIONAL ARRAY
3 RESERVE P_ARRAY AS 10
4 DEFINE TIN.N1 AS A REAL VARIABLE
5 TIN.N1 = TIME.V
6 ** TABULATE TJOBS B0019
7
8 LET TJOBS = VSCOUNT
9 ** TERMINATE 1 B0020
10
11 LET N1 = (TIME.V - TIN.N1)*HOURS.V*MINUTES.V
12 LET C.TRAN.CNT.TIMER = C.TRAN.CNT.TIMER - 1
13 LET STOP.FLAG = STOP.FLAG - 1
14 IF STOP.FLAG LE 0
15 CALL OUTPUT_ROUTINE
16 STOP
17 ALWAYS
18 RETURN
19 ***
20
21 ***
22
23 **** NEW SEGMENT
24
25 ***
26
27 ***
28
29 ** GENERATE 220.,DSSRI%CENTRAL.P,,,,,JOB2
30
31 END

```

Figure 3-7g. SIMSCRIPT Model for the Production Shop (Cont'd)

 *
 ** GRAPHICAL SIMULATION SYSTEM OUTPUT **
 *

FACILITY NAME	CAPACITY	AVERAGE UTILIZATION	NUMBER OF ENTRIES	AVERAGE TIME/TAN	CURRENT CONTENT	MAXIMUM CONTENT
---------------	----------	---------------------	-------------------	------------------	-----------------	-----------------

STORAGE NAME	CAPACITY	AVERAGE UTILIZATION	NUMBER OF ENTRIES	AVERAGE TIME/TAN	CURRENT CONTENT	MAXIMUM CONTENT
CASSET	14	.810	164	1652.142	14	14
LATHE	5	.652	171	457.415	4	5
PLANE	4	.509	140	341.454	4	4
DRIIL	8	.733	176	782.383	8	8
SHAPE	16	.726	189	1424.493	10	16
POLISHER	4	.618	136	433.793	3	4

RESOURCE QUEUE	AVERAGE CONTENT	TOTAL ENTRIES	AVERAGE TIME/TAN	CURRENT CONTENT	MAXIMUM CONTENTS
CASSET	1.193	164	174.589	0	7
LATHE	.740	171	103.915	0	9
PLANE	.234	142	37.740	2	4
DRIIL	1.273	179	168.519	3	13
SHAPE	.150	189	19.001	0	4
POLISHER	.727	136	128.289	0	6

TRANSACTION NAME	NUMBER CREATED	AVERAGE CREATION TIME
JOB1	82	291.670
TIMER	5	4799.999
JOB2	114	208.421
JOB3	82	291.567

TABLE	AVERAGE	STD. DEV	NO. ENTRIES
T1	3010.43	1433.63	68

Figure 3-8a. Simulation Run Results for Production Shop

UPPER LIMIT	FREQUENCY	PERCENT OF TOTAL
1200	28	.41
2400	22	.32
3600	9	.13
4800	7	.10
6000	1	.01
7200	1	.01
8400	0	0.
9600	0	0.
10800	0	0.

TABLE	AVERAGE	STD. DEV	NO. ENTRIES
T2	2711.58	1441.19	97

UPPER LIMIT	FREQUENCY	PERCENT OF TOTAL
1200	40	.47
2400	24	.25
3600	20	.21
4800	5	.05
6000	1	.01
7200	1	.01
8400	0	0.
9600	0	0.
10800	0	0.

TABLE	AVERAGE	STD. DEV	NO. ENTRIES
T3	5759.51	3115.71	65

UPPER LIMIT	FREQUENCY	PERCENT OF TOTAL
1200	6	.12
2400	9	.18
3600	11	.17
4800	9	.14
6000	8	.12
7200	6	.09
8400	10	.15
9600	2	.03
10800	2	.03

TABLE	AVERAGE	STD. DEV	NO. ENTRIES
TJ025	44.60	8.48	5

UPPER LIMIT	FREQUENCY	PERCENT OF TOTAL
10	0	0.
20	0	0.
30	1	.20
40	3	.60
50	1	.20

Figure 3-8b. Simulation Run Results for Production Shop (Cont'd)

3.3 A Bus Stop Simulation

3.3.1 Statement Of The Problem - A bus is scheduled to arrive at a bus stop every 30 minutes, but it may be as 1.5 ± 1.5 minutes late. Whether a bus is late or not in no way depends on whether the preceeding bus was late, and has no influence on whether the next bus will be late.

People arrive at the bus stop in a Poisson stream at a rate of 12 people every 30 minutes. The bus, with a capacity of 50, carries 35 ± 15 passengers when it arrives. After some 3 to 7 of the passengers get off (uniformly distributed), as many waiting people as possible board the bus. Those unable to board the bus is leave the bus stop and do not return.

It takes 4 ± 3 seconds to unload a passenger and 8 ± 4 seconds to load a passenger. Passengers unload one-by-one, and get onboard one-by-one. Waiting people do not begin to board until everyone intending to get off has done so. The sequence for getting onboard is on a first-come, first-served basis. Any person who arrives at the bus stop while a bus is still loading is able to get on, providing there is room for them. In case of a time tie between the events "bus is now

finished loading" and "next passenger arrives", the arriving passenger is taken onboard (providing, of course, that there is room for that passenger).

We are to build a model which simulates the events at the bus stop. Design the model to gather the following information:

1. Obtain a waiting line statistics for people waiting at the bus stop, include an estimate of the distribution of in-line residence time.
2. Estimate the distribution of the random variable "number of passengers per arriving bus who are unable to receive service".

3.3.2 Discussion Of The Model - We have selected to build this model in BGSL so that it is exactly the same as the GPSS model. This is being done in order to verify upward compatibility of BGSL and GPSS and prove the correctness of translations for many advanced blocks. As Schriber describes in his book the model consists of two model segments. The first segment simulates the passengers who arrive at the bus stop, wait for the bus, and then either get on or leave if there are no further seats available. The second

segment simulates the bus and the passengers on board who want to get off the bus. A passenger can get on the bus when a bus arrives at the bus stop. They must however wait for all the intended passengers to get off prior to getting on the bus. A gate block is used as a logic switch which controls the entrance of passengers to the bus. The bus-gate can be controlled from the bus segment or the passenger segment. The bus transaction opens the gate when a bus arrives and intended passengers get off the bus. In the passenger segment, each passenger while getting on the bus will close the gate so the other passengers have to wait for their turn. The passenger transaction will open the gate when it leaves the model, thus allowing the next passenger in line to try to get on the bus.

The bus waits until all the passengers get on the bus before the bus exits the stop and closes the gate. Figure 3-9 presents the BGSL model and Figure 3-10 shows the BCSL translation for the same model. In this model, even though we could use SIMSCRIPT provided distribution functions, we used the FUNCTION command to stay compatible with the GPSS model and to demonstrate that this block is translated correctly in case a user wants to define his own distribution function.

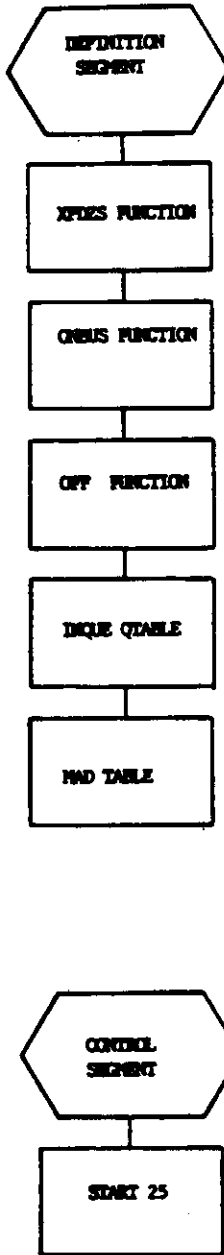


Figure 3-9a. BGS L Model for the Bus Stop

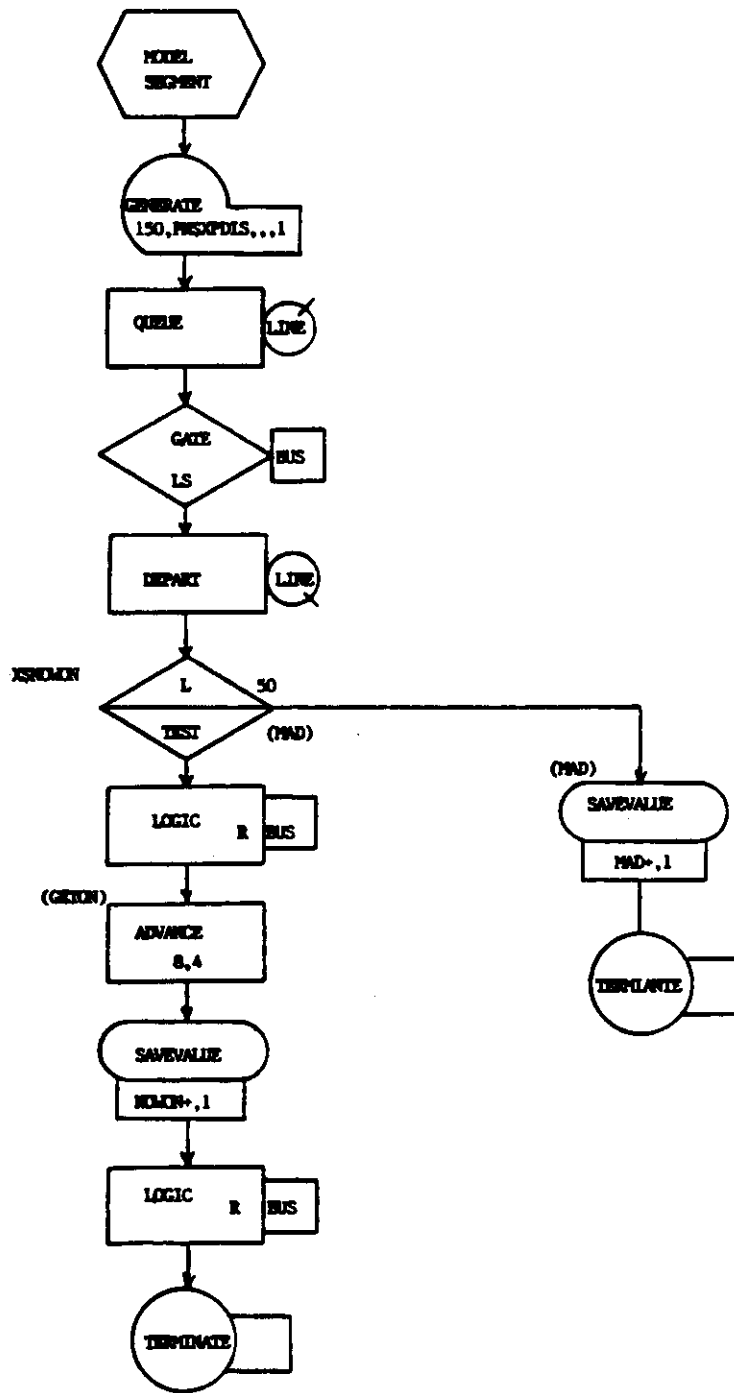


Figure 3-9b. BGS Model for the Bus Stop (Cont'd)

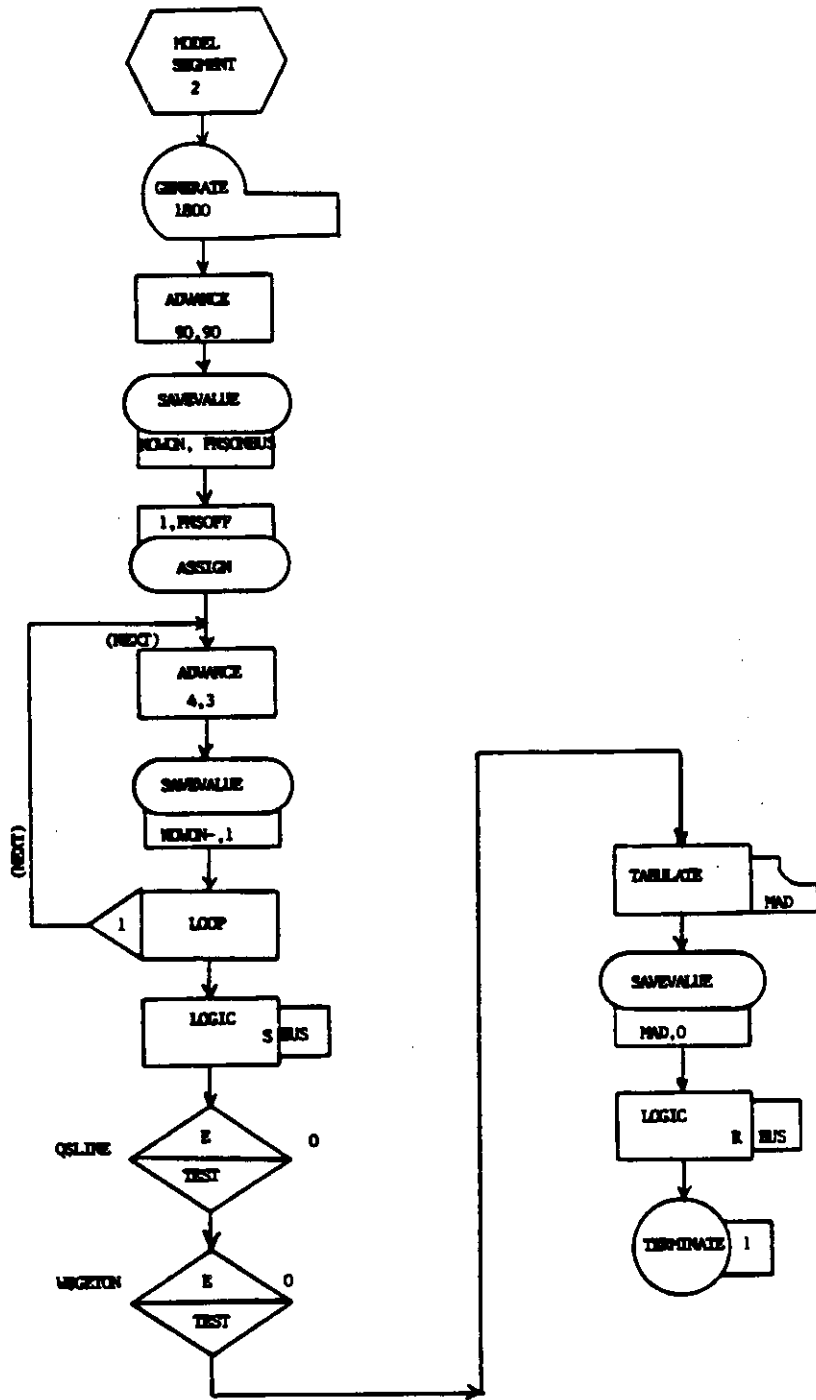


Figure 3-9c. BGSL Model for the Bus Stop (Cont'd)

```

*****
**
** BCSL CODE GENERATOR VERSION 1.0
**
*****
*
*      SIMULATE
*
*
*** DEFINITION SEGMENT
*
*
* XPDIS FUNCTION  LN1,C24                                B0002
0,3/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69
B0002
.6,.915/.7,1.2/.75,1.48/.8,1.6/.88,1.83/.88,
B0002
2.12/.3,2.3/.9,.52/.94,2.81/.95,2.99/.96,
B0002
3.2/.97,3.5/.98,3.5/.99,4.6/.995,5.3/.996,
B0002
6.2/.999,7./9996,8./1.,10000.
B0003
* XBUS FUNCTION  LN1,C2
B0003
0.,20./1.,51.
B0004
* OFF FUNCTION  LN1,C2
B0004
0.,3./1.,8.
B0005
* INPUT TABLE  LINE,300,300,7
B0006
* NAME TABLE  X$BAD,0,1,10
B0006
*
*
*** NAME SEGMENT
*
*
* GENERATE 150,FMSXPDIS,,1                                B0008
* QUEUE   LINE                                           B0009
* GATE LS  BUS                                           B0010
* DEPART  LINE                                           B0011
* TEST L  X$MONON,50,BAD                                B0012
* LOGIC K  BUS                                           B0013
* GETON ADVANCE 8,4                                       B0014
* SAVEVALUE MONON+,1                                     B0015
* LOGIC S  BUS                                           B0016
* TERMINATE B0017
* NAME SAVEVALUE BAD+,1                                  B0018
* TERMINATE B0019
*
*
*** NAME SEGMENT
*
*
* GENERATE 1000                                          B0021
* ADVANCE 90,90                                          B0022
* SAVEVALUE MONON,F$ONBUS                               B0023
* ASSIGN 1,F$OFF                                         B0024
* NEXT ADVANCE 4,3                                       B0025
* SAVEVALUE MONON-,1                                    B0026
* LOOP 1,NEXT                                           B0027
* LOGIC S  BUS                                           B0028

```

Figure 3-10a. BCSL Model for the Bus Stop

FILE: CASE5E BCSL A1

VE/SP CONVERSATIONAL MONITOR SYSTEM

```

      TEST E   QSLINE,0           B0029
      TEST E   ASSEION,C         B0030
      TABULATE MAD                B0031
      SAVEVALUE WEL,J           B0032
      LOGIC A   BCS              B0033
      TERMINATE 1                B0034
*
*
*** CONTROL SEGMENT
*
*      STAGE 25                  B0036
*
*      END

```

Figure 3-10b. BCSL Model for the Bus Stop (Cont'd)

3.3.3 Discussion Of SIMSCRIPT Equivalent Of The Bus Model - In this model in order to stay compatible with the GPSS model we did not define the transaction names. As a result, the BCSL compiler has automatically assigned names to each transaction (the passenger transaction is called TRAN.11 and the bus transaction is called TRAN.12).

Figure 3-11 contains the SIMSCRIPT equivalent for the bus stop model. The preamble section defines the processes, resources, variables equivalences and standard numerical attributes. Tally, accumulate statements and histogram statements are included to accumulate needed statistics. The distribution function variables like XPD2IS ONBUS and OFF are defined as Random Linear variables.

The gate controller switch (BUS variable), the number of people in the line (Q\$LINE) and the number of passengers getting on the bus (W\$GETON) are defined as left monitored variables.

The initialization routine reads the values for user defined distribution functions, initializes the monitored variables to zero, initializes the counters and stop flags to zero and finally activates the generator processes. The output routines generate

reports for resources, queues, transactions, tables and statistical variables.

The Monitor routine which monitors the number of people in the line (called LEFT ROUTINE Q\$LINE) checks if the line is empty. It then reactivates the bus transaction that is blocked waiting for the line to be empty. The monitor routine for the number of people trying to get on the bus checks if the last passenger has gotten on the bus. It then reactivates the BUS transaction that has been blocked waiting for the last person to get on the bus. These two monitored routines in conjunction allow the bus to leave the station if the line is empty and the last person has finished getting on the bus. The monitor routine for the bus gate controller checks if the bus is at the station and no one is getting on the bus then it reactivates the first passenger transaction which is waiting to get on the bus.

The passenger transaction uses QUEUE and DEPART blocks to gather statistics on the GATE block and its corresponding queue. Notice that for RESOURCES we do not need to use the QUEUE and DEPART blocks. This is because the queue statistic is calculated automatically for them. The GATE block will cause the passenger

transaction to be blocked if the bus is not at the station. The TEST block checks if the bus capacity is full. If so, it directs the remainder of the waiting passengers to leave the station. This is done by transferring control to the label "MAD", where the number of passengers which could not get on the bus is incremented and the passenger transaction leaves the simulation model without getting on the bus. Otherwise, if a passenger gets on the bus it will close the bus gate so that the remainder of the passengers wait their turn in line. Notice that every time a monitored variable like W\$GETON is updated, the left-monitor routine is activated and a check will be made to find out if the original condition is satisfied and all transactions which are blocked waiting for the variable to be met the condition is reactivated.

The bus transaction, after arrival, decides on the number of passengers on board and the number of passengers trying to get off the bus. Using the user defined distribution functions, bus transactions waits for passengers who want to get off the bus and opens the bus gate for them. This unblocks the passenger transactions which have been blocked waiting for the gate to open. Then the bus transaction is suspended (blocked) until the line is empty and the last passenger

has gotten on the bus, thus allowing the bus to leave the station. Upon leaving the station, the bus transaction tabulates the number of passengers who could not get on the bus (MAD people), resets the MAD counter and finally closes the bus gate before leaving the simulation model.

3.3.4 Discussion Of Results - The report contains the queue and transaction statistics in addition to distribution of the number of people which left the bus stop without getting on the bus and distribution of the number of people in the line. Figure 3-12 shows the output report. There were 291 arrivals to the line in total. The expected value would be about 300 and Schribers GPSS model result is 288. The maximum number of people in the line is 21 compared to the GPSS simulation result of 16 and average number of people in the line is 5.5 compared to the GPSS average of 5.6. There were no passengers that did not have to wait to get on the bus; therefore, the number of zero entries is zero. \$AVERAGE (average time/tran excluding zero entries) is equal to the average time/tran.

The in-line residence time distribution is demonstrated by the INQUE table. The average wait is 855 seconds compared with a GPSS waiting time of 889 seconds. There were about 51 passengers who waited between 300 to 600 seconds.

The distribution of people not served per stopping bus is included in the MAD table. Everyone was able to get aboard the bus 17 times out of 25, compared with a GPSS results of 21 times out of 25. All these statistics are consistent with the data provided in the problem statement and results achieved by the GPSS model for the same problem.

```

*.....*
*.....*
*.....* SIMSCRIPT CODE GENERATED BY .....*
*.....* BCSL TO SIMSCRIPT CROSS COMPILER .....*
*.....* ..... VERSION 1.2 .....*
*.....*
*.....*
1  PROGRAM
2  MODBALL NODE IS REAL
3  PROCESSES
4  EVERY TRM.11 HAS A MONITOR.V
5  AND MAY BELONG TO THE BLOCKED.11
6  EVERY TRM.12 HAS A MONITOR.V
7  AND MAY BELONG TO THE BLOCKED.12
8  THE SYSTEM OWNS THE BLOCKED.11
9  THE SYSTEM OWNS THE BLOCKED.12
10 EVERY GEN.11 HAS A STOP.FLAG.11
11 EVERY GEN.12 HAS A STOP.FLAG.12
12 **DEFINE VARIABLES AND LEFT MONITORED VARIABLES
13 DEFINE NSCELOS AS INTEGER VARIABLE
14 DEFINE NSGETON AS INTEGER VARIABLE MONITORED ON THE LEFT
15 DEFINE NSHAD AS INTEGER VARIABLE
16 DEFINE NSHAD AS INTEGER VARIABLE
17 DEFINE NSNEXT AS INTEGER VARIABLE
18 DEFINE NSNEXT AS INTEGER VARIABLE
19 DEFINE STOP.FLAG AS INTEGER VARIABLE
20 DEFINE BUS AS INTEGER VARIABLE MONITORED ON THE LEFT
21 DEFINE ISHOWN AS INTEGER VARIABLE
22 DEFINE ISHAD AS INTEGER VARIABLE
23 **DEFINE SNA POM QUEUES
24 DEFINE WSLINE AS INTEGER VARIABLE MONITORED ON THE LEFT
25 DEFINE WUSLINE AS INTEGER VARIABLE
26 DEFINE WCSLINE AS INTEGER VARIABLE
27 DEFINE WJSLINE AS INTEGER VARIABLE
28 DEFINE W.T.LINE AS INTEGER VARIABLE
29 DEFINE W.X.LINE AS INTEGER VARIABLE
30 **DEFINE TEST BLOCK FLAGS
31 DEFINE BLOCKED.FLAG.11 AS AN INTEGER VARIABLE
32 DEFINE BLOCKED.FLAG.12 AS AN INTEGER VARIABLE
33 **DEFINE STATISTICAL VARIABLES
34 **DEFINE RANDOM VARIABLES
35 THE SYSTEM HAS A XPLS RANDOM LINEAR VARIABLE
36 DEFINE XPLS AS A REAL VARIABLE
37 THE SYSTEM HAS A QBUS RANDOM LINEAR VARIABLE
38 DEFINE QBUS AS A REAL VARIABLE
39 THE SYSTEM HAS A OFF RANDOM LINEAR VARIABLE
40 DEFINE OFF AS A REAL VARIABLE
41 **DEFINE MATRICES
42 **DEFINE GPSS GLOBAL VARIABLES
43 DEFINE P1 TO BEAN P_ARRAY(1)
44 DEFINE P2 TO BEAN P_ARRAY(2)
45 DEFINE P3 TO BEAN P_ARRAY(3)
46 DEFINE P4 TO BEAN P_ARRAY(4)
47 DEFINE P5 TO BEAN P_ARRAY(5)
48 DEFINE P6 TO BEAN P_ARRAY(6)

```

Figure 3-11a. SIMSCRIPT Model for the Bus Stop

```

49 DEFINE P7 TO BEAN P_ARRAY(7)
50 DEFINE P8 TO BEAN P_ARRAY(8)
51 DEFINE P9 TO BEAN P_ARRAY(9)
52 DEFINE P10 TO BEAN P_ARRAY(10)
53 DEFINE X1 TO BEAN X_ARRAY(1)
54 DEFINE X2 TO BEAN X_ARRAY(2)
55 DEFINE X3 TO BEAN X_ARRAY(3)
56 DEFINE X4 TO BEAN X_ARRAY(4)
57 DEFINE X5 TO BEAN X_ARRAY(5)
58 DEFINE X6 TO BEAN X_ARRAY(6)
59 DEFINE X7 TO BEAN X_ARRAY(7)
60 DEFINE X8 TO BEAN X_ARRAY(8)
61 DEFINE X9 TO BEAN X_ARRAY(9)
62 DEFINE X10 TO BEAN X_ARRAY(10)
63 DEFINE UNITS TO BEAN UNITS
64 DEFINE MONTION.V AS TEXT VARIABLE
65 DEFINE M1 AS A REAL VARIABLE
66 DEFINE C1 TO BEAN TIME.V*HOURS.V*MINUTES.V
67 **TRANSACTION DEFINITION
68 DEFINE TRAN.CNT.11 AS AN INTEGER VARIABLE
69 DEFINE C.TRAN.CFT.11 AS AN INTEGER VARIABLE
70 DEFINE SEED.TRAN.11 AS AN INTEGER VARIABLE
71 DEFINE TRAN.CNT.12 AS AN INTEGER VARIABLE
72 DEFINE C.TRAN.CFT.12 AS AN INTEGER VARIABLE
73 DEFINE SEED.TRAN.12 AS AN INTEGER VARIABLE
74 **DEFINE TABLE VARIABLES
75 DEFINE MAD AS REAL VARIABLE
76 **TALLY AND ACCUMULATE FOR RESOURCES
77 **TALLY AND ACCUMULATE FOR STATISTICAL VARIABLES
78 **TALLY AND ACCUMULATE FOR QUEUES
79 ACCUMULATE QSLINE AS AVERAGE AND QNSLINE AS MAXIMUM
80 CF QSLINE
81 TALLY QSLINE AS MEAN OF Q.T.LINE
82 TALLY QNSLINE AS MEAN OF Q.I.LINE
83 DEFINE GEN.TIME.11 AS A REAL VARIABLE
84 TALLY AVG.GEN.TIME.11 AS MEAN OF GEN.TIME.11
85 DEFINE GEN.TIME.12 AS A REAL VARIABLE
86 TALLY AVG.GEN.TIME.12 AS MEAN OF GEN.TIME.12
87 **HISTOGRAM FOR TABLES
88 TALLY MAD.HISTO(0 TO          8 BY 1)
89 AS THE HISTOGRAM AND MAD.AVG AS AVERAGE AND
90 MAD.STD AS STD.DEV AND MAD.NUM.ENT AS NUMBER OF MAD
91 TALLY IN_UP(500 TO          1800 BY 500)
92 AS THE HISTOGRAM AND AND
93 Q.T.LINE.STD AS STD.DEV OF Q.T.LINE
94 END

```

Figure 3-11a. SIMSCRIPT Model for the Bus Stop
(Cont'd)

CACI SIMSCRIPT II.5 IBM S/370 49.3 PAGE 1

OPTIONS TERM,LOAD,10,TRACE2,NOTERM,CNR,REN=NE 17-NOV-1985 17:40 (1

```

1 ROUTINE INITIALIZE
2 NORMALLY MODE IS INITGZA
3 READ IPDIS
4 READ OMBUS
5 READ OFF
6 LET BUS = 0
7 LET ISHCON = 0
8 LET ISHAD = 0
9 LET TRAB.CN12.11 = 0
10 LET TRAB.CN12.12 = 0
11 LET BLOCKED.FLAG.11 = 0
12 LET BLOCKED.FLAG.12 = 0
13 LET STOP.FLAG = 25
14 ACTIVATE A GEN.11 NOW
15 ACTIVATE A GEN.12 NOW
16 END

```

CACI SIMSCRIPT II.5 IBM S/370 49.3 PAGE 1

OPTIONS TERM,LOAD,10,TRACE2,NOTERM,CNR,REN=ELU 17-NOV-1985 17:40 (1

```

1 LET POUTLINE BUS
2 DEFINE N.BUS AS AN INTEGER VARIABLE
3 ENTER WITH N.BUS
4 IF N.BUS EQ 1
5 FOR EACH TRAB.11 IN BLOCKED.11 WITH
6 MONITOR.V(TRAN.11) EQ "BUS"
7 DO
8 REMOVE THE FIRST TRAB.11 FROM BLOCKED.11
9 REACTIVATE THIS TRAB.11 NOW
10 LOOP
11 ALWAYS
12 MOVE FROM N.BUS
13 RETURN
14 END

```

Figure 3-11b. SIMSCRIPT Model for the Bus Stop
(Cont'd)

```

OPTIONS  TRSN,LOAD,LD,TRACE2,NOTERS,CHK,REN=REN
1  LEFT ROUTINE QSLINE
2  DEFINE N.QSLINE AS AN INTEGER VARIABLE
3  ENTER WITH N.QSLINE
4  IF N.QSLINE EQ 0
5  FOR EACH TRSN.12 IN BLOCKED.12 WITH
6  MONITOR.V(TRAN.12) EQ "QSLINE"
7  DO
8  REMOVE THE FIRST TRSN.12 FROM BLOCKED.12
9  REACTIVATE THIS TRSN.12 NOW
10 LOOP
11 ALWAYS
12 MOVE FROM N.QSLINE
13 RETURN
14 END

```

```

OPTIONS  TRSN,LOAD,LD,TRACE2,NOTERS,CHK,REN=REN
1  LEFT ROUTINE NSGETON
2  DEFINE N.NSGETON AS AN INTEGER VARIABLE
3  ENTER WITH N.NSGETON
4  IF N.NSGETON EQ 0
5  FOR EACH TRSN.12 IN BLOCKED.12 WITH
6  MONITOR.V(TRAN.12) EQ "NSGETON"
7  DO
8  REMOVE THE FIRST TRSN.12 FROM BLOCKED.12
9  REACTIVATE THIS TRSN.12 NOW
10 LOOP
11 ALWAYS
12 MOVE FROM N.NSGETON
13 RETURN
14 END

```

Figure 3-11c. SIMSCRIPT Model for the Bus Stop
 (Cont'd)

```

1  PROCESS TRAL.11
2  DEFINE P_ARRAY AS INTEGER,1-DIMENSIONAL ARRAY
3  RESERVE P_ARRAY AS 10
4  DEFINE TIME.11 AS A REAL VARIABLE
5  TIME.11 = TIME.V
6  **      QUEUE      LINE      B0009
7
8  LET TIME.LINE = TIME.V
9  LET QSLINE = QSLINE + 1
10 LET QSLINE = QSLINE + 1
11 LET QSLINE = QSLINE + 1
12 **      GATE LS      BUS      B0010
13
14 'L_11'
15 IF BUS EQ 1
16 ELSE
17 LET MONITOR.V(INAL.11) = "BUS"
18 FILE TRAL.11 IN PLOCKED.11
19 SUSPEND
20 GO TO L_11
21 ALWAYS
22 **      DEFAL      LINE      B0011
23
24 LET Q.I.LINE = (TIME.V - TIME.LINE)*HOURS.V*MINUTES.V
25 IF Q.I.LINE LE 0
26 LET QSLINE = QSLINE + 1
27 ELSE
28 LET Q.I.LINE = (TIME.V - TIME.LINE)*HOURS.V*MINUTES.V
29 ALWAYS
30 LET QSLINE = QSLINE - 1
31 LET QSLINE = QSLINE - 1
32 **      TEST L      XSNOWN,50,HAD      B0012
33
34 IF XSNOWN LI 50
35 ELSE
36 GO TO HAD
37 ALWAYS
38 **      LOGIC L      BUS      B0013
39
40 LET BUS = 0
41 **      GATE ADVANCE      0,4      GATE
42
43 'GATE'
44 LET NSGATE = NSGATE + 1
45 LET NSGATE = NSGATE + 1
46 HOUR UNIFORM.F((8-4)/1.0,(8+4)/1.0.
47 SELD.TAN.11) UNITS
48 **      SAVEVALUE NOWON+,1      B0015
49
50 LET NSGATE = NSGATE - 1
51 LET XSNOWN = XSNOWN + 1
52 **      LOGIC S      BUS      B0016
53
54 LET BUS = 1
55 **      TESTIDATF      B0017
56

```

Figure 3-11d. SIMSCRIPT Model for the Bus Stop (Cont'd)


```

57 LET M1 = (TIME.V - TIME.B1)*HOURS.V*MINUTES.V
58 LET C.TRAN.CNT.11 = C.TRAN.CNT.11 - 1
59 RETURN
60 ** M1D SAVEVALUE M1D*,1 M1D
61
62 **M1D *
63 LET M1DAD = M1DAD + 1
64 LET M1HAD = M1HAD + 1
65 LET M1HAD = M1HAD + 1 M1HAD
66 ** TERMINATE M1HAD
67
68 LET M1HAD = M1HAD - 1
69 LET M1 = (TIME.V - TIME.B1)*HOURS.V*MINUTES.V
70 LET C.TRAN.CNT.11 = C.TRAN.CNT.11 - 1
71 RETURN
72 ***
73
74 ***
75
76 ***** M1D SEGMENT
77
78 ***
79
80 ***
81
82 ** GENERATE 1800 M1HAD
83
84 END

```

Figure 3-11e. SIMSCRIPT Model for the Bus Stop (Cont'd)

```

1  PROCESS TRAK.12
2  DEFINE P_ARRAY AS INTEGER,1-DIMENSIONAL ARRAY
3  RESERVE P_ARRAY AS 10
4  DEFINE TIME.1 AS A REAL VARIABLE
5  TIME.1 = TIME.V
6  **      ADVANCE      90,90                                B0022
7
8  WORK UNIFORM.F((90-90)/1.0,(90+90)/1.0,
9  SPEED,TRAK.12) UNITS
10 **      SAVEVALUE  WORKCN,PNOWBUS                                B0023
11
12  LET PNOWBUS = CNBUS
13  LET XNOWCN = PNOWBUS
14  **      ASSIGN      1,PNOWOFF                                B0024
15
16  LET PNOWOFF = OFF
17  LET P_ARRAY(1) = PNOWOFF * 1
18  **      NEXT ADVANCE  4,3                                NEXT
19
20  *NEAR *
21  LET NSNEXT = NSNEXT + 1
22  LET WSNEXT = WSNEXT + 1
23  WORK UNIFORM.F((4-1)/1.0,(4+3)/1.0,
24  SLEP,TRAK.12) UNITS
25  **      SAVEVALUE  NWORKN-,1                                B0026
26
27  LET WSNEXT = WSNEXT - 1
28  LET XSNOWCN = XSNOWCN - 1
29  **      LOG2      1,NEXT                                B0027
30
31  LET P_ARRAY(1) = P_ARRAY(1) - (1 * 1)
32  IF P_ARRAY(1) LE 0
33  ELSE
34  GO TO NEXT
35  ALWAYS
36  **      LOGIC 5      BUS                                B0028
37
38  LET BUS = 1
39  **      TEST 2      QSLINE,0                                B0029
40
41  IF QSLINE EQ 0
42  ELSE
43  LET MONITOR.V(12) = "QSLINE"
44  FILE TRAK.12 IN BLOCKED.12
45  SUSPEND
46  ALWAYS
47  **      TEST 2      WGETON,0                                B0030
48
49  IF WGETON EQ 0
50  ELSE
51  LET MONITOR.V(12) = "WGETON"
52  FILE TRAK.12 IN BLOCKED.12
53  SUSPEND
54  ALWAYS
55  **      TABULATE  3AD                                B0031
56

```

Figure 3-11f. SIMSCRIPT Model for the Bus Stop
 (Cont'd)

```

57 LET MAD = ISHAD
58 '' SAVEVALUE MAD,0 B0032
59
60 LET ISHAD = 0
61 '' LOGIC P BUS B0033
62
63 LET EUS = 0
64 '' TERMINATE 1 B0034
65
66 LET N1 = (TIME.V - TIM.N1)*HOURS.V*MINUTES.V
67 LET C.TAN.CN1.12 = C.TAN.CN1.12 - 1
68 IF STOP.FLAG = STOP.FLAG - 1
69 IF STOP.FLAG LE 0
70 CALL OUTPUT_ROUTINE
71 STOP
72 ALWAYS
73 RETURN
74 ***
75
76 ***
77
78 ***** CONTROL SEGMENT
79
80 ***
81
82 ***
83
84 '' START 25 B0036
85
86 ***
87
88 ***
89
90 '' END
91
92 END
  
```

Figure 3-11g. SIMSCRIPT Model for the Bus Stop
(Cont'd)

 *
 * GRAPHICAL SIMULATION SYSTEM OUTPUT *
 *

QUEUE NAME	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL EMPLOYEES	ZERO ENTRIES	AVERAGE TIME/TRAN	SAVERAGE TIME/TRAN	CURRENT CONTENTS
LINE	21	5.510	291	0	855.649	855.649	0

TRANSACTION NAME	NUMBER CREATED	AVERAGE CREATION TIME
11	291	155.251
12	25	1800.000

TABLE NAME	AVERAGE	STD. DEV	NO. ENTRIES
H&E	1.88	3.60	25

UPPER LIMIT	FREQUENCY	PERCENT OF TOTAL
0	17	.68
1	1	.04
2	1	.04
3	2	.08
4	0	0.
5	0	0.
6	1	.04
7	0	0.
8	3	.12

QUEUE TABLE NAME	AVERAGE	STD. DEV	NO. ENTRIES
INQUE	855.65	508.47	291

UPPER LIMIT	FREQUENCY	PERCENT OF TOTAL
300	101	.35
600	51	.18
900	50	.17
1200	52	.18
1500	36	.12
1800	1	.00

Figure 3-12. Simulation Run Results for the Bus Stop Model

4 ADVANCED FEATURES OF THE GRAPHICAL SIMULATION SYSTEM

The Graphical Simulation system is able to simulate a Flexible Manufacturing System using special purpose blocks. These blocks are designed so that an unsophisticated user (non-programmer) can use the system to simulate a Flexible Manufacturing System using the floor plan of the target manufacturing system. In this chapter we discuss the approach taken in the simulation of Flexible Manufacturing systems using a manufacturing shop case. In addition a description of each new block used in the modeling of Flexible Manufacturing systems is followed by translation techniques used for each block.

Later, we discuss how the Graphical Simulation System can be expanded in order to create special purpose simulation systems and the rules used in creation of new blocks in the Block Command Symbolic Language and Block Graphic Symbolic Language are described. Other advanced features of GSS, like adding user written SIMSCRIPT code to the models and usage of MACRO blocks is also included.

4.1 Modeling Flexible Manufacturing Systems

A Flexible Manufacturing Systems consists of input/output (receiving/shipping) stations and workstations connected together via transporter lines. The raw material enters the shop through input parts and follows a scheduled visit to workstations on its route, and leaves the system at an output station. Very often there is only one station used for Input/Output throughout the shop. A simple Flexible Manufacturing System is selected to demonstrate how a Block Graphic Simulation Model can be developed using the drawing of the FMS floor plan. Figure 4-1 displays the drawing of the FMS floor plan and the job routes.

In order to model this shop, we simply need to unfold each job's routing so that we can draw the workstations in the order in which each job visits them. The given FMS example in Figure 4-1 has the following workstation routing.

JOB TYPE	STATIONS IN ROUTE
1	0, 1, 2, 4, 3, 0
2	0, 2, 3, 1, 4, 0

Thereby input and output stations are the same as station zero. The distances between stations are entered into a distance table for the manufacturing plan.

We can graphically draw the order in which the job visits each station given the fact that in order for the workpieces to move from station to station it needs a transporter. Figure 4-2 shows how an unfolded visitation sequence can be drawn graphically. Notice that entering and leaving Flexible Manufacturing Systems can be done through any workstation. In order to build the Block Graphic Symbolic language model for this FMS all we need to do is to add Generator Blocks which introduce jobs into the system and terminator block which allow jobs to exit from the system.

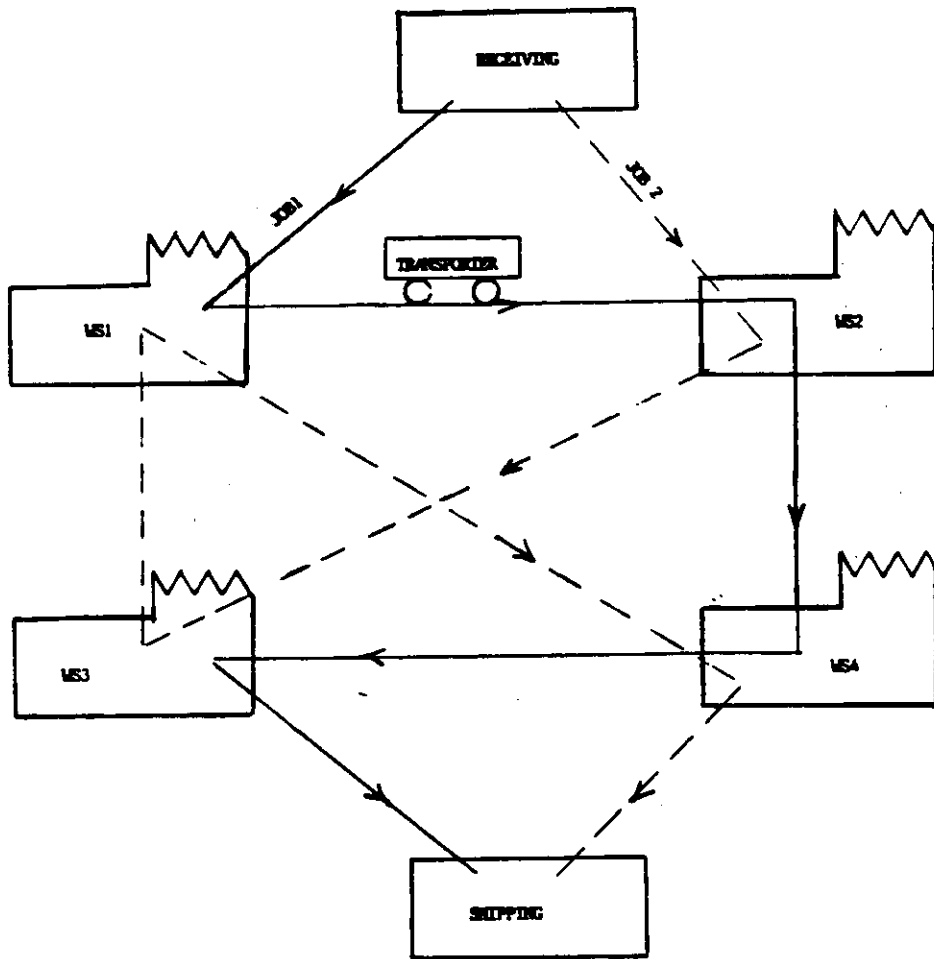


Figure 4-1. A Flexible Manufacturing System Floor Plan and job routes

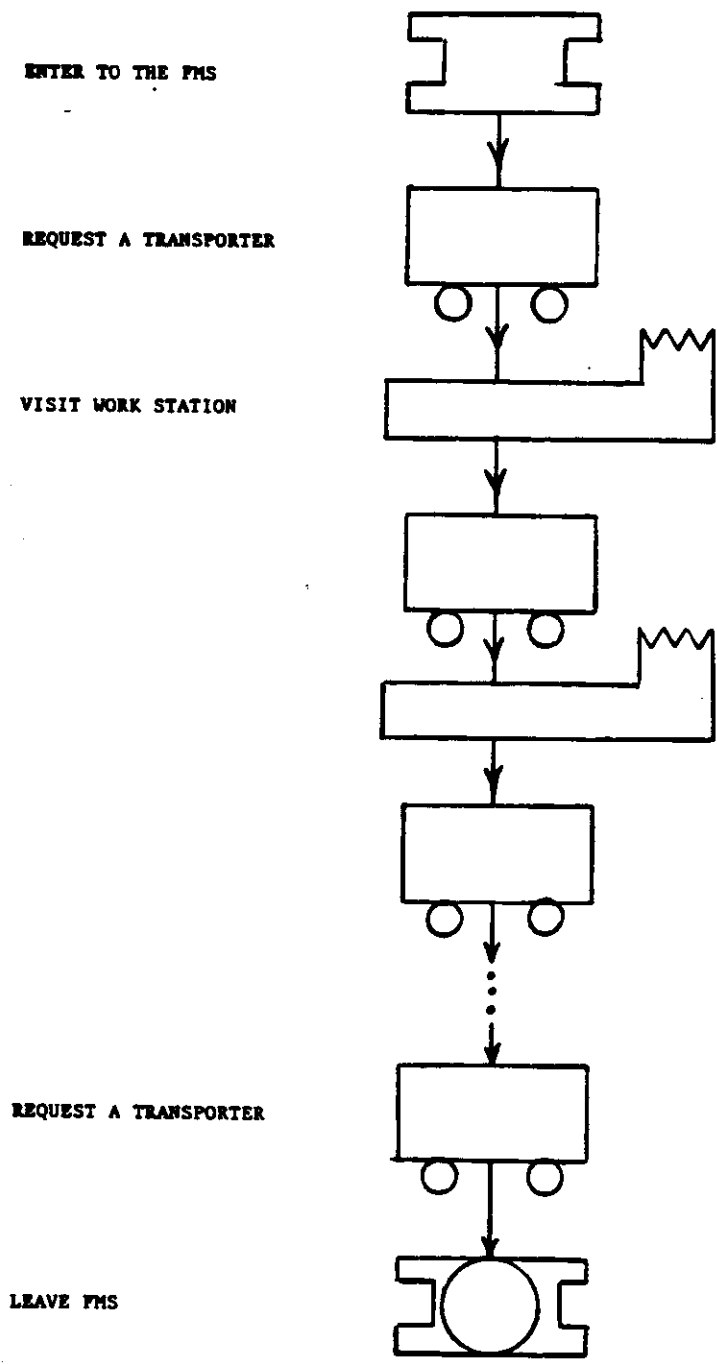


Figure 4-2. Unfolded Visitation Sequence

4.1.1 Statement Of The Problem - The Flexible Manufacturing example presented by Law and Larmey [25] has been selected to illustrate some of the concepts involved in modeling such systems. This example has been modified to represent a general case for the Flexible Manufacturing System. The manufacturing shop consists of an input/output station and five work stations. At present, workstations 1, 2, 3, 4, and 5 consist of 3,3, 4,4, and 1 identical machines, respectively. The distance (in feet) between the six stations are as follows (the input/output station is number 6).

STATION	1	2	3	4	5	6
1	0	90	100	180	200	270
2	90	0	100	200	180	270
3	100	100	0	100	100	180
4	180	200	100	0	90	100
5	200	180	100	90	0	100
6	270	270	180	100	100	0

Thus, for example, the distance between the input/output station and workstation 3 is 180 feet.

Assume that jobs (or workpieces) arrive at the input/output station with interarrival times that are independent exponential random variables with a mean of 0.25 hours. There are three types of jobs, and jobs are

of types 1, 2, and 3 with respective probabilities 0.3, 0.5, and 0.2. Job types 1, 2, 3 require 4, 3, 5 tasks to be done, respectively, and each task must be done at a specified workstation and in a prescribed order. Each job begins at the input/output station, travels to workstations, and then leaves the system at the input/output station. The routing for different job types are as follows:

JOB TYPE	WORKSTATIONS IN ROUTING
1	3, 1, 2, 5
2	4, 1, 3
3	2, 5, 1, 4, 3

Thus, type 2 jobs enter the system at station 6 (i.e., the receiving station). They have tasks done at workstations 4, 1, 3 and finally leave the system at station 6. A job must be moved from one station to another by a transporter (e.g., an automated guided vehicle), and there is only one transporter in the shop. The transporter moves at a speed of 5 feet per second. Thus, 36 seconds are required for the transporter to move from station 6 to station 3. The transporter processes requests by jobs in a FIFO manner. Furthermore, when the transporter finishes moving a job

to a workstation, it will remain at that station if there are no pending job requests.

If a job is brought to a particular workstation and all machines in that station are already busy or blocked, the job joins a single FIFO queue at that station. The time to perform a task at a particular machine is a 2-Erlang random variable whose mean depends on the job type and the workstation to which the machine belongs. (A 2-Erlang random variable with mean m is the sum of 2 independent exponential random variables each with mean $m/2$.) The mean service time for each job type and each task is as follows:

JOB TYPE	MEAN SERVICE TIME FOR SUCCESSIVE TASKS, HOURS
1	0.5, 0.60, 0.85, 0.50
2	1.10, 0.80, 0.75
3	1.20, 0.25, 0.70, 0.90, 1.00

Thus, a type 2 job requires a mean service time of 1.10 hours at work station 4 (the station where its first task will be done). When a machine finishes processing a job, the job waits until the job is removed by the transporter while the machine continues processing the next job in its queue.

Assuming no loss of continuity between successive days operation of the shop, we are to simulate the facility for 365 eight-hour days (or 2920 hours) and gather statistics on the following:

1. The mean total delay in the queue (exclusive of service times) and the mean total transporter delay, for each job type.
2. The Time-average number of jobs in the queue and the mean delay in the queue, for each workstation.
3. The average proportion of time that machines are working and are idle for each machine group.
4. The utilization of the transporter.

The transporter job delay at a particular workstation is the time interval between the instant the job requests the transporter and the instant the transporter arrives at the station. It does not include the known time for the transporter to move the job to the next station. The total transporter job delay is the sum of its transporter delays at all stations.

4.1.2 Discussion Of The Model - The model includes a matrix containing the distance table listing the

distances between stations. A workstation definition allocates a number to each workstation which corresponds to its row number and column number in the distance table. The transporter definition specifies the speed of the transporter and its corresponding distance table. Tables define the system life time of workpieces to be tabulated for statistical analysis.

Each job type is modeled using a separate model segment. The GENERATE block in each segment introduces the corresponding job type into the system using an exponential distribution function. A series of transporter request blocks and workstation request blocks connected to each other, represent all the events happening to each workpiece (job) in the modeled Flexible Manufacturing System. Finally, the tabulate block is used to gather statistics on the time spent in the model for each job type and a terminate block ends all the activities within a normal segment. A clock segment stops the simulation after 365 days. Figure 4-3 shows the Block Graphical Symbolic language, and Figure 4-4 shows the BCSL model for the FMS model.

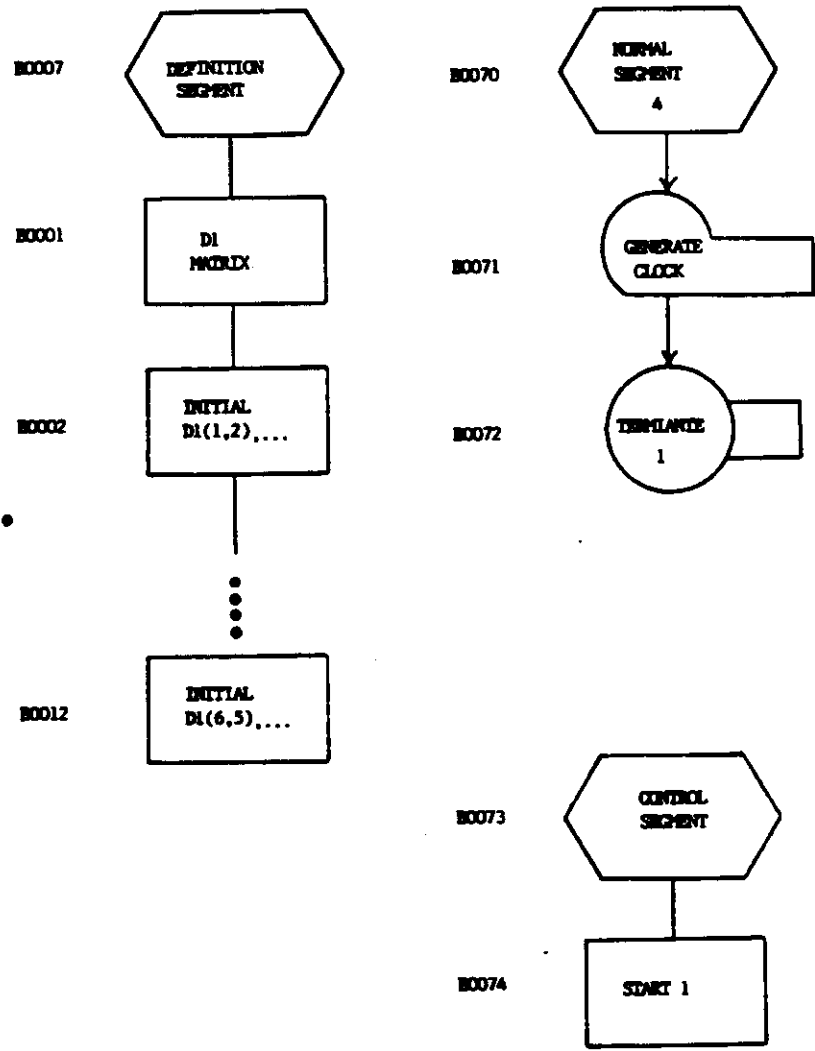


Figure 4-3a. BGS Model for the Flexible Manufacturing System

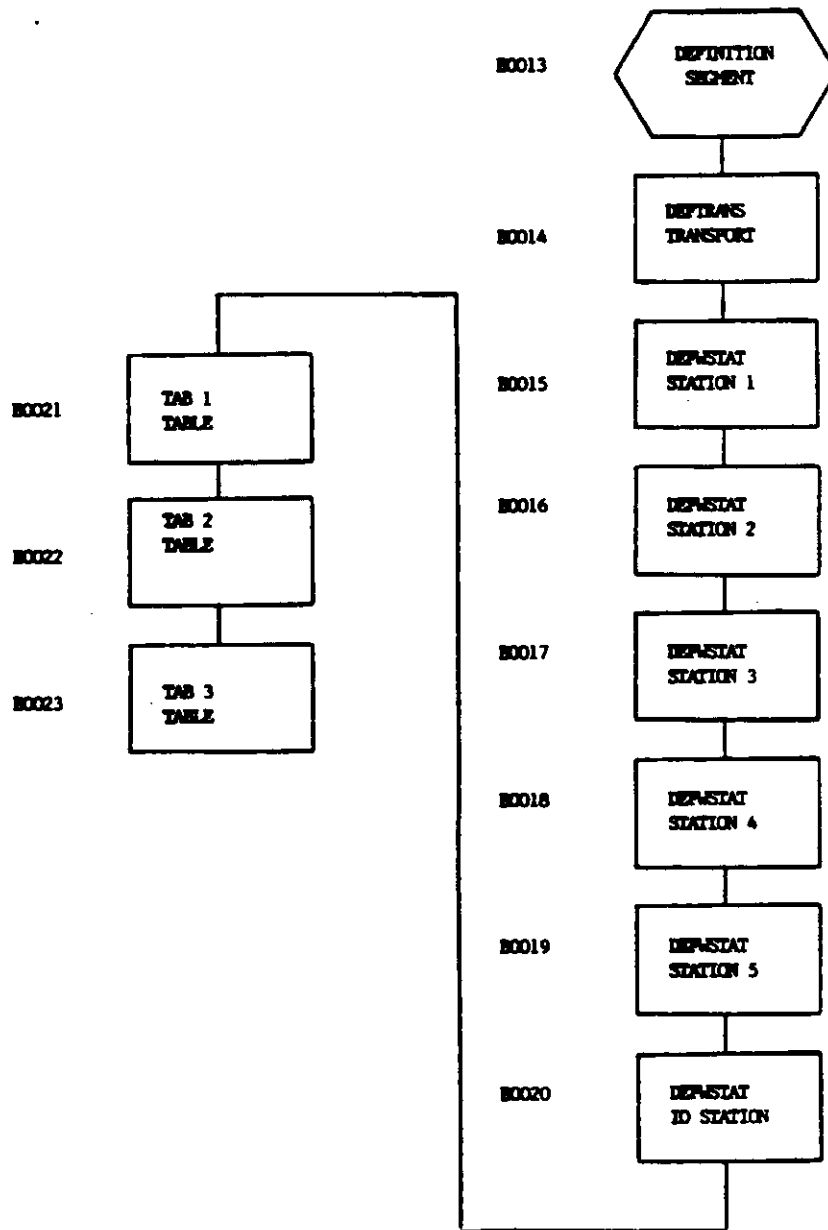


Figure 4-3b. BGS Model for the Flexible Manufacturing System (Cont'd)

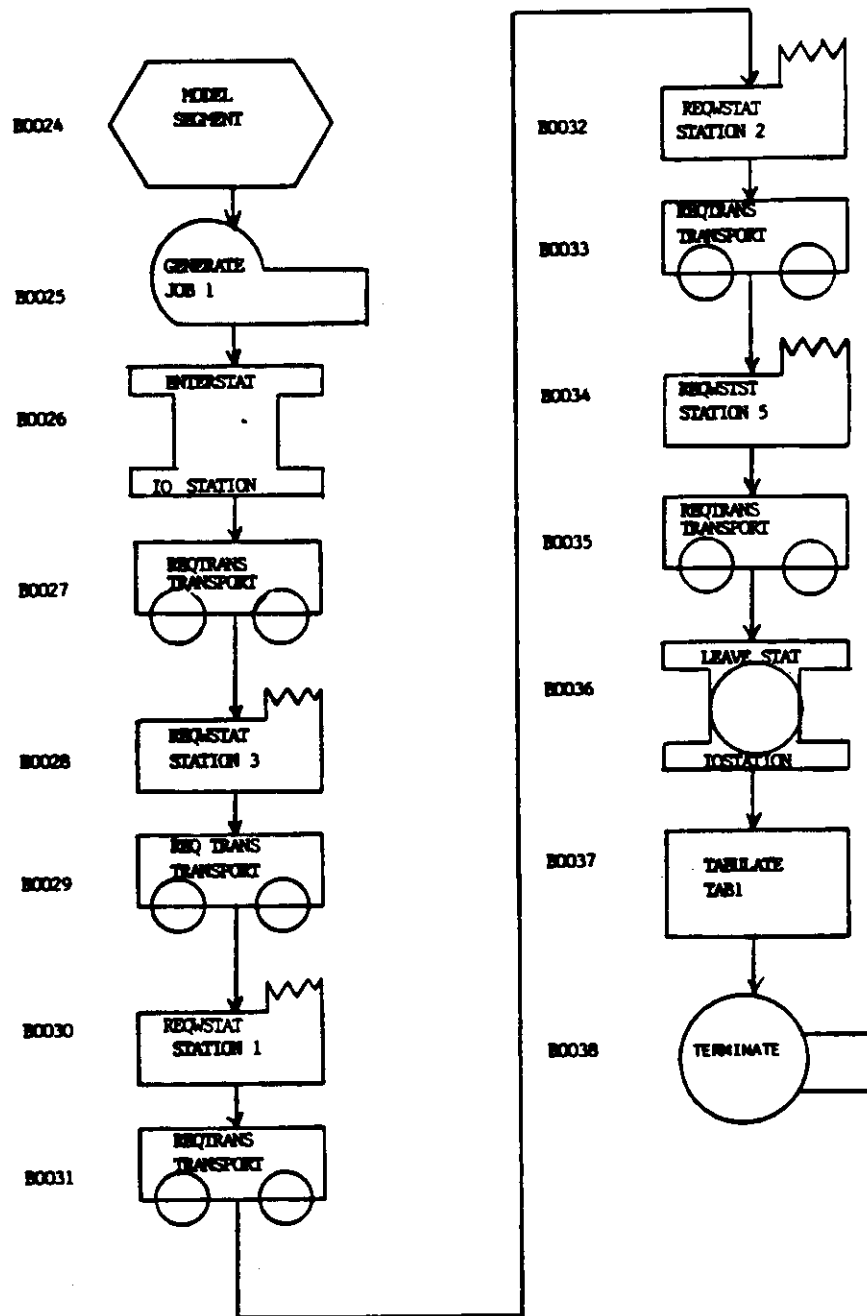


Figure 4-3c. BGS Model for the Flexible Manufacturing System (Cont'd)

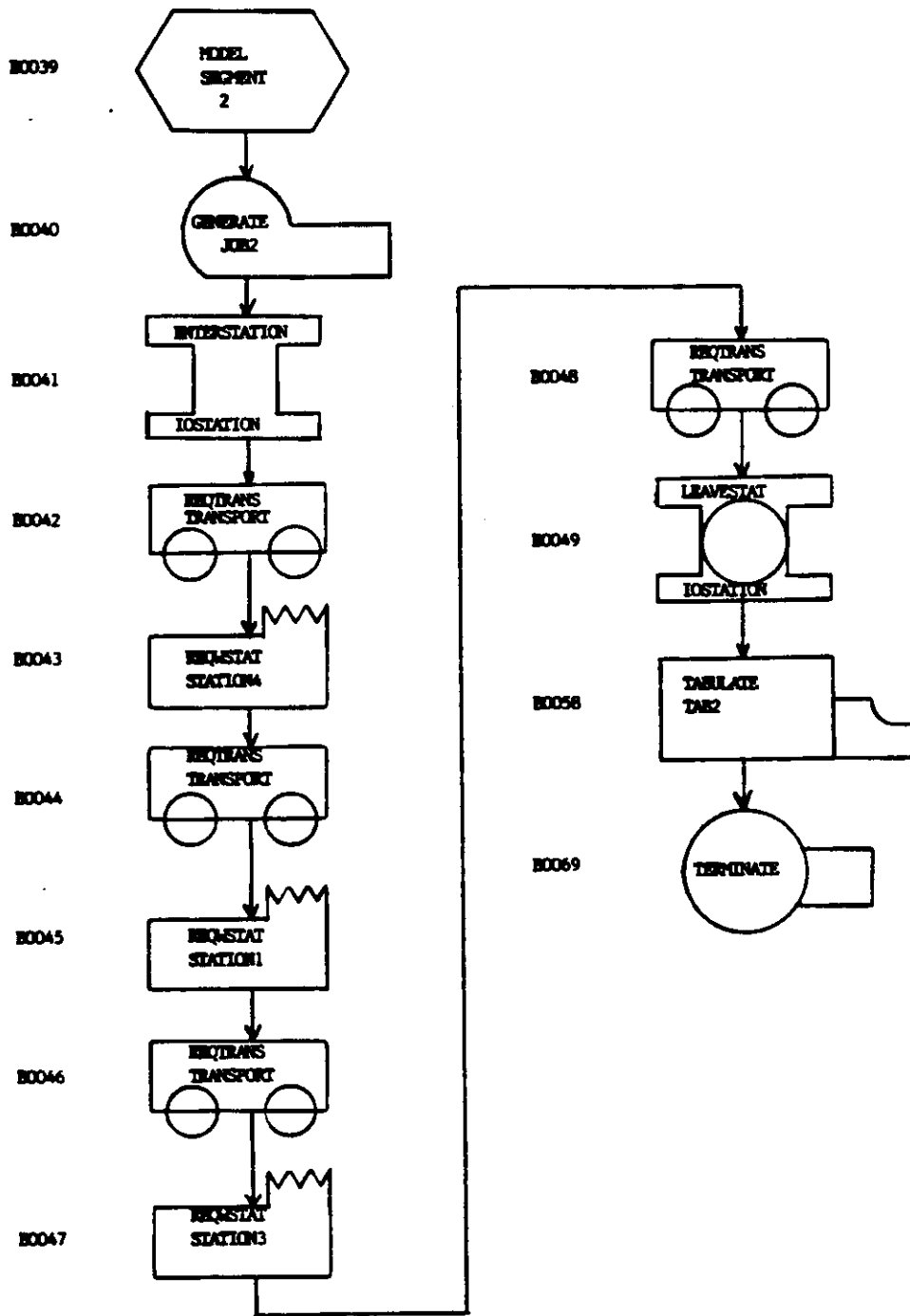


Figure 4-3d. BGS Model for the Flexible Manufacturing System (Cont'd)

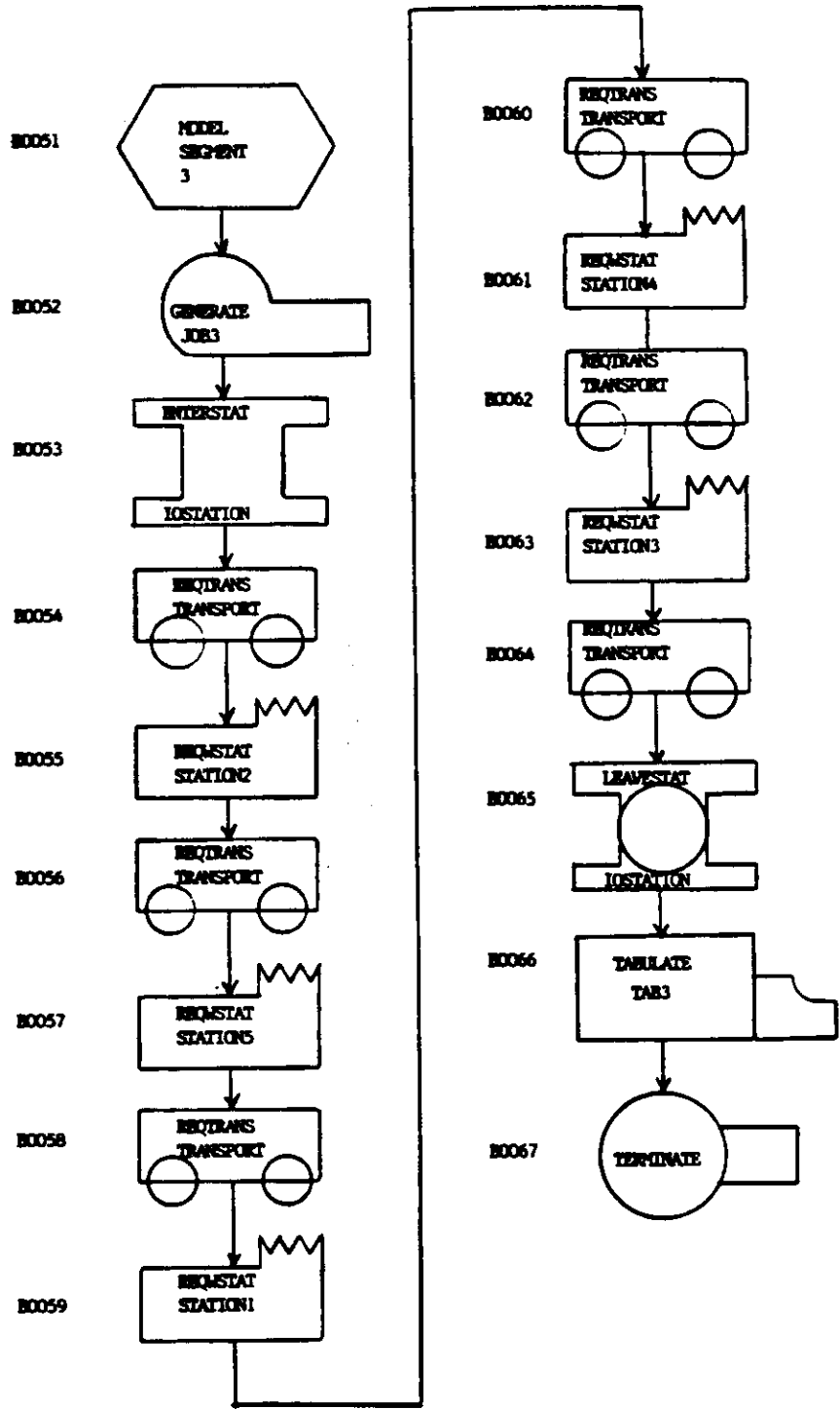


Figure 4-3e. BGS L Model for the Flexible Manufacturing System (Cont'd)

```

*****
**
** BCSL CODE GENERATOR VERSION 1.0
**
*****
*
*      SIMULATE
*
*** DEFINITION SEGMENT
*
D1  NAMEIA      I,0,0
    INITIAL    HXSD1(1,4),90/HXSD1(1,3),100/HXSD1(1,4),180      B0001
    INITIAL    HXSD1(1,5),200/HXSD1(1,6),270/HXSD1(2,1),90      B0002
    INITIAL    HXSD1(2,3),100/HXSD1(2,4),260/HXSD1(2,5),180      B0004
    INITIAL    HXSD1(2,6),270/HXSD1(3,1),100/HXSD1(3,2),100      B0005
    INITIAL    HXSD1(3,4),100/HXSD1(3,5),100/HXSD1(3,6),160      B0006
    INITIAL    HXSD1(4,1),180/HXSD1(4,2),200/HXSD1(4,3),100      B0008
    INITIAL    HXSD1(4,5),90/HXSD1(4,6),270/HXSD1(5,1),200      B0009
    INITIAL    HXSD1(5,4),180/HXSD1(5,3),100/HXSD1(5,4),90      B0010
    INITIAL    HXSD1(5,6),100/HXSD1(6,1),270/HXSD1(6,2),270      B0011
    INITIAL    HXSD1(6,3),180/HXSD1(6,4),100/HXSD1(6,5),100      B0012
*
*** NAME SEGMENT
*
    DEFNAME    TRANSPORT,50,1,HXSD1,6      B0014
    DEFSTAT    STATION1,1,3      B0015
    DEFSTAT    STATION2,2,3      B0016
    DEFSTAT    STATION3,3,4      B0017
    DEFSTAT    STATION4,4,4      B0018
    DEFSTAT    STATION5,5,1      B0019
    DEFSTAT    IOSTATION,6      B0020
    TAB1 TABLE    H1,200,20,20      B0021
    TAB2 TABLE    H1,100,20,20      B0022
    TAB3 TABLE    H1,200,20,20      B0023
*
*** NAME SEGMENT
*
    GENERATE    50/.3,DSSEXPONENTIAL.F,,,,,JOB1      B0025
    ENSTAT      IOSTATION      B0026
    REQTRANS    TRANSPORT,STATION3      B0027
    REQSTAT     STATION3,1,50.,DSSELANG.F,2      B0028
    REQTRANS    TRANSPORT,STATION1      B0029
    REQSTAT     STATION1,1,60.,DSSELANG.F,2      B0030
    REQTRANS    TRANSPORT,STATION2      B0031
    REQSTAT     STATION2,1,85.,DSSELANG.F,2      B0032
    REQTRANS    TRANSPORT,STATION5      B0033
    REQSTAT     STATION5,1,50.,DSSELANG.F,2      B0034
    REQTRANS    TRANSPORT,IOSTATION      B0035

```

Figure 4-4a. BCSL Model for the Flexible Manufacturing System

```

LEAVESTAT IOSTATION B0036
TABULATE TAB1 B0037
TERMINATE B0038
*
*
*** NEW SEGMENT
*
GENERATE 50/.5,DSSEXPNENTIAL.F,,,,,JOB2 B0040
ENTERSTAT IOSTATION B0041
REQTRANS TRANSPORT,STATION4 B0042
REQSTAT STATION4,1,110.,DSSEBLANG.F,2 B0043
REQTRANS TRANSPORT,STATION1 B0044
REQSTAT STATION1,1,80.,DSSEBLANG.F,2 B0045
REQTRANS TRANSPORT,STATION3 B0046
REQSTAT STATION3,1,75.,DSSEBLANG.F,2 B0047
LEAVESTAT IOSTATION B0048
TABULATE TAB2 B0049
TERMINATE B0068
*
*
*** NEW SEGMENT
*
GENERATE 50/.2,DSSEXPNENTIAL.F,,,,,JOB3 B0052
ENTERSTAT IOSTATION B0053
REQTRANS TRANSPORT,STATION2 B0054
REQSTAT STATION2,1,120.,DSSEBLANG.F,2 B0055
REQTRANS TRANSPORT,STATION5 B0056
REQSTAT STATION5,1,25.,DSSEBLANG.F,2 B0057
REQTRANS TRANSPORT,STATION1 B0058
REQSTAT STATION1,1,70.,DSSEBLANG.F,2 B0059
REQTRANS TRANSPORT,STATION4 B0060
REQSTAT STATION4,1,90.,DSSEBLANG.F,2 B0061
REQTRANS TRANSPORT,STATION3 B0062
REQSTAT STATION3,1,100.,DSSEBLANG.F,2 B0063
LEAVESTAT IOSTATION B0064
TABULATE TAB3 B0065
TERMINATE B0066
*
*
*** NEW SEGMENT
*
GENERATE 2400,,,,,CLUCA B0071
TERMINATE 1 B0072
*
*
*** CONTROL SEGMENT
*
START 1 B0074

```

Figure 4-4b. BCSL Model for the Flexible Manufacturing System (Cont'd)

4.1.3 Discussion Of SIMSCRIPT Equivalent For The FMS Model - The preamble defines the transactions and their transaction generators as processes. There are three transaction (job) types: job1, job2, and job3. In this Flexible Manufacturing System each work station and transporter is defined as a resource and several variables are defined to keep track of needed statistics for the resources and their queues.

The total delay, due to unavailability of workstations and transporter, for each job type is defined as statistical variables. Therefore an automatic print out of their values will be done at the end of the simulation run. The distance table is defined as a matrix, in addition to the usual SNA definitions. Three Table variables are defined as real variables for the TAB1, TAB7, and TAB3 tables. Tally and Accumulate statements are generated for resources, queues, and statistical variables (including Total delays in the system). Finally, Histogram statements will keep statistics on table entries.

The Initialization routine establishes the resources capacity, initializes the distance table and original location of the transporter. Finally the generator transactions are activated. The output

routines generate extensive reports on resource utilization, transaction interarrival time, Tables of total system time and total delays for each job type.

There is a transaction generator for each job type which generates transactions with exponentially distributed interarrival times. Each transaction enters the Flexible Manufacturing System through an ENTERSTAT block, which does not cause any delays in this example. This block initiates the statistical calculations for delays in the model for the job type. Then the transaction requests a transporter to visit the first workstation, it may have to wait for the transporter to arrive. Then the transporter time is added to the total delay before it reaches the destination.

In the next step, the transaction has arrived at the destination work station and the work will take an Erlang distribution time. The sequence of "request for transporter" block and "request for workstation" block is repeated for all the workstations in the job path. The Transaction finally leaves the FMS model through a LEAVESTAT block which calculates the total workstation delay and transporter delay.

The TABULATE block adds the total time spent in the model to the corresponding table for the job type. The clock transaction steps the simulation by decrementing the "stop counter" and starts the report generation. Figure 4-5 contains the SIMSCRIPT equivalent for this Flexible Manufacturing System.


```

*.....*
*.....*
*.....* SIMSCRIPT CODE GENERATED BY .....*
*.....*..... HCSL TO SIMSCRIPT CROSS COMPILER .....*
*.....*..... VERSION 1.2 .....*
*.....*
*.....*
1 PREAMBLE
2 NORMALLY MODE IS REAL
3 ACCESSPS
4 EVERY TRAN.JOB1 HAS A MONITOR.V
5 AND MAY BELONG TO THE BLOCKED.JOB1
6 EVERY TRAN.JOB2 HAS A MONITOR.V
7 AND MAY BELONG TO THE BLOCKED.JOB2
8 EVERY TRAN.JOB3 HAS A MONITOR.V
9 AND MAY BELONG TO THE BLOCKED.JOB3
10 EVERY TRAN.CLOCK HAS A MONITOR.V
11 AND MAY BELONG TO THE BLOCKED.CLOCK
12 THE SYSTEM OWNS THE BLOCKED.JOB1
13 THE SYSTEM OWNS THE BLOCKED.JOB2
14 THE SYSTEM OWNS THE BLOCKED.JOB3
15 THE SYSTEM OWNS THE BLOCKED.CLOCK
16 EVERY GEN.JOB1 HAS A STOP.FLAG.JOB1
17 EVERY GEN.JOB2 HAS A STOP.FLAG.JOB2
18 EVERY GEN.JOB3 HAS A STOP.FLAG.JOB3
19 EVERY GEN.CLOCK HAS A STOP.FLAG.CLOCK
20 RESOURCES
21 EVERY TRANSPORT HAS A NUM.OF.TRANSPORT
22 EVERY STATION1 HAS A NUM.OF.STATION1
23 EVERY STATION2 HAS A NUM.OF.STATION2
24 EVERY STATION3 HAS A NUM.OF.STATION3
25 EVERY STATION4 HAS A NUM.OF.STATION4
26 EVERY STATION5 HAS A NUM.OF.STATION5
27 EVERY IOSTATION HAS A NUM.OP.IOSTATION
28 DEFINE TIMEIN.F.TRANSPORT AS A REAL VARIABLE
29 DEFINE TIMEIN.Q.TRANSPORT AS A REAL VARIABLE
30 DEFINE NUM.EN1.Q.TRANSPORT AS AN INTEGER VARIABLE
31 DEFINE CAPACITY.TRANSPORT1 AS AN INTEGER VARIABLE
32 DEFINE TIMEIN.F.STATION1 AS A REAL VARIABLE
33 DEFINE TIMEIN.Q.STATION1 AS A REAL VARIABLE
34 DEFINE NUM.EN1.Q.STATION1 AS AN INTEGER VARIABLE
35 DEFINE CAPACITY.STATION1 AS AN INTEGER VARIABLE
36 DEFINE TIMEIN.F.STATION2 AS A REAL VARIABLE
37 DEFINE TIMEIN.Q.STATION2 AS A REAL VARIABLE
38 DEFINE NUM.EN1.Q.STATION2 AS AN INTEGER VARIABLE
39 DEFINE CAPACITY.STATION2 AS AN INTEGER VARIABLE
40 DEFINE TIMEIN.F.STATION3 AS A REAL VARIABLE
41 DEFINE TIMEIN.Q.STATION3 AS A REAL VARIABLE
42 DEFINE NUM.EN1.Q.STATION3 AS AN INTEGER VARIABLE
43 DEFINE CAPACITY.STATION3 AS AN INTEGER VARIABLE
44 DEFINE TIMEIN.F.STATION4 AS A REAL VARIABLE
45 DEFINE TIMEIN.Q.STATION4 AS A REAL VARIABLE
46 DEFINE NUM.EN1.Q.STATION4 AS AN INTEGER VARIABLE
47 DEFINE CAPACITY.STATION4 AS AN INTEGER VARIABLE
48 DEFINE TIMEIN.F.STATION5 AS A REAL VARIABLE

```

Figure 4-5a. SIMSCRIPT Model for the Flexible Manufacturing System

```

49 DEFINE TIMEIN.Q.STATIONS AS A REAL VARIABLE
50 DEFINE NUM.ENT.Q.STATIONS AS AN INTEGER VARIABLE
51 DEFINE CAPACITY.STATIONS AS AN INTEGER VARIABLE
52 DEFINE TIMEIN.F.JOSTATION AS A REAL VARIABLE
53 DEFINE TIMEIN.Q.JOSTATION AS A REAL VARIABLE
54 DEFINE NUM.ENT.Q.JOSTATION AS AN INTEGER VARIABLE
55 DEFINE CAPACITY.JOSTATION AS AN INTEGER VARIABLE
56 **DEFINE VARIABLES AND LEFT MONITORED VARIABLES
57 DEFINE STOP.FLAG AS INTEGER VARIABLE
58 DEFINE TRANSPORT.LOC.NUM AS INTEGER VARIABLE
59 **DEFINE SMA FOR QUEUES
60 **DEFINE TEST BLOCK FLAGS
61 **DEFINE STATISTICAL VARIABLES
62 DEFINE JOB1.WORKSTATION.DELAY AS A REAL VARIABLE
63 DEFINE JOB1.TRANSPORT.DELAY AS A REAL VARIABLE
64 DEFINE JOB2.WORKSTATION.DELAY AS A REAL VARIABLE
65 DEFINE JOB2.TRANSPORT.DELAY AS A REAL VARIABLE
66 DEFINE JOB3.WORKSTATION.DELAY AS A REAL VARIABLE
67 DEFINE JOB3.TRANSPORT.DELAY AS A REAL VARIABLE
68 **DEFINE RANDOM VARIABLES
69 **DEFINE MATRICES
70 DEFINE MADD1 AS AN 2-DIMENSIONAL INTEGER ARRAY
71 **DEFINE GPSS GLOBAL VARIABLES
72 DEFINE P1 TO BEAN P_ARRAY(1)
73 DEFINE P2 TO BEAN P_ARRAY(2)
74 DEFINE P3 TO BEAN P_ARRAY(3)
75 DEFINE P4 TO BEAN P_ARRAY(4)
76 DEFINE P5 TO BEAN P_ARRAY(5)
77 DEFINE P6 TO BEAN P_ARRAY(6)
78 DEFINE P7 TO BEAN P_ARRAY(7)
79 DEFINE P8 TO BEAN P_ARRAY(8)
80 DEFINE P9 TO BEAN P_ARRAY(9)
81 DEFINE P10 TO BEAN P_ARRAY(10)
82 DEFINE X1 TO BEAN X_ARRAY(1)
83 DEFINE X2 TO BEAN X_ARRAY(2)
84 DEFINE X3 TO BEAN X_ARRAY(3)
85 DEFINE X4 TO BEAN X_ARRAY(4)
86 DEFINE X5 TO BEAN X_ARRAY(5)
87 DEFINE X6 TO BEAN X_ARRAY(6)
88 DEFINE X7 TO BEAN X_ARRAY(7)
89 DEFINE X8 TO BEAN X_ARRAY(8)
90 DEFINE X9 TO BEAN X_ARRAY(9)
91 DEFINE X10 TO BEAN X_ARRAY(10)
92 DEFINE UNITS TO BEAN MINUTES
93 DEFINE MONITOR.V AS TEXT VARIABLE
94 DEFINE S1 AS A REAL VARIABLE
95 DEFINE C1 TO BEAN TIME.V*HOURS.V*MINUTES.V
96 **TRANSACTION DEFINITION
97 DEFINE C.TRAN.CMT.JOB1 AS AN INTEGER VARIABLE
98 DEFINE C.TRAN.CM1.JOB1 AS AN INTEGER VARIABLE
99 DEFINE SEED.TRAN.JOB1 AS AN INTEGER VARIABLE
100 DEFINE C.TRAN.CMT.JOB2 AS AN INTEGER VARIABLE
101 DEFINE C.TRAN.CM1.JOB2 AS AN INTEGER VARIABLE
102 DEFINE SEED.TRAN.JOB2 AS AN INTEGER VARIABLE
103 DEFINE C.TRAN.CMT.JOB3 AS AN INTEGER VARIABLE
104 DEFINE C.TRAN.CM1.JOB3 AS AN INTEGER VARIABLE

```

Figure 4-5a. SIMSCRIPT Model for the Flexible Manufacturing System (Cont'd)

```

105 DEFINE SEED.TMAN.JOB3 AS AN INTEGER VARIABLE
106 DEFINE TMAN.CM1.CLOCK AS AN INTEGER VARIABLE
107 DEFINE C.TMAN.CM1.CLOCK AS AN INTEGER VARIABLE
108 DEFINE SEED.TMAN.CLOCK AS AN INTEGER VARIABLE
109 **DEFINE TABLE VARIABLES
110 DEFINE TAB1 AS REAL VARIABLE
111 DEFINE TAB2 AS REAL VARIABLE
112 DEFINE TAB3 AS REAL VARIABLE
113 **TALLY AND ACCUMULATE FOR RESOURCES
114 ACCUMULATE UTIL.TRANSPORT1 AS AVERAGE AND
115 MAX.MX.TRANSPORT AS MAXIMUM OF M.X.TRANSPORT
116 ACCUMULATE AVG.Q.LEN.TRANSPORT1 AS AVERAGE AND
117 MAX.Q.LEN.TRANSPORT1 AS MAXIMUM OF M.Q.TRANSPORT
118 TALLY AVG.TIMEIN.P.TRANSPORT AS MEAN OF TIMEIN.P.TRANSPORT
119 TALLY AVG.TIMEIN.Q.TRANSPORT AS MEAN OF TIMEIN.Q.TRANSPORT
120 ACCUMULATE UTIL.STATION1 AS AVERAGE AND
121 MAX.MX.STATION1 AS MAXIMUM OF M.X.STATION1
122 ACCUMULATE AVG.Q.LEN.STATION1 AS AVERAGE AND
123 MAX.Q.LEN.STATION1 AS MAXIMUM OF M.Q.STATION1
124 TALLY AVG.TIMEIN.P.STATION1 AS MEAN OF TIMEIN.P.STATION1
125 TALLY AVG.TIMEIN.Q.STATION1 AS MEAN OF TIMEIN.Q.STATION1
126 ACCUMULATE UTIL.STATION2 AS AVERAGE AND
127 MAX.MX.STATION2 AS MAXIMUM OF M.X.STATION2
128 ACCUMULATE AVG.Q.LEN.STATION2 AS AVERAGE AND
129 MAX.Q.LEN.STATION2 AS MAXIMUM OF M.Q.STATION2
130 TALLY AVG.TIMEIN.P.STATION2 AS MEAN OF TIMEIN.P.STATION2
131 TALLY AVG.TIMEIN.Q.STATION2 AS MEAN OF TIMEIN.Q.STATION2
132 ACCUMULATE UTIL.STATION3 AS AVERAGE AND
133 MAX.MX.STATION3 AS MAXIMUM OF M.X.STATION3
134 ACCUMULATE AVG.Q.LEN.STATION3 AS AVERAGE AND
135 MAX.Q.LEN.STATION3 AS MAXIMUM OF M.Q.STATION3
136 TALLY AVG.TIMEIN.P.STATION3 AS MEAN OF TIMEIN.P.STATION3
137 TALLY AVG.TIMEIN.Q.STATION3 AS MEAN OF TIMEIN.Q.STATION3
138 ACCUMULATE UTIL.STATION4 AS AVERAGE AND
139 MAX.MX.STATION4 AS MAXIMUM OF M.X.STATION4
140 ACCUMULATE AVG.Q.LEN.STATION4 AS AVERAGE AND
141 MAX.Q.LEN.STATION4 AS MAXIMUM OF M.Q.STATION4
142 TALLY AVG.TIMEIN.P.STATION4 AS MEAN OF TIMEIN.P.STATION4
143 TALLY AVG.TIMEIN.Q.STATION4 AS MEAN OF TIMEIN.Q.STATION4
144 ACCUMULATE UTIL.STATION5 AS AVERAGE AND
145 MAX.MX.STATION5 AS MAXIMUM OF M.X.STATION5
146 ACCUMULATE AVG.Q.LEN.STATION5 AS AVERAGE AND
147 MAX.Q.LEN.STATION5 AS MAXIMUM OF M.Q.STATION5
148 TALLY AVG.TIMEIN.P.STATION5 AS MEAN OF TIMEIN.P.STATION5
149 TALLY AVG.TIMEIN.Q.STATION5 AS MEAN OF TIMEIN.Q.STATION5
150 ACCUMULATE UTIL.IOSTATION AS AVERAGE AND
151 MAX.MX.IOSTATION AS MAXIMUM OF M.X.IOSTATION
152 ACCUMULATE AVG.Q.LEN.IOSTATION AS AVERAGE AND
153 MAX.Q.LEN.IOSTATION AS MAXIMUM OF M.Q.IOSTATION
154 TALLY AVG.TIMEIN.P.IOSTATION AS MEAN OF TIMEIN.P.IOSTATION
155 TALLY AVG.TIMEIN.Q.IOSTATION AS MEAN OF TIMEIN.Q.IOSTATION
156 **TALLY AND ACCUMULATE FOR STATISTICAL VARIABLES
157 TALLY AVG.JOB1.WORKSTATION.DELAY AS AVERAGE OF JOB1.WORKSTATION.DELAY
158 TALLY AVG.JOB1.TRANSPORT.DELAY AS AVERAGE OF JOB1.TRANSPORT.DELAY
159 TALLY AVG.JOB2.WORKSTATION.DELAY AS AVERAGE OF JOB2.WORKSTATION.DELAY
160 TALLY AVG.JOB2.TRANSPORT.DELAY AS AVERAGE OF JOB2.TRANSPORT.DELAY
  
```

Figure 4-5a. SIMSCRIPT Model for the Flexible Manufacturing System (Cont'd)

```

161 TALLY AVG.JOB3.WORKSTATION.DELAY AS AVERAGE OF JOB3.WORKSTATION.DELAY
162 TALLY AVG.JOB3.TRANSPORT.DELAY AS AVERAGE OF JOB3.TRANSPORT.DELAY
163 **TALLY AND ACCUMULATE FOR QUEUES
164 DEFINE GEN.TIME.JOB1 AS A REAL VARIABLE
165 TALLY AVG.GEN.TIME.JOB1 AS MEAN OF GEN.TIME.JOB1
166 DEFINE GEN.TIME.JOB2 AS A REAL VARIABLE
167 TALLY AVG.GEN.TIME.JOB2 AS MEAN OF GEN.TIME.JOB2
168 DEFINE GEN.TIME.JOB3 AS A REAL VARIABLE
169 TALLY AVG.GEN.TIME.JOB3 AS MEAN OF GEN.TIME.JOB3
170 DEFINE GEN.TIME.CLOCK AS A REAL VARIABLE
171 TALLY AVG.GEN.TIME.CLOCK AS MEAN OF GEN.TIME.CLOCK
172 **HISTOGRAM FOR TABLES
173 TALLY TAB1.HISTO(200 TO 560 BY 20)
174 AS THE HISTOGRAM AND TAB1.AVG AS AVERAGE AND
175 TAB1.SID AS STD.DEV AND TAB1.NUM.ENT AS NUMBER OF TAB1
176 TALLY TAB2.HISTO(100 TO 460 BY 20)
177 AS THE HISTOGRAM AND TAB2.AVG AS AVERAGE AND
178 TAB2.SID AS STD.DEV AND TAB2.NUM.ENT AS NUMBER OF TAB2
179 TALLY TAB3.HISTO(200 TO 560 BY 20)
180 AS THE HISTOGRAM AND TAB3.AVG AS AVERAGE AND
181 TAB3.SID AS STD.DEV AND TAB3.NUM.ENT AS NUMBER OF TAB3
182 END
  
```

C R O S S - R E F E R E N C E

NAME	TYPE	MODE
AVG.GEN.TIME.CLOCK	ROUTINE	DOUBLE
AVG.GEN.TIME.JOB1	ROUTINE	DOUBLE
AVG.GEN.TIME.JOB2	ROUTINE	DOUBLE
AVG.GEN.TIME.JOB3	ROUTINE	DOUBLE
AVG.JOB1.TRANSPORT.DELAY	ROUTINE	DOUBLE
AVG.JOB1.WORKSTATION.DELAY	ROUTINE	DOUBLE
AVG.JOB2.TRANSPORT.DELAY	ROUTINE	DOUBLE
AVG.JOB2.WORKSTATION.DELAY	ROUTINE	DOUBLE
AVG.JOB3.TRANSPORT.DELAY	ROUTINE	DOUBLE
AVG.JOB3.WORKSTATION.DELAY	ROUTINE	DOUBLE
AVG.Q.LEN.STATION	FUNCTION ATTRIBUTE	DOUBLE
AVG.Q.LEN.STATION1	FUNCTION ATTRIBUTE	DOUBLE
AVG.Q.LEN.STATION2	FUNCTION ATTRIBUTE	DOUBLE
AVG.Q.LEN.STATION3	FUNCTION ATTRIBUTE	DOUBLE
AVG.Q.LEN.STATION4	FUNCTION ATTRIBUTE	DOUBLE
AVG.Q.LEN.STATION5	FUNCTION ATTRIBUTE	DOUBLE
AVG.Q.LEN.TRANSPORT	FUNCTION ATTRIBUTE	DOUBLE
AVG.TIMEIN.F.STATION	ROUTINE	DOUBLE
AVG.TIMEIN.F.STATION1	ROUTINE	DOUBLE
AVG.TIMEIN.F.STATION2	ROUTINE	DOUBLE
AVG.TIMEIN.F.STATION3	ROUTINE	DOUBLE
AVG.TIMEIN.F.STATION4	ROUTINE	DOUBLE
AVG.TIMEIN.F.STATION5	ROUTINE	DOUBLE
AVG.TIMEIN.F.TRANSPORT	ROUTINE	DOUBLE
AVG.TIMEIN.Q.STATION	ROUTINE	DOUBLE
AVG.TIMEIN.Q.STATION1	ROUTINE	DOUBLE
AVG.TIMEIN.Q.STATION2	ROUTINE	DOUBLE
AVG.TIMEIN.Q.STATION3	ROUTINE	DOUBLE

Figure 4-5a. SIMSCRIPT Model for the Flexible Manufacturing System (Cont'd)

```

1 ROUTINE INITIALIZE
2 NORMALLY MODE IS INTEGER
3 RESERVE N1001 AS 6 BY 6
4 LET CAPACITY.TRANSPORT = 1
5 CREATE EVERY TRANSPORT(1)
6 LET U.TRANSPORT(1) = 1
7 LET CAPACITY.STATION1 = 3
8 CREATE EVERY STATION1(1)
9 LET U.STATION1(1) = 3
10 LET CAPACITY.STATION2 = 3
11 CREATE EVERY STATION2(1)
12 LET U.STATION2(1) = 3
13 LET CAPACITY.STATION3 = 4
14 CREATE EVERY STATION3(1)
15 LET U.STATION3(1) = 4
16 LET CAPACITY.STATION4 = 4
17 CREATE EVERY STATION4(1)
18 LET U.STATION4(1) = 4
19 LET CAPACITY.STATION5 = 1
20 CREATE EVERY STATION5(1)
21 LET U.STATION5(1) = 1
22 LET CAPACITY.IOSTATION = 1
23 CREATE EVERY IOSTATION(1)
24 LET U.IOSTATION(1) = 1
25 LET HXSD1(1,2) = 90
26 LET HXSD1(1,3) = 100
27 LET HXSD1(1,4) = 180
28 LET HXSD1(1,5) = 200
29 LET HXSD1(1,6) = 270
30 LET HXSD1(2,1) = 90
31 LET HXSD1(2,3) = 100
32 LET HXSD1(2,4) = 200
33 LET HXSD1(2,5) = 160
34 LET HXSD1(2,6) = 270
35 LET HXSD1(3,1) = 100
36 LET HXSD1(3,2) = 100
37 LET HXSD1(3,4) = 100
38 LET HXSD1(3,5) = 100
39 LET HXSD1(3,6) = 160
40 LET HXSD1(4,1) = 180
41 LET HXSD1(4,2) = 200
42 LET HXSD1(4,3) = 100
43 LET HXSD1(4,5) = 90
44 LET HXSD1(4,6) = 270
45 LET HXSD1(5,1) = 200
46 LET HXSD1(5,2) = 180
47 LET HXSD1(5,3) = 100
48 LET HXSD1(5,4) = 90
49 LET HXSD1(5,6) = 100
50 LET HXSD1(6,1) = 270
51 LET HXSD1(6,2) = 270
52 LET HXSD1(6,3) = 180
53 LET HXSD1(6,4) = 100
54 LET HXSD1(6,5) = 100
55 LET TRANSPORT_LOC_NUM = 6
56 LET TRAN.CRTA.JOB1 = 0

```

Figure 4-5b. SIMSCRIPT Model for the Flexible Manufacturing System (Cont'd)

```

57 LET TRAN.CNTL.JOB2 = 0
58 LET TRAN.CNTL.JOB3 = 0
59 LET TRAN.CNTL.CLOCK = 0
60 LET STOP.FLAG = 1
61 ACTIVATE A GEN.JOB1 NOW
62 ACTIVATE A GEN.JOB2 NOW
63 ACTIVATE A GEN.JOB3 NOW
64 ACTIVATE A GEN.CLOCK NOW
65 END
  
```

C R O S S - R E F E R E N C E

NAME	TYPE	MODE
CAPACITY.STATION	GLOBAL VARIABLE	INTEGER
CAPACITY.STATION1	GLOBAL VARIABLE	INTEGER
CAPACITY.STATION2	GLOBAL VARIABLE	INTEGER
CAPACITY.STATION3	GLOBAL VARIABLE	INTEGER
CAPACITY.STATION4	GLOBAL VARIABLE	INTEGER
CAPACITY.STATION5	GLOBAL VARIABLE	INTEGER
CAPACITY.TRANSPORT	GLOBAL VARIABLE	INTEGER
GEN.CLOCK	PROCESS NOTICE	
	* GLOBAL VARIABLE	INTEGER
GEN.JOB1	PROCESS NOTICE	
	* GLOBAL VARIABLE	INTEGER
GEN.JOB2	PROCESS NOTICE	
	* GLOBAL VARIABLE	INTEGER
GEN.JOB3	PROCESS NOTICE	
	* GLOBAL VARIABLE	INTEGER
INITIALIZE	ROUTINE	INTEGER
IOSATION	RESOURCE	
MSFD1	GLOBAL VARIABLE	(2-D) INTEGER
	* GLOBAL VARIABLE	(2-D) INTEGER
STATION1	RESOURCE	
STATION2	RESOURCE	
STATION3	RESOURCE	
STATION4	RESOURCE	
STATION5	RESOURCE	
STOP.FLAG	GLOBAL VARIABLE	INTEGER
TRAN.CNTL.CLOCK	GLOBAL VARIABLE	INTEGER
TRAN.CNTL.JOB1	GLOBAL VARIABLE	INTEGER
TRAN.CNTL.JOB2	GLOBAL VARIABLE	INTEGER
TRAN.CNTL.JOB3	GLOBAL VARIABLE	INTEGER
TRANSPORT	RESOURCE	
TRANSPCAT_LOL_BUG	GLOBAL VARIABLE	INTEGER
U.IOSATION	PERMANENT ATTRIBUTE	(1-D) INTEGER
U.STATION1	PERMANENT ATTRIBUTE	(1-D) INTEGER
U.STATION2	PERMANENT ATTRIBUTE	(1-D) INTEGER
U.STATION3	PERMANENT ATTRIBUTE	(1-D) INTEGER
U.STATION4	PERMANENT ATTRIBUTE	(1-D) INTEGER
U.STATION5	PERMANENT ATTRIBUTE	(1-D) INTEGER

Figure 4-5b. SIMSCRIPT Model for the Flexible Manufacturing System (Cont'd)

```

***
**      DEFTRANS    TRANSPORT,50,1,EXSD1,6                    B0014
**      DEFNSTAT    STATION1,1,3                              B0015
**      DEFNSTAT    STATION2,2,3                              B0016
**      DEFNSTAT    STATION3,3,4                              B0017
**      DEFNSTAT    STATION4,4,4                              B0018
**      DEFNSTAT    STATION5,5,1                              B0019
**      DEFNSTAT    IOSTATION,6                                B0020
** TAB1 TABLE     N1,200,20,20                                B0021
** TAB2 TABLE     N1,100,20,20                                B0022
** TAB3 TABLE     N1,200,20,20                                B0023
***
***
**** NEW SEGMENT
***
***
**      GENERATE    50/.3,DSSEXPOENTIAL.P,,,,,JOB1            B0025

1    PROCESS GEN.JOB1
2    LET STOP.FLAG.JOB1(GEN.JOB1) = 1
3    LET SEED.TRAN.JOB1 = 1
4    WAIT 0 UNITS
5    UNTIL STOP.FLAG.JOB1(GEN.JOB1) <= 0
6    DO
7    DEFINE GEN1.JOB1 AS A REAL VARIABLE
8    LET GEN1.JOB1 = TIME.V
9    WAIT EXPONENTIAL.P (50/.3,
10    SEED.TRAN.JOB1) UNITS
11    LET GEN.TIME.JOB1=(TIME.V - GEN1.JOB1)*HOURS.V*MINUTES.V
12    ACTIVATE A TRAN.JOB1 NOW
13    LET TRAN.CNTX.JOB1=TRAN.CNTX.JOB1 + 1
14    LET C.TRAN.CNT.JOB1=C.TRAN.CNT.JOB1 + 1
15    LOGF
16    END

```

Figure 4-5c. SIMSCRIPT Model for the Flexible Manufacturing System (Cont'd)

```

1  PROCESS TRAN.JOB1
2  DEFINE P_ARRAY AS INTEGER,1-DIMENSIONAL ARRAY
3  RESERVE P_ARRAY AS 10
4  DEFINE TIME1 AS A REAL VARIABLE
5  TIME1 = TIME.V
6  **            ENLISTA    IOSTATION                            B0026
7
8  LET CURRENT_WS_NUM = 6
9  LET TIME.Q.IOSTATION = TIME.V
10 LET NUM.ENT.Q.IOSTATION = NUM.ENT.Q.IOSTATION + 1
11 REQUEST 1 IOSTATION
12 LET TIMEIN.Q.IOSTATION=(TIME.V - TIME.Q.IOSTATION)*HOURS.V*MINUTES.V
13 LET TIME.F.IOSTATION = TIME.V
14 ADD TIMEIN.Q.IOSTATION TO TOTAL_WS_DELAY
15 SOLA UNIFORM.F((0-0)/1.0,(0+0)/1.0,
16 SEED.TRAN.JOB1) UNITS
17 RELINQUISH 1 IOSTATION
18 LET TIMEIN.F.IOSTATION=(TIME.V - TIME.F.IOSTATION)*HOURS.V*MINUTES.V
19 **            REQTRANS    TRANSPORT,STATION3                    B0027
20
21 LET REQ.TIME = TIME.V
22 LET TIME.Q.TRANSPORT = TIME.V
23 LET NUM.ENT.Q.TRANSPORT = NUM.ENT.Q.TRANSPORT + 1
24 REQUEST 1 TRANSPORT
25 LET TIMEIN.Q.TRANSPORT=(TIME.V - TIME.Q.TRANSPORT)*HOURS.V*MINUTES.V
26 LET TIME.F.TRANSPORT = TIME.V
27 WAIT MIXD1(TRANSPORT_LOC_NUM,CURRENT_WS_NUM)/50 UNITS
28 LET TRANSPORT_LOC_NUM = CURRENT_WS_NUM
29 LET TIME.TEMP = (TIME.V - REQ.TIME)*HOURS.V*MINUTES.V
30 ADD TIME.TEMP TO TOTAL_TRANS_DELAY
31 LET NEXT_WS_NUM = 3
32 WAIT MIXD1(CURRENT_WS_NUM,NEXT_WS_NUM)/50 UNITS
33 LET TRANSPORT_LOC_NUM = NEXT_WS_NUM
34 RELINQUISH 1 TRANSPORT
35 LET TIMEIN.F.TRANSPORT=(TIME.V - TIME.F.TRANSPORT)*HOURS.V*MINUTES.V
36 **            REQSTAT    STATION3,1,50.,DSSELANG.F,2                    B0028
37
38 LET CURRENT_WS_NUM = 3
39 LET TIME.Q.STATION3 = TIME.V
40 LET NUM.ENT.Q.STATION3 = NUM.ENT.Q.STATION3 + 1
41 REQUEST 1 STATION3
42 LET TIMEIN.Q.STATION3=(TIME.V - TIME.Q.STATION3)*HOURS.V*MINUTES.V
43 LET TIME.F.STATION3 = TIME.V
44 ADD TIMEIN.Q.STATION3 TO TOTAL_WS_DELAY
45 SOLA ERLANG.F(50.,2,SEED.TRAN.JOB1) UNITS
46 RELINQUISH 1 STATION3
47 LET TIMEIN.F.STATION3=(TIME.V - TIME.F.STATION3)*HOURS.V*MINUTES.V
48 **            REQTRANS    TRANSPORT,STATION1                    B0029
49
50 LET REQ.TIME = TIME.V
51 LET TIME.Q.TRANSPORT = TIME.V
52 LET NUM.ENT.Q.TRANSPORT = NUM.ENT.Q.TRANSPORT + 1
53 REQUEST 1 TRANSPORT
54 LET TIMEIN.Q.TRANSPORT=(TIME.V - TIME.Q.TRANSPORT)*HOURS.V*MINUTES.V
55 LET TIME.F.TRANSPORT = TIME.V
56 WAIT MIXD1(TRANSPORT_LOC_NUM,CURRENT_WS_NUM)/50 UNITS

```

Figure 4-5d. SIMSCRIPT Model for the Flexible Manufacturing System (Cont'd)


```

57 LET TRANSPORT1_LOC_NUM = CURRENT_WS_NUM
58 LET TIME_TEMP = (TIME.V - REQ.TIME)*HOURS.V*MINUTES.V
59 ADD TIME_TEMP TO TOTAL_TRANS_DELAY
60 LET NEXT_WS_NUM = 1
61 WAIT M3SD1(CURRENT_WS_NUM,NEXT_WS_NUM)/50 UNITS
62 LET TRANSPORT_LOC_NUM = NEXT_WS_NUM
63 RELINQUISH 1 TRANSPORT
64 LET TIMEIN.F.TRANSPORT=(TIME.V - TIM.F.TRANSPORT)*HOURS.V*MINUTES.V
65 **        REQSTAT    STATION1,1,00.,DSSELANG.F,2                      B0030
66
67 LET CURRENT_WS_NUM = 1
68 LET TIM.Q.STATION1 = TIME.V
69 LET NUM.ENT.Q.STATION1 = NUM.ENT.Q.STATION1 + 1
70 REQUEST 1 STATION1
71 LET TIMEIN.Q.STATION1=(TIME.V - TIM.Q.STATION1)*HOURS.V*MINUTES.V
72 LET TIM.F.STATION1 = TIME.V
73 ADD TIMEIN.Q.STATION1 TO TOTAL_WS_DELAY
74 WORK ELANG.F(60.,2,SEED.TRAN.JCB1) UNITS
75 RELINQUISH 1 STATION1
76 LET TIMEIN.F.STATION1=(TIME.V - TIM.F.STATION1)*HOURS.V*MINUTES.V
77 **        REQTRANS    TRANSPORT,STATION2                      B0031
78
79 LET REQ.TIME = TIME.V
80 LET TIM.Q.TRANSPORT = TIME.V
81 LET NUM.ENT.Q.TRANSPORT = NUM.ENT.Q.TRANSPORT + 1
82 REQUEST 1 TRANSPORT
83 LET TIMEIN.Q.TRANSPORT=(TIME.V - TIM.Q.TRANSPORT)*HOURS.V*MINUTES.V
84 LET TIM.F.TRANSPORT = TIME.V
85 WAIT M3SD1(TRANSPORT_LOC_NUM,CURRENT_WS_NUM)/50 UNITS
86 LET TRANSPORT_LOC_NUM = CURRENT_WS_NUM
87 LET TIME_TEMP = (TIME.V - REQ.TIME)*HOURS.V*MINUTES.V
88 ADD TIME_TEMP TO TOTAL_TRANS_DELAY
89 LET NEXT_WS_NUM = 2
90 WAIT M3SD1(CURRENT_WS_NUM,NEXT_WS_NUM)/50 UNITS
91 LET TRANSPORT_LOC_NUM = NEXT_WS_NUM
92 RELINQUISH 1 TRANSPORT
93 LET TIMEIN.F.TRANSPORT1=(TIME.V - TIM.F.TRANSPORT)*HOURS.V*MINUTES.V
94 **        REQSTAT    STATION2,1,05.,DSSELANG.F,2                      B0032
95
96 LET CURRENT_WS_NUM = 2
97 LET TIM.Q.STATION2 = TIME.V
98 LET NUM.ENT.Q.STATION2 = NUM.ENT.Q.STATION2 + 1
99 REQUEST 1 STATION2
100 LET TIMEIN.Q.STATION2=(TIME.V - TIM.Q.STATION2)*HOURS.V*MINUTES.V
101 LET TIM.F.STATION2 = TIME.V
102 ADD TIMEIN.Q.STATION2 TO TOTAL_WS_DELAY
103 WORK ELANG.F(85.,2,SEED.TRAN.JOB1) UNITS
104 RELINQUISH 1 STATION2
105 LET TIMEIN.F.STATION2=(TIME.V - TIM.F.STATION2)*HOURS.V*MINUTES.V
106 **        REQTRANS    TRANSPORT,STATION5                      B0033
107
108 LET REQ.TIME = TIME.V
109 LET TIM.Q.TRANSPORT1 = TIME.V
110 LET NUM.ENT.Q.TRANSPORT = NUM.ENT.Q.TRANSPORT + 1
111 REQUEST 1 TRANSPORT
112 LET TIMEIN.Q.TRANSPORT=(TIME.V - TIM.Q.TRANSPORT)*HOURS.V*MINUTES.V

```

Figure 4-5d. SIMSCRIPT Model for the Flexible Manufacturing System (Cont'd)

```

113 LET TIN.F.TRANSPORT = TIME.V
114 WAIT BXSD1(TRANSPORT_LOC_NUM,CURRENT_WS_NUM)/50 UNITS
115 LET TRANSPORT_LOC_NUM = CURRENT_WS_NUM
116 LET TIME.TEMP = (TIME.V - REQ.TIME)*HOURS.V*MINUTES.V
117 ADD TIME.TEMP TO TOTAL.TRANS.DELAY
118 LET NEXT_WS_NUM = 5
119 WAIT BXSD1(CURRENT_WS_NUM,NEXT_WS_NUM)/50 UNITS
120 LET TRANSPORT_LOC_NUM = NEXT_WS_NUM
121 RELINQUISH 1 TRANSPORT
122 LET TIMEIN.F.TRANSPORT=(TIME.V - TIN.F.TRANSPORT)*HOURS.V*MINUTES.V
123 **      *EQUSTAT STATIONS,1,50.,DSSE=LANG.F,2      B0034
124
125 LET CURRENT_WS_NUM = 5
126 LET TIN.Q.STATIONS = TIME.V
127 LET NUM.ENT.Q.STATIONS = NUM.ENT.Q.STATIONS + 1
128 *EQUSTAT 1 STATIONS
129 LET TIMEIN.Q.STATIONS=(TIME.V - TIN.Q.STATIONS)*HOURS.V*MINUTES.V
130 LET TIN.F.STATIONS = TIME.V
131 ADD TIMEIN.Q.STATIONS TO TOTAL.WS.DELAY
132 *WORK ELANG.F(50.,2,SEED.TRAN.JOB1) UNITS
133 RELINQUISH 1 STATIONS
134 LET TIMEIN.F.STATIONS=(TIME.V - TIN.F.STATIONS)*HOURS.V*MINUTES.V
135 **      *EQUTRANS TRANSPORT,1,STATION      B0035
136
137 LET LEQ.TIME = TIME.V
138 LET TIN.Q.TRANSPORT = TIME.V
139 LET NUM.ENT.Q.TRANSPORT = NUM.ENT.Q.TRANSPORT + 1
140 *EQUDES1 1 TRANSPORT
141 LET TIMEIN.Q.TRANSPORT=(TIME.V - TIN.Q.TRANSPORT)*HOURS.V*MINUTES.V
142 LET TIN.F.TRANSPORT = TIME.V
143 WAIT BXSD1(TRANSPORT_LOC_NUM,CURRENT_WS_NUM)/50 UNITS
144 LET TRANSPORT_LOC_NUM = CURRENT_WS_NUM
145 LET TIME.TEMP = (TIME.V - REQ.TIME)*HOURS.V*MINUTES.V
146 ADD TIME.TEMP TO TOTAL.TRANS.DELAY
147 LET NEXT_WS_NUM = 6
148 WAIT BXSD1(CURRENT_WS_NUM,NEXT_WS_NUM)/50 UNITS
149 LET TRANSPORT_LOC_NUM = NEXT_WS_NUM
150 RELINQUISH 1 TRANSPORT
151 LET TIMEIN.F.TRANSPORT=(TIME.V - TIN.F.TRANSPORT)*HOURS.V*MINUTES.V
152 **      *LEAVES1AT 1,STATION      B0036
153
154 LET JOB1.WORKSTATION.DELAY = TOTAL.WS.DELAY
155 LET JOB1.TRANSPORT.DELAY = TOTAL.TRANS.DELAY
156 **      *TABULATE TAB1      B0037
157
158 LET S1 = (TIME.V - TIN.S1)*HOURS.V*MINUTES.V
159 LET TAB1 = S1
160 **      *TERMINATE      B0038
161
162 LET S1 = (TIME.V - TIN.S1)*HOURS.V*MINUTES.V
163 LET C.TRAN.CMT.JOB1 = C.TRAN.CMT.JOB1 - 1
164 *PETO=H
165 ***
166
167 ***
168

```

Figure 4-5d. SIMSCRIPT Model for the Flexible Manufacturing System (Cont'd)

4.1.4 Discussion Of The Results Of The FMS Simulation - The standard output report for the resources include the utilization report for the transporter (under facilities) and the workstations (under storages). The IOSTATION does not have any working units. As a result, the average utilization of this station is zero. The resource report also includes the queue statistics for the transporter and workstations. The report shows that the number of machines in the 5th station is not sufficient. As a result, the waiting time for this station is considerably higher than for others. Figure 4-6 shows the output results for the Flexible Manufacturing System.

The standard report also includes the average interarrival time for the Transaction by type. The TAB1, TAB2, and TAB3 tables contain the distribution for the total time spent in the FMS model by each job type. The average for total delays occurred for each job type because of Transporter delay or unavailability of workstation on its path is reported under the variable report.

The total time spent in the model by each job type can be estimated by adding the workstation delays, transporter delay and mean value for all of the service

times in the job's path. For example, the total workstation delay for job1 is approximately 84 units. Its transporter delay is 28 units. The mean service time of workstations that job1 visits are 50, 60, 85 and 50 units which add up to 245 units. Therefore, the estimated average time spent in the model by job1 is 353 units. This shows less than 2% deviation from the simulation run result (360.87 units). Furthermore, the correctness of the SIMSCRIPT model of this Flexible Manufacturing System is confirmed by adding print statements within the SIMSCRIPT code and tracing the state transactions of the model

 *
 ** GRAPHICAL SIMULATION SYSTEM OUTPUT **
 *

FACILITY NAME	CAPACITY	AVERAGE UTILIZATION	NUMBER OF ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENT	MAXIMUM CONTENT
TRANSPORT	1	.481	216	5.356	0	1

STORAGE NAME	CAPACITY	AVERAGE UTILIZATION	NUMBER OF ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENT	MAXIMUM CONTENT
STATION1	3	.484	46	72.534	3	3
STATION2	3	.343	29	85.074	0	3
STATION3	4	.262	44	57.489	1	4
STATION4	4	.299	27	100.971	2	4
STATION5	1	.494	27	43.593	1	1
IOSTATION	1	0.	50	.000	0	1

RESOURCE QUEUE	AVERAGE CONTENT	TOTAL ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENT	MAXIMUM CONTENT
TRANSPORT	.301	216	3.341	0	5
STATION1	.136	46	7.079	0	5
STATION2	.025	29	2.037	0	2
STATION3	.031	44	1.665	0	2
STATION4	.006	27	.520	0	1
STATION5	.031	29	52.131	2	4
IOSTATION	0.	50	.000	0	0

TRANSACTION NAME	NUMBER CREATED	AVERAGE CREATION TIME
JOB1	22	106.562
JOB2	20	112.133
JOB3	8	286.538
CLOCK	1	2400.000

Figure 4-6a. Simulation Results for FMS Model

TABLE	AVERAGE	STD. DEV	NO. ENTRIES
1AE1	363.87	103.36	19
UPPER LIMIT	FREQUENCY	PERCENT OF TOTAL	
200	2	.11	
220	0	0.	
240	1	.05	
260	2	.11	
280	1	.05	
300	1	.05	
320	1	.05	
340	2	.11	
360	0	0.	
380	3	.16	
400	1	.05	
420	1	.05	
440	1	.05	
460	1	.05	
480	0	0.	
500	0	0.	
520	1	.05	
540	0	0.	
560	1	.05	

TABLE	AVERAGE	STD. DEV	NO. ENTRIES
1AE2	277.52	104.54	15
UPPER LIMIT	FREQUENCY	PERCENT OF TOTAL	
100	0	0.	
120	0	0.	
140	2	.13	
160	1	.07	
180	1	.07	
200	1	.07	
220	1	.07	
240	2	.13	
260	2	.13	
280	0	0.	
300	0	0.	
320	2	.13	
340	0	0.	
360	0	0.	
380	1	.07	
400	0	0.	
420	0	0.	
440	1	.07	
460	1	.07	

Figure 4-6b. Simulation Results for FMS Model (Cont'd)

TABLE TABJ	AVERAGE 403.60	STD. DEV 99.98	NO. ENTRIES 7
GPPEA LAMAT	FREQUENCY	PERCENT OF TOTAL	
200	0	0.	
220	0	0.	
240	0	0.	
260	0	0.	
280	0	0.	
300	1	.14	
320	0	0.	
340	1	.14	
360	0	0.	
380	0	0.	
400	0	0.	
420	0	0.	
440	1	.14	
460	1	.14	
480	1	.14	
500	0	0.	
520	1	.14	
540	0	0.	
560	1	.14	

VARIABLE NAME	AVERAGE
JOB1.WORKSTATION.DELAY	84.056
JOB1.TRANSPORT.DELAY	28.141
JOB2.WORKSTATION.DELAY	6.651
JOB2.TRANSPORT.DELAY	23.755
JOB3.WORKSTATION.DELAY	12.035
JOB3.TRANSPORT.DELAY	29.503

ISVAR NAME	VALUE
HXSD1(1,2)	90
HXSD1(1,3)	100
HXSD1(1,4)	180
HXSD1(1,5)	200
HXSD1(1,6)	270
HXSD1(2,1)	90
HXSD1(2,3)	100
HXSD1(2,4)	200
HXSD1(2,5)	180
HXSD1(2,6)	270
HXSD1(3,1)	100
HXSD1(3,2)	100
HXSD1(3,4)	100
HXSD1(3,5)	100
HXSD1(3,6)	160
HXSD1(4,1)	180
HXSD1(4,2)	200
HXSD1(4,3)	100
HXSD1(4,5)	90

Figure 4-6c. Simulation Results for FMS Model (Cont'd)

4.2 Description Of New FMS Blocks And Their Translation

4.2.1 Define Transporter (DEF TRAN) - This block introduces the transporter into the simulation of flexible manufacturing systems and defines the characteristics of the transporter. The transporter specification consists of transporter name, speed, capacity, distance table and initial position for all transporters from the same type.

The transporter speed is given in feet per second and transporter capacity specifies the number of independent units of transporter type in the Flexible Manufacturing System. At any instant of time, the capacity refers to both active and inactive units. The transporter capacity remains constant within the model. However, the number of active units may change over time as transporter units are used and released.

The distance table selects the name of the matrix previously defined in the definition segment of the model which contains the distances between each two stations. Each station is assigned a number which represents its column and row number in the distance matrix. The distance matrix contains the distance in feet if the transporter speed is defined in feet/second otherwise if a metric system is used we can use meter

for distance and meter/second for speed.

The define transporter block appears only in the definition segment. The format of the DEFTRAN block is as follows:

```
DEFTRAN    T, S, C, D, I
```

WHERE

```
T =    TRANSPORTER NAME
S =    TRANSPORTER SPEED
C =    CAPACITY
D =    DISTANCE
I =    INITIAL LOCATION
```

Pass 1 of BCSL Compiler for DEFTRAN block creates a transporter data structure, defines its attributes and adds it to the transporter list. It creates a variable to hold current location of the transporter and selects its initial value and adds the variable to the variable list. Basically, Pass 1 of the translator creates and files the above three data structures where distance table name and transporters speed are saved to be used when a request for a transporter block is being processed.

Pass 2 of the compiler uses the resource list and generates accumulate statements for the average and maximum number of jobs in queue for transporter, also maximum and average utilization for the transporter.

Pass 2 generates code to tally the average time in the queue and the service time. It defines the "current location" variable globally so all the process routines can share it. The current location variable holds the station number of the last station that the transporter visits at any time during the simulation.

The initialize routine uses the starting station number to initialize the current location of the transporter when the simulation starts. In addition, it sets the capacity for the transporter resource. The output generator routine for resources generates SIMSCRIPT code to output statistical results of averages and maximum values for queues and service periods for the transporters.

4.2.2 Define Workstation (DEFWSTAT) - The define workstation block introduces a workstation into the FMS model and defines its characteristics. The workstation specification consists of the workstation name, its corresponding number and capacity. Each workstation consists of several identical machines whose number is defined by the capacity of the workstation. An integer number allocated to each workstation represents the workstation's corresponding column and row number in the distance matrix. The workstation number allocation will start from one and is incremented by one every time. The define workstation block appears in the Definition Segment only. The format of the DEFWSTAT block is as follows:

```
DEFWSTAT      W, N, C
```

```
WHERE
```

```
W = WORKSTATION NAME  
N = WORKSTATION NUMBER  
C = CAPACITY
```

Pass 1 of the BCSL Compiler creates a workstation data structure, defines its attributes (name, number and capacity) and files the element into a workstation list. Later during processing of request for workstation blocks in Pass 1, this structure is used to find the station number given the name or vice versa.

Pass 2 of the compiler, defines workstation as a resource, generates, accumulate and tally statements for corresponding variables in the preamble routines. The initialization routine generates capacity definition statement for the workstation resource using the workstation capacity attribute. The output generation routine generates statements to generate statistical results for the workstation resource.

4.2.3 Request Workstation (REQWSTAT) - The request workstation block is used to simulate a typical workstation in a Flexible Manufacturing System. This block basically represents what happens to a job while it is in the corresponding workstation.

It specifies the workstation name, the number of machines used by the job and the distribution function of service time. Mean service time and two additional parameters are defined for given distribution functions. This block is used in model segments of Flexible Manufacturing models. The format of REQWSTAT is as follows:

```
REQWSTAT    W, N, M, D, P2, P3
```

WHERE

W = WORKSTATION NAME
N = NUMBER OF UNITS NEEDED
M = MEAN SERVICE TIME (FIRST PARAMETER)
D = DISTRIBUTION FUNCTION
P2 = SECOND PARAMETER
P3 = THIRD PARAMETER

The request workstation block in fact is a macro block implemented using three standard BCSL (GPSS) blocks; SEIZE, ADVANCE and RELEASE. Figure 4-7 demonstrates a pictorial equivalence of the REQWSTAT

block.

A workstation is a resource which jobs will seize and release as they move through the model. We don't need to enter and leave the queue in BCSL in order to gather queue statistics for the resources because these statistics are accumulated automatically for each resource in the model.

Pass 1 of the BCSL Compiler searches the workstation list, finds the workstation element with the same name and extracts the workstation number. It sets the SEIZE block parameters using the temporary parameters, calls the SEIZE block generator routine, it then sets the ADVANCE block parameters and calls the ADVANCE block generator routine. Finally, pass 1 sets the RELEASE block parameters and calls the RELEASE block generator. Pass 2 of the translator does not take any special actions for this block.

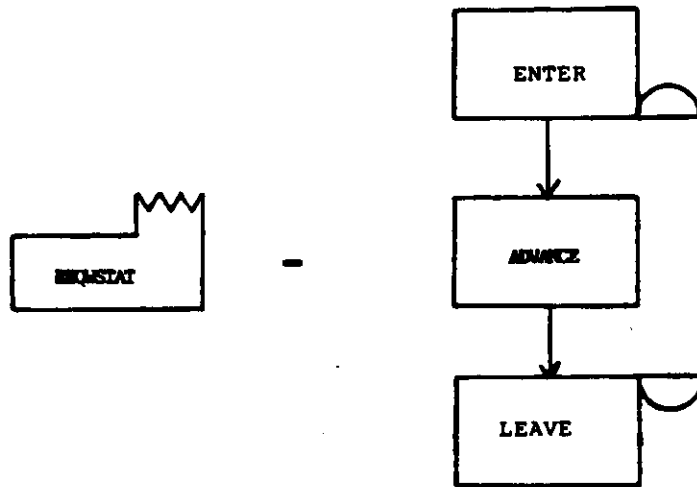


Figure 4-7. Pictorial Equivalence of the REQSTAT Block

4.2.4 Request Transport (REQTRANS) - The request transport block simulates usage of a transporter by a workpiece (job) in the Flexible Manufacturing Models. This block specifies the transporter name and name of destination station. The definition of the next station to be visited by the transporter allows the workstation block to be more general and makes it possible to use this block in any job shop simulation. The format of the REQTRANS block is as follows:

```
REQTRANS    T, S
```

WHERE

```
T = TRANSPORTER NAME  
S = DESTINATION STATION NAME
```

The request for a transport block is a mixture of standard BCSL (GPSS) blocks and SIMSCRIPT code. It uses SEIZE and RELEASE blocks, and utilizes a WAIT statement to simulate the transportation time and transporter delay. The transporter delay represents the time needed by the transporter to move from the station from which it is released to the current station in order to pickup the workpiece (job). Figure 4-8 shows the pictorial equivalence of the REQTRANS block.

Pass 1 of the BCSL Compiler sets the SEIZE block parameters using the temporary parameters, calls the SEIZE block generator to lock transporter resource, finds the distance matrix, and extracts the transporter speed from the transporter list. Pass 1 calculates the distance of the transporter to current stations and allows the job to wait for transporter to arrive to the current station, calculates transporter delay statistics, finds the distance to the destination station and waits for the transporter to deliver the job to the next station. It finally calls the RELEASE block generator in order to release the transporter. The transporter will remain in the last station until a new request is issued for it to move a workpiece.

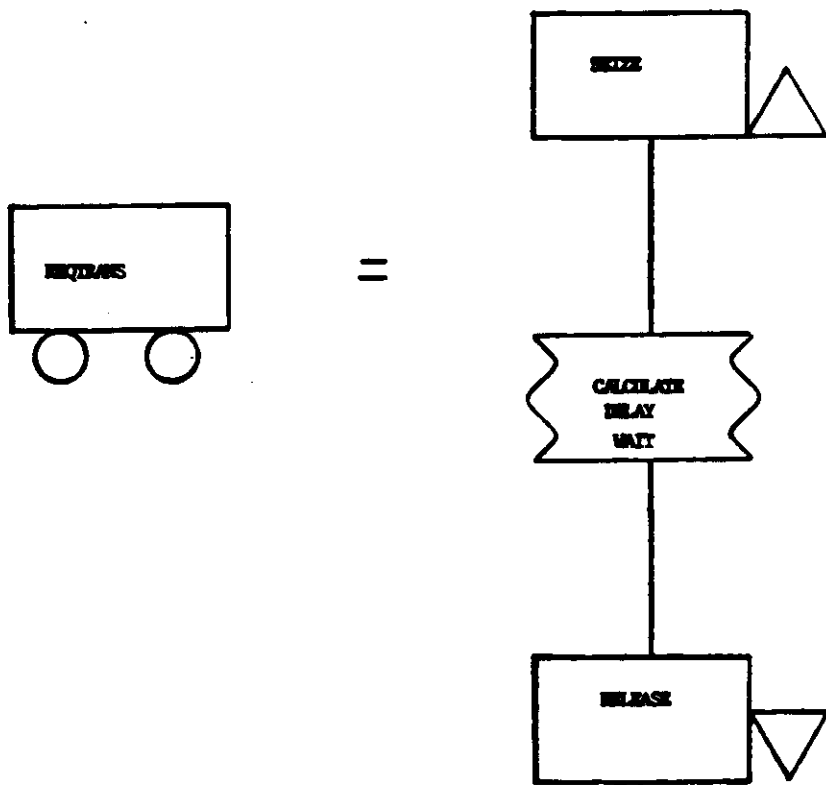


Figure 4-8. Equivalent of REQTRANS Block

4.2.5 Enter Station (ENTERSTAT) - The enter station block simulates the entrance of a job into the Flexible Manufacturing System Model. This block in fact is similar to a "Request a Workstation" block in regards to the fact that the job can be waiting in receiving due to paper work and space limitation delays. The ENTERSTAT block specifies the name of the station through which the job first enters the FMS, the number of units in that station, the distribution function of delays and related parameters. The format of the ENTERSTAT block is as follows:

ENTERSTAT S, N, M, D, P2, P3

WHERE:

S = STATION NAME
N = NUMBER OF UNITS
M = MEAN TIME (FIRST PARAMETER)
D = DISTRIBUTION FUNCTION
P2 = SECOND PARAMETER
P3 = THRID PARAMETER

Usually the amount of delay is zero, or the number of resources are infinite and a fixed delay is imposed on incoming jobs.

The Enter Station block is a composite of SIMSCRIPT code and request workstation block. The SIMSCRIPT section creates and files needed statistical variables in their corresponding sets. Figure 4-9 shows the pictorial equivalence of ENTERSTAT block.

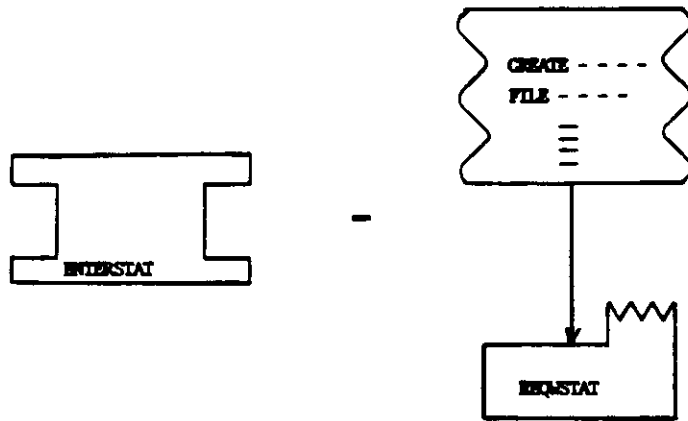


Figure 4-9. Equivalent of ENTERSTAT Block

4.2.6 Leave Station (LEAVESTAT) - The leave station block simulates exiting the Flexible Manufacturing System by a job. This block does not represent exiting of the simulation model. In order to exit the simulation model we still need to use a TERMINATE block. Separation of Leave Station and TERMINATE block provides more flexibility and generality in using FMS blocks, by allowing a FMS model to be added to a global simulation model. Thus the global simulation model includes the Flexible Manufacturing System model. The format of LEAVESTAT is as follows:

```
LEAVESTAT      S
```

WHERE:

S = STATION NAME

The equivalent of the Leave Station Block calculates statistical values, like total workstation delay and transporter delay in the Flexible Manufacturing System.

4.3 User Written SIMSCRIPT

There are two ways that user written SIMSCRIPT code can be added to Block Graphic Symbolic Language (BGSL): through SIMSLINE block or by means of an append file.

4.3.1 SIMSCRIPT Line (SIMSLINE) Block - A SIMSCRIPT Line Block represents a SIMSCRIPT statement to be added to the main line of the generated SIMSCRIPT equivalent of the BGS L model. The format of a SIMSLINE block is as follows:

```
SIMSLINE          SS
```

WHERE:

SS = IS A SIMSCRIPT STATEMENT

Thus the user can add as many SIMSCRIPT statements as desired between BGS L (or BCS L) Commands.

4.3.2 Append File - Using any editor available in the host operating system, the user can generate an append file containing SIMSCRIPT routines. This file is appended to the output of the BCS L Compiler (SIMSCRIPT Translator) and later, in combination with generated SIMSCRIPT, it is compiled to generate the executable machine code. Figure 4-10 shows how the append file is concatenated with pass 1 and pass 2 generated SIMSCRIPT code.

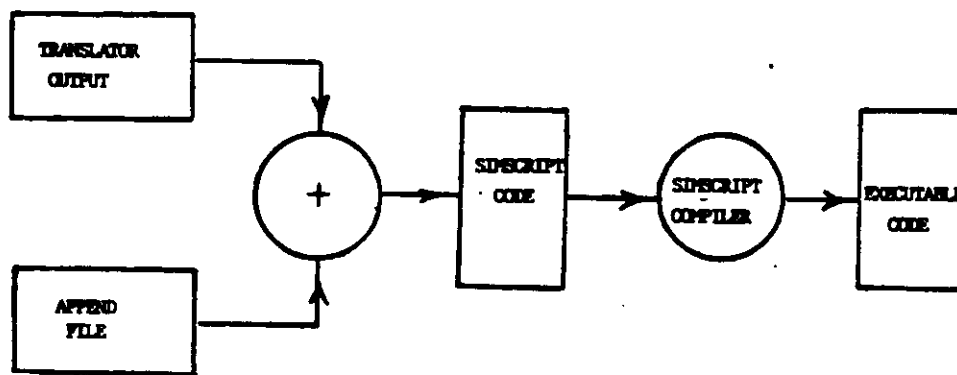


Figure 4-10. Append File Concatenation

In order to call the users written routines in the append file, a SIMSLINE block is used within the BGS model. This block contains a SIMSCRIPT statement which calls the corresponding routine. In this fashion the user can add large numbers of SIMSCRIPT statements into the BGS model by using only one block. Figure 4-11 shows how a SIMSLINE block calls a routine within the append file.

The ability to add SIMSCRIPT statements to BGS provides several debugging tools currently available through SIMSCRIPT, to BGS users. These debugging tools are as follows:

1. Any print statement within blocks to check the variables as the simulation time progresses.
2. The trace back utility in SIMSCRIPT.
3. User supplied debugging routines in SIMSCRIPT like SNAP.R
4. BETWEEN.V tracing routing, which traces the flow of SIMSCRIPT execution.

Notice that in order to use these debugging tools, the user needs to know the SIMSCRIPT programming language.

Custom made output routines can be added to the simulation model by writing special purpose output generator routines which are called from within BGSF blocks. This feature provides the user with flexible output generation in addition to standard BGSF outputs.

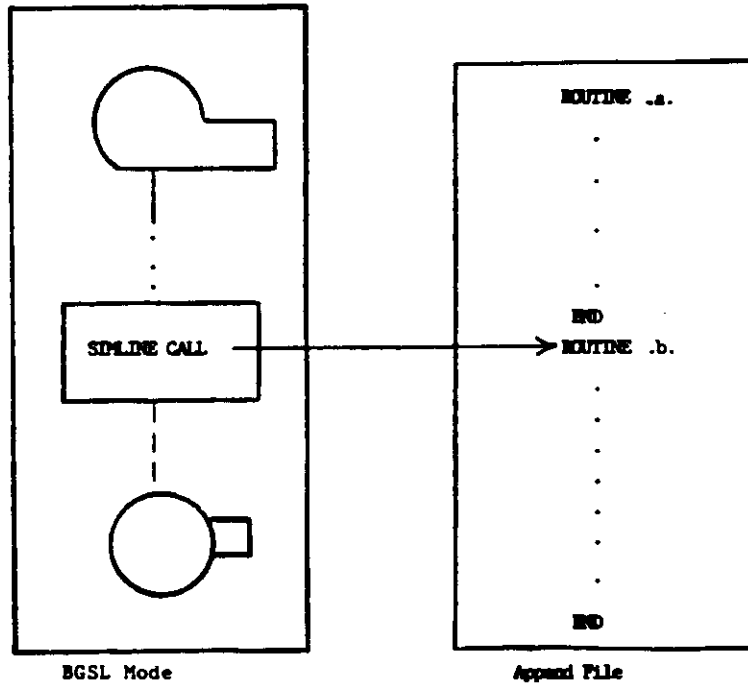


Figure 4-11. Usage of SIMSLINE Block

4.4 Generation Of Customized Simulation Systems

4.4.1 Generation Of New Blocks In BCSL - Generation of new blocks in Block Command Symbolic language requires the addition of a block processor routine into the BCSL compiler (SIMSCRIPT Translator). This routine will be used during Pass 1 processing of the new block. In addition a call to the "new block processor routine" must be added to the "operation processor routine". Figure 4-12 demonstrates the calling sequence in Pass 1 of the compiler regarding the addition of the new block.

Very often Pass 1 of the BCSL compiler satisfies all the requirements for processing of a new block and there is no need to add a new data structure to the compiler. In these cases, the above two steps will be sufficient to permanently add a new block to BCSL. But if there is a new data structure to be added or if additional action needs to be taken in the Preamble, Initialization or output routines, then the following steps should be added to the compiler:

1. Definition of new data structures in the preamble section of the compiler. Add a set which contains the new data structure elements and is owned by the system.

2. Within Pass 1 "new block processor" routine create a new element (Temporary entity) corresponding to the new data structure, define its attributes and eventually file the new element into the new set.

3. Through Pass 2 routines, preamble, initialization or output generator, for each entry in the new set, add the needed SIMSCRIPT code to generate the corresponding sections of equivalent SIMSCRIPT code.

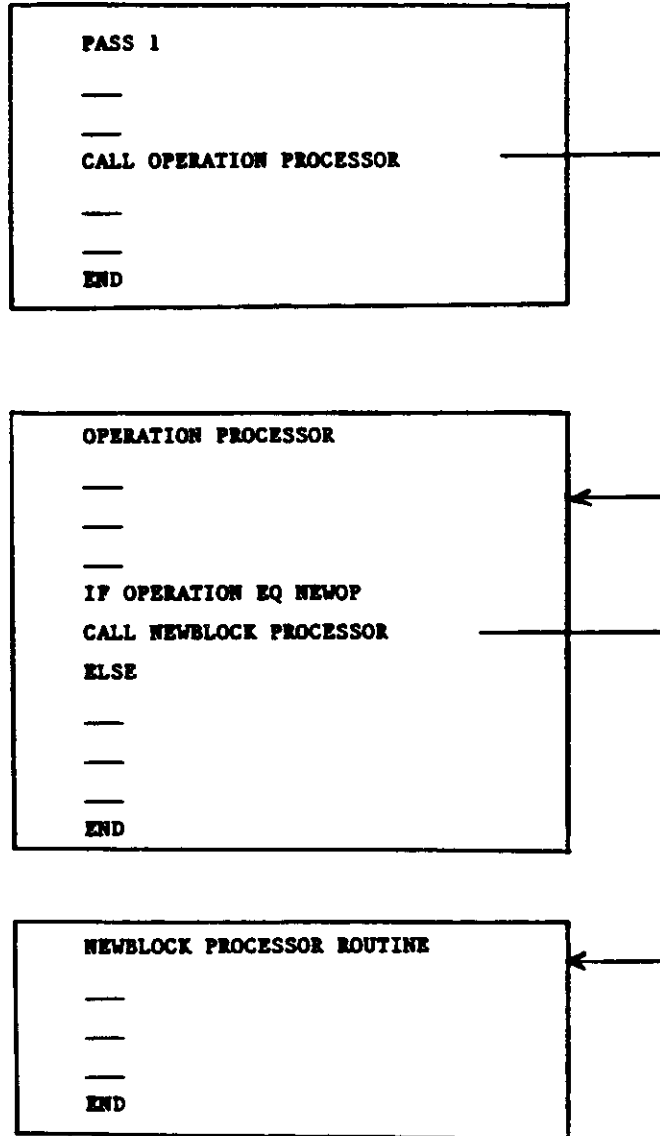


Figure 4-12. Calling Sequence for a New Block

4.4.2 WILDCARD And Expansion Of BGSL - In the last section we described how to create new blocks and expand the Block Command Symbolic language. It is also possible to add new blocks to the Block Graphic Symbolic language. This requires modifying the user interface in order to add the new block's pictorial representation (ICON) into the graphic data base and to add the new block's name into the block selection menu.

In order to isolate the user interface from possible modifications as new blocks are added to the Block Command Symbolic Language, the WILDCARD concept is developed. The WILDCARD block is added to the block selection menu and can have up to ten attributes. The format of a WILDCARD block can be represented as follows:

WILDCARD N, A1, A2,.....A9

WHERE:

N = NEW BLOCK'S NAME
A1 = FIRST ATTRIBUTE OF NEW BLOCK
.
.
A9 = THE NINTH ATTRIBUTE OF NEW BLOCK

The WILDCARD block can be processed either by the BCSL generator or the BCSL compiler. The BCSL generator accepts the WILDCARD and generates a new block with the name of the new block using its corresponding attributes to replace the WILDCARD block in the generated BSCL code. In addition, the BCSL compiler is also capable of receiving WILDCARD blocks. Pass 1 of the compiler will generate a new command using the new block's name (first attribute of the WILDCARD block) and will call Pass 1 of the compiler recursively in order to process the new block. Figure 4-13 shows how a WILDCARD block is transformed into the corresponding new block.

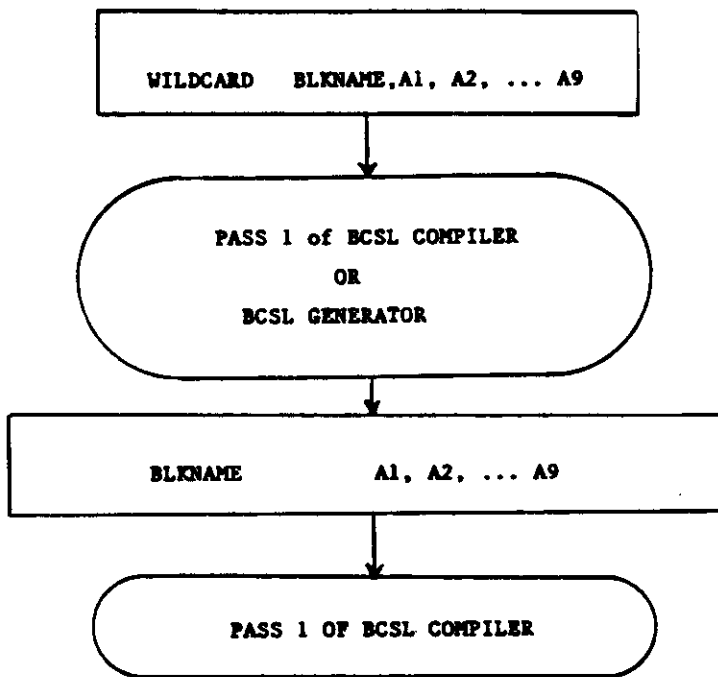


Figure 4-13. WILDCARD Block Translation

4.4.3 MACRO Blocks - In previous sections we described how to create new blocks in Block Graphic Symbolic language. Very often the new block is created using existing blocks. By definition a MACRO block is a new block created using restricted existing blocks in BCSL. For example, block "M" may consist of six existing blocks B1, B2, B3, B4, B5 and B6

where:

B1 has 3 attributes

B2 is a TEST block and has 3 attributes

B3 has 2 attributes

B4 has 2 attributes

B5 is a TRANSFER block with 1 attribute

B6 has 1 attribute

Figure 4-14 demonstrates the Graphical representation of an "M" block and its equivalent blocks.

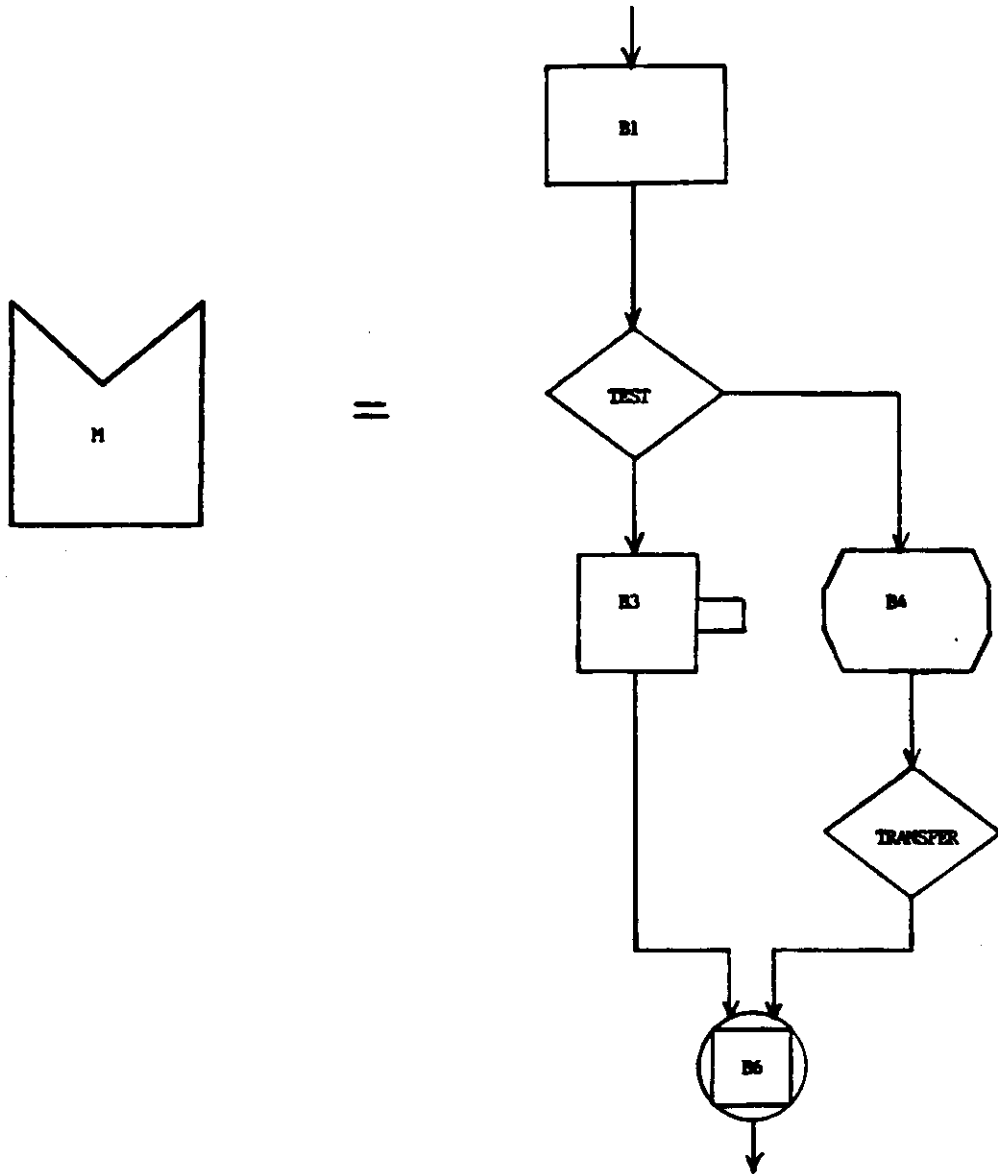


Figure 4-14. Equivalent of the "M" Block

The "M" Block can be represented in BCSL as follows:

	B1	P1, P2, P3
	TEST	P2, P4, L1
	B3	P5, P6
M =	TRANSFER	L2
	L1 B4	P2, P7
	L2 B5	(P8 + P7)/2

For this example we need 8 attributes for the "M" block in order to be able to supply the needed information to all of the construct blocks. The format of an "M" block will be as follows:

M P1, P2,.....P8

WHERE:

P1 through P8 are the attributes

Attribute P2 is shared by several internal blocks and labels L1 and L2 are used internally; therefore there is no need to supply them through MACRO block attributes.

Note that one of the limitations of a MACRO block is the maximum number of attributes allowed in a block (in BCSL there is a maximum of 10 attributes). This limits the attributes available for the internal blocks and finally limits the number of blocks that a MACRO block can hold.

4.5 AN ALTERNATIVE SOLUTION TO THE PRODUCTION SHOP MODEL

This section presents an improved solution for the manufacturing job shop model discussed in Section 3.2 of this dissertation. The new model is built primarily using the new blocks developed for simulation of the Flexible Manufacturing System. The define workstation (DEFWSTAT) and the request workstation (REQWSTAT) blocks are used in conjunction with the standard BCSL blocks in development of the alternative model. As a result, the number of blocks used to simulate the same job shop has been reduced to about 1/3 of the original model.

Figure 4-15 shows a BCSL model segment which represents JOB1 sequence in the model. As illustrated the REQWSTAT blocks have replaced the SEIZE-ADVANCE-RELEASE block chains (there is no transporter involved in this model). Figure 4-16 shows the BCSL equivalent for the model. The number of command lines is reduced by a factor of 3 for each model segment. Figure 4-17 shows the SIMSCRIPT equivalent transaction process representing the JOB1 segment. A comparison of this transaction process and JOB1 transaction process in Figure 3.7e demonstrates that the SIMSCRIPT equivalent of both sequences are equal.

Finally, Figure 4-18 contains the simulation run results for the new alternative model. As it is obvious, the results are the same for both models.

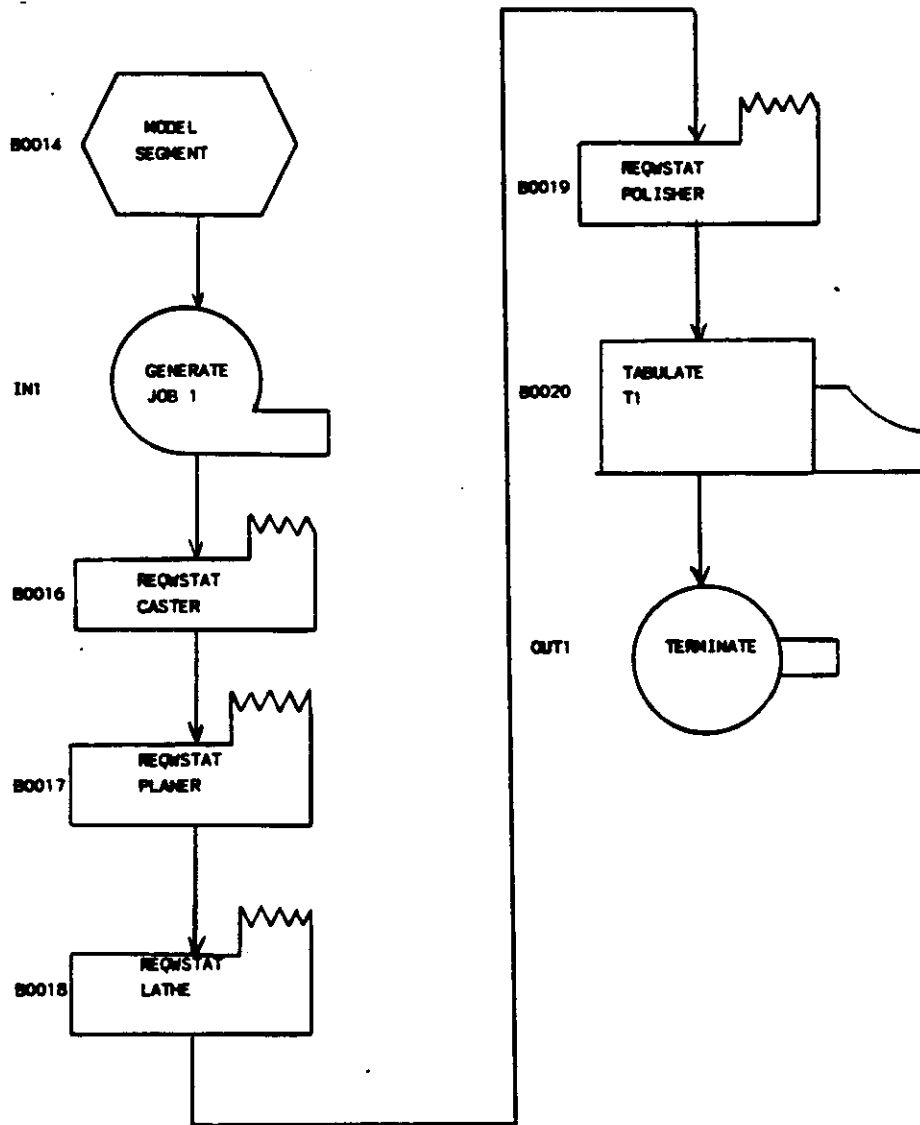


Figure 4-15. BGSL Model Representing JOB1 Transaction


```

      REQSTAT  DRILL,1,900.,DSSEXPONENTIAL.F          B0029
      REQSTAT  LATHE,1,650.,DSSEXPONENTIAL.F          B0030
      TABULATE T2                                       B0031
OUT2  TERMINATE                                       OUT2
  
```

** NEW SEGMENT

```

      GENERATE 288.,DSSEXPONENTIAL.F,,,,,JOB3,JOB3    B0034
IN3  REQSTAT  CASTER,1,2350.,DSSEXPONENTIAL.F        IN3
      REQSTAT  SHAPER,1,2500.,DSSEXPONENTIAL.F        B0036
      REQSTAT  DRILL,1,500.,DSSEXPONENTIAL.F          B0037
      REQSTAT  PLANE,1,300.,DSSEXPONENTIAL.F          B0038
      REQSTAT  POLISHER,1,250.,DSSEXPONENTIAL.F       B0039
      TABULATE T3                                       B0040
OUT3  TERMINATE                                       OUT3
  
```

** CONTROL SEGMENT

```

      START    5                                       B0043
      END
  
```

Figure 4-16b. Improved BCSL Model for the Production Shop

```

PTIONS  TERM,LOAD,ID,TRACE2,NOTERN,CHK,REN=REN

1  PROCESS TRAN.JOB1
2  DEFINE P_ARRAY AS INTEGER,1-DIMENSIONAL ARRAY
3  RESERVE P_ARRAY AS 10
4  DEFINE TIN.#1 AS A REAL VARIABLE
5  TIME.#1 = TIME.V
6  '' IN1  REQSTAT  CASTER,1,1250.,DSSEXPOENTIAL.F
7
8  'IN1 '
9  LET NSIN1  = NSIN1  + 1
10 LET NSIN1  = NSIN1  + 1
11 LET CURRENT_WS_NUM = 1
12 LET TIN.Q.CASTER = TIME.V
13 LET NUM.ENT.Q.CASTER = NUM.ENT.Q.CASTER + 1
14 REQUEST 1 CASTER
15 LET TIMEIN.Q.CASTER=(TIME.V - TIN.Q.CASTER)*HOURS.V*MINUTES.V
16 LET TIN.F.CASTER = TIME.V
17 ADD TIMEIN.Q.CASTER TO TOTAL_WS_DELAY
18 WORK EXPONENTIAL.F(1250.,SEED.TRAN.JOB1) UNITS
19 RELINQUISH 1 CASTER
20 LET TIMEIN.F.CASTER=(TIME.V - TIN.F.CASTER)*HOURS.V*MINUTES.V
21 ''      REQSTAT  PLANNER,1,350.,DSSEXPOENTIAL.F
22
23 LET NSIN1  = NSIN1  - 1
24 LET CURRENT_WS_NUM = 3
25 LET TIN.Q.PLANNER = TIME.V
26 LET NUM.ENT.Q.PLANNER = NUM.ENT.Q.PLANNER + 1
27 REQUEST 1 PLANNER
28 LET TIMEIN.Q.PLANNER=(TIME.V - TIN.Q.PLANNER)*HOURS.V*MINUTES.V
29 LET TIN.F.PLANNER = TIME.V
30 ADD TIMEIN.Q.PLANNER TO TOTAL_WS_DELAY
31 WORK EXPONENTIAL.F(350.,SEED.TRAN.JOB1) UNITS
32 RELINQUISH 1 PLANNER
33 LET TIMEIN.F.PLANNER=(TIME.V - TIN.F.PLANNER)*HOURS.V*MINUTES.V
34 ''      REQSTAT  LATHE,1,200.,DSSEXPOENTIAL.F
35
36 LET CURRENT_WS_NUM = 2
37 LET TIN.Q.LATHE = TIME.V
38 LET NUM.ENT.Q.LATHE = NUM.ENT.Q.LATHE + 1
39 REQUEST 1 LATHE
40 LET TIMEIN.Q.LATHE=(TIME.V - TIN.Q.LATHE)*HOURS.V*MINUTES.V
41 LET TIN.F.LATHE = TIME.V
42 ADD TIMEIN.Q.LATHE TO TOTAL_WS_DELAY
43 WORK EXPONENTIAL.F(200.,SEED.TRAN.JOB1) UNITS
44 RELINQUISH 1 LATHE
45 LET TIMEIN.F.LATHE=(TIME.V - TIN.F.LATHE)*HOURS.V*MINUTES.V
46 ''      REQSTAT  POLISHER,1,600.,DSSEXPOENTIAL.F
47
48 LET CURRENT_WS_NUM = 6
49 LET TIN.Q.POLISHER = TIME.V
50 LET NUM.ENT.Q.POLISHER = NUM.ENT.Q.POLISHER + 1
51 REQUEST 1 POLISHER
52 LET TIMEIN.Q.POLISHER=(TIME.V - TIN.Q.POLISHER)*HOURS.V*MINUTES.V
53 LET TIN.F.POLISHER = TIME.V
54 ADD TIMEIN.Q.POLISHER TO TOTAL_WS_DELAY
55 WORK EXPONENTIAL.F(600.,SEED.TRAN.JOB1) UNITS
56 RELINQUISH 1 POLISHER

```

Figure 4-17a. SIMSCRIPT Equivalent for JOB1 Transaction

```
57 LET TIMEIN.P.POLISHER=(TIME.V - TIN.P.POLISHER)*HOURS.V*MINUTES.V  
58 '' TABULATE T1 B  
59  
60 LET M1=(TIME.V - TIN.M1)*HOURS.V*MINUTES.V  
61 LET T1 = M1 C  
62 '' OUT1 TERMINATE  
63  
64 'OUT1 '  
65 LET #SOUT1 = #SOUT1 + 1  
66 LET #SOUT1 = #SOUT1 + 1  
67 LET M1=(TIME.V - TIN.M1)*HOURS.V*MINUTES.V  
68 LET C.TRAM.CNT.JOB1 = C.TRAM.CNT.JOB1 - 1  
69 RETURN  
70 ***  
71  
72 ***  
73  
74 ***** NEW SEGMENT  
75  
76 ***  
77  
78 ***  
79  
80 '' GENERATE 4800.....TIMER,TIMER E  
81  
82 END
```

Figure 4-17b. SIMSCRIPT Equivalent for JOB1 Transaction

FACILITY NAME	CAPACITY	AVERAGE UTILIZATION	NUMBER OF ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENT
CASTER	1	1.000	19	1309.337	1
PLANE	1	.141	9	374.932	0
LATHE	1	.207	16	325.009	1
POLISHER	1	.280	8	854.558	1
SHAPER	1	.987	23	1039.425	1
DRILL	1	.598	10	1583.916	1

STORAGE NAME	CAPACITY	AVERAGE UTILIZATION	NUMBER OF ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENT
--------------	----------	---------------------	-------------------	-------------------	-----------------

RESOURCE QUEUE	AVERAGE CONTENT	TOTAL ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENT
CASTER	70.229	151	12272.845	132
PLANE	.083	9	221.812	0
LATHE	.147	16	220.108	0
POLISHER	.056	9	138.998	1
SHAPER	55.942	126	11638.306	103
DRILL	3.817	22	3149.440	12

QUEUE NAME	MAXIMUM	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	AVERAGE TIME/TRANS
------------	---------	------------------	---------------	--------------	--------------------

TRANSACTION NAME	NUMBER CREATED	AVERAGE CREATION TIME
JOB1	66	357.562
JOB2	5	8799.999
JOB3	116	203.125
JOB4	85	278.120

TABLE	AVERAGE	STD. DEV	NO. ENTRIES
T1	12147.12	7746.44	6

UPPER LIMIT	FREQUENCY	PERCENT OF TOTAL
-------------	-----------	------------------

Figure 4-18a. Simulation Results for the Improved Production Shop Model

1200	0	0.
2400	0	0.
3600	2	.33
4800	1	.17
6000	0	0.
7200	0	0.
8400	0	0.
9600	0	0.
10800	3	.50

TABLE	AVERAGE	STD.DEV	NO.ENTRIES
T2	10782.67	6349.47	7

UPPER LIMIT	FREQUENCY	PERCENT OF TOTAL
1200	0	0.
2400	1	.14
3600	2	.29
4800	0	0.
6000	0	0.
7200	0	0.
8400	0	0.
9600	0	0.
10800	4	.57

TABLE	AVERAGE	STD.DEV	NO.ENTRIES
T3	2990.53	0.	1

UPPER LIMIT	FREQUENCY	PERCENT OF TOTAL
1200	0	0.
2400	1	1.00
3600	0	0.
4800	0	0.
6000	0	0.
7200	0	0.
8400	0	0.
9600	0	0.
10800	0	0.

TABLE	AVERAGE	STD.DEV	NO.ENTRIES
TJOBS	158.20	70.52	5

UPPER LIMIT	FREQUENCY	PERCENT OF TOTAL
10	0	0.
20	0	0.
30	0	0.
40	0	0.
50	5	1.00

Figure 4-18b. Simulation Results for the Improved Production Shop Model

5 CONCLUSIONS AND RECOMMENDATIONS

5.1 SUMMARY

A Graphical Simulation System has been developed which allows graphic and interactive construction of a model, rather than statement inputs. This proves that purely graphical languages based on menu driven interactive interfaces are practical. Further, this system has improved the conventional programming concepts, in such a way that the user interactively develops a model through visual images rather than through logically connected procedural statements or commands.

Developing a model using the Graphical Simulation System does not require conventional programming knowledge. There are no syntax barriers, no command language statements to memorize and no procedural programming conventions to follow. The interactive menu driven user interface provides an extensive help facility which supports the user in every step of model development, telling what each option on the menu means. In addition, there are always guidelines displayed on the CRT, telling the user what to do next and how to get to the next step. As a result; all the user needs to know is: (1) what it is that he wants to model, (2)

general concept of process oriented simulation, and (3) the function of each block that is needed to build the model. Furthermore, the user interface displays the meaning of all the parameters needed for each block and will check the validity and correctness of user input.

According to our experience, using the Graphical simulation System and building models are rather trivial to learn. Development of Flexible Manufacturing System models does not need any programming background and requires very short training period; Which proves that the GSS is well suited for doing simulation by non-programmers. However, the user needs to be familiar with process oriented world view of simulation (in specific GPSS programming mentality) in order to develop general purpose simulation models.

The Graphical Simulation System provides general purpose simulation capabilities in addition to special purpose custom made simulation blocks. One of the original goals of this research has been achieved by successfully developing and testing a graphic manufacturing production shop model. A model has been built for the production shop, described by Schriber [41]; correctness of this model and related simulation results have been discussed. In addition, an equivalent

model for the same production shop has been developed using special purpose FMS blocks, which has simplified the model and reduced its size by 1/3.

The Graphical Simulation System is built on top of SIMSCRIPT which is a simulation language with the power of a general purpose programming language. SIMSCRIPT has text processing, list processing capabilities and extensive report generation qualities. Users can write SIMSCRIPT routines and include them in the generated SIMSCRIPT equivalent of the model to take advantage of SIMSCRIPT capabilities like flexible output generation and tracing.

The major achievement of this research has been the extendability of the Block Command Symbolic Language. Specifically, development of MACRO blocks each of which represents several graphically connected blocks. These features made it possible to use the GSS as a tool in the development of the Flexible Manufacturing System Simulation Sub-language. As a result a new concept in computer science has been experienced namely that software tools for development of very high level languages (dedicated, natural or graphic languages) maybe a common practice in the future. These tools are capable of generating systems that are expandable and

can adapt to the users needs as technology changes and new requirements are established

The modularity of the Graphical Simulation System components makes it possible to further expand the system and provide animation of simulation runs. The animation would be based on original graphic models which makes it superior to existing animation packages. The integrated display makes it more understandable and allows more complicated model animation. Also by adding expert system components or knowledge based components to the Graphical Simulation System , we can develop expert simulation system.

The recent surge of interest in the development of truly graphical simulation languages, demonstrates the importance of these systems and confirms our belief that graphical interfaces are the way of the future. Even though several industrial attempts have been in progress since this research activity started, a general purpose interactive graphical simulation system is not yet on the market; which makes GSS a pioneer in this field.

5.2 Performance

In order to evaluate the performance of the Graphical Simulation System, we will consider three major areas; model development, compilation of the model and execution run. The first and most important area is the user time required to build a model. Using the process-oriented world view and iconic graphic-oriented languages has greatly improved the time required for every phase of model development especially the original model formulation steps. The actual time spent to enter GSS models into the computer is less than the time needed for the same model in SIMSCRIPT (on the average it improves the time by a factor of 1/3) and is the same as the time needed for GPSS or SIMAN models.

Compilation of a BGS� model requires more CPU time than compilation of equivalent models in GPSS, SIMSCRIPT or SIMAN. The main reason for this is that a BGS� graphic model is first translated to BCSL and then translated to an equivalent SIMSCRIPT model. The equivalent SIMSCRIPT model on the average contains more statements in it than a manually written SIMSCRIPT model. As a result it takes longer to compile the generated equivalent SIMSCRIPT model. The SIMSCRIPT compiler takes about 90% of the total BGS� compilation

time.

Finally, execution of generated models are as fast as any other language including GPSS, SIMSCRIPT or SIMAN.

5.3 Future Research Possibilities

The ability to add new blocks to the proposed language has facilitated the development of new special purpose simulation systems as proved by the development of the Flexible Manufacturing System Simulation Block-Commands. By defining a new set of blocks, we can simulate different activities in the target model. Expansion of this system and development of special purpose simulation systems (i.e., "Robot" simulation) can be a follow up research project. Improving the front end graphical interface to be able to easily add new icons for new blocks can be a part of this research activity

Adding a graphical output generator to the current system will allow the animation of simulation runs by simulating the movements of entities into the system and displaying them on the screen. The user can find out about the build up of the bottlenecks in the modeled system by observing the animated flow of material through the plant (or in a more general case, the movement of objects through the queue and service components of the system).

Animation can be done by generation of animation output routines for each block in the model. The animation output routine modifies the screen whenever the corresponding block is executed. This way, as the simulation time progresses, new objects are introduced by the GENERATE block. These objects will be shown entering the model and moving from block to block as they wait in the queue or service blocks. Finally they will disappear into the TERMINATE block. The originally drawn graphical model can be used as the static background display for the animated objects. The animation output routine generator can be a separate module written in PL/1 which generates the graphic output commands for each block based on the type, parameters and location of that block. Figure 5-1 shows how the animation output routine generator can be added to the compiler software. The animation executable code will finally be linked to the simulation executable code. In this case, the SIMSCRIPT equivalent of each block contains a call to the corresponding animation output routine. Therefore every time a process routine containing the block equivalent code is activated, the animation output routines are called in the same sequence as the equivalent code is executed.

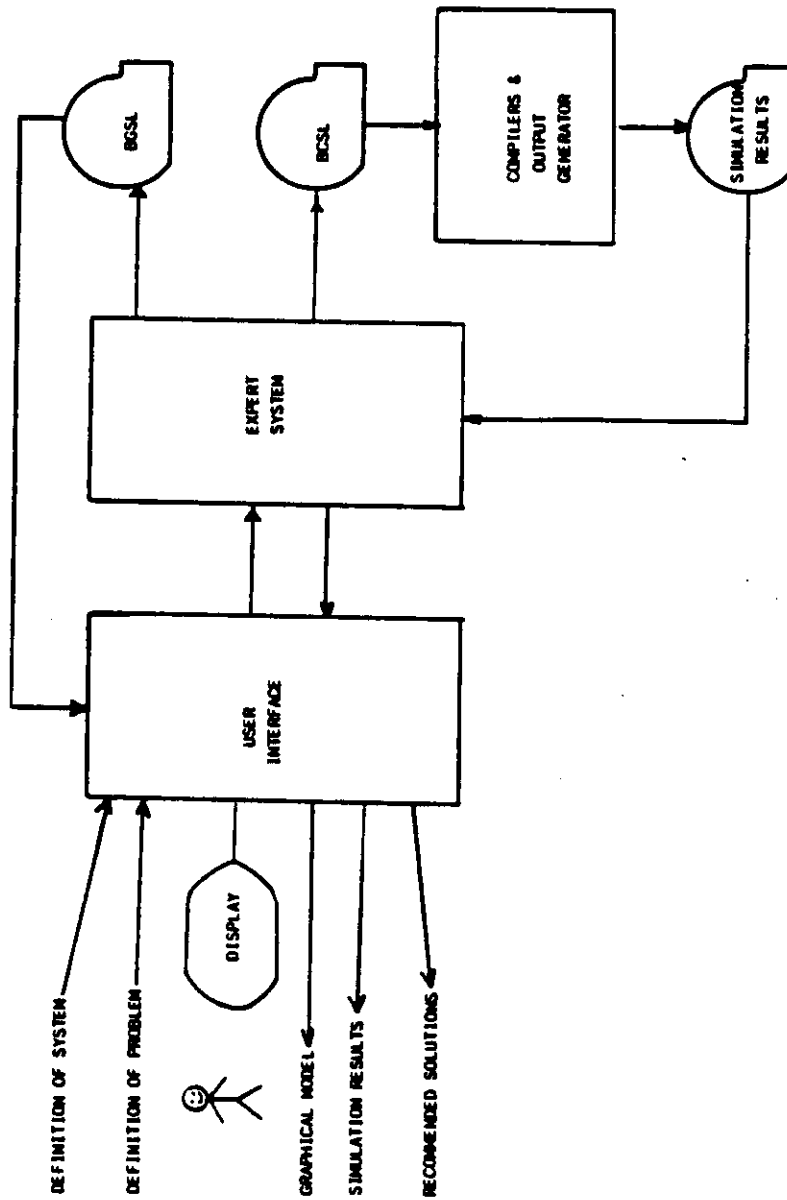


Figure 5-1. Animation Of Simulation Run

Figure 5-2 shows how the animation output routine is called via the equivalent SIMSCRIPT process routines. The animated objects could be symbolized by little balls which move from block to block and can be accumulated either in a block or outside a block.

Another candidate for future research is the development of an expert simulation system based on existing GSS system. In the proposed system, the user describes the facts about the system, its environment and the problem which he is trying to solve. Later the expert simulation system generates the simulation model, runs the model, collects the results, analyzes the results and recommends the potential solution.

The main goal of this research would be to isolate users from the expertise and knowledge needed to build a simulation model and further improve the user interface.

The user of this system basically enters all he knows about the problem through the user interface and the system builds the model for him. The expert system generates a model in the Block Command Symbolic language or the Block Graphic Symbolic language for the simulation using the rules and knowledge base. Later the Block-oriented model is translated into SIMSCRIPT for final compilation and run.

This system also displays the graphical model so that the user can visually verify if the model is valid. Figure 5-3 shows the overview of the proposed expert simulation system.

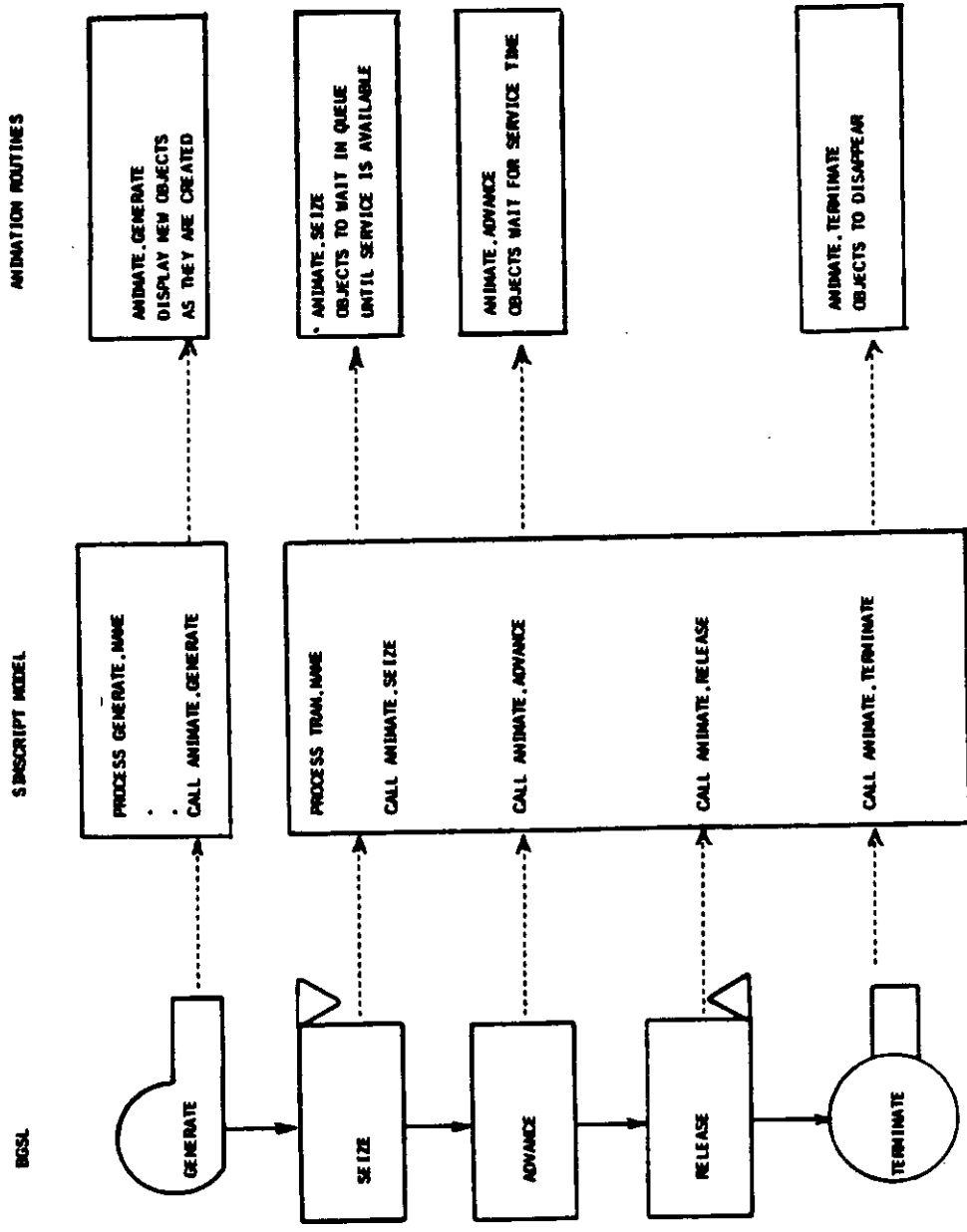


Figure 5-2. Animation Routines and Block Commands

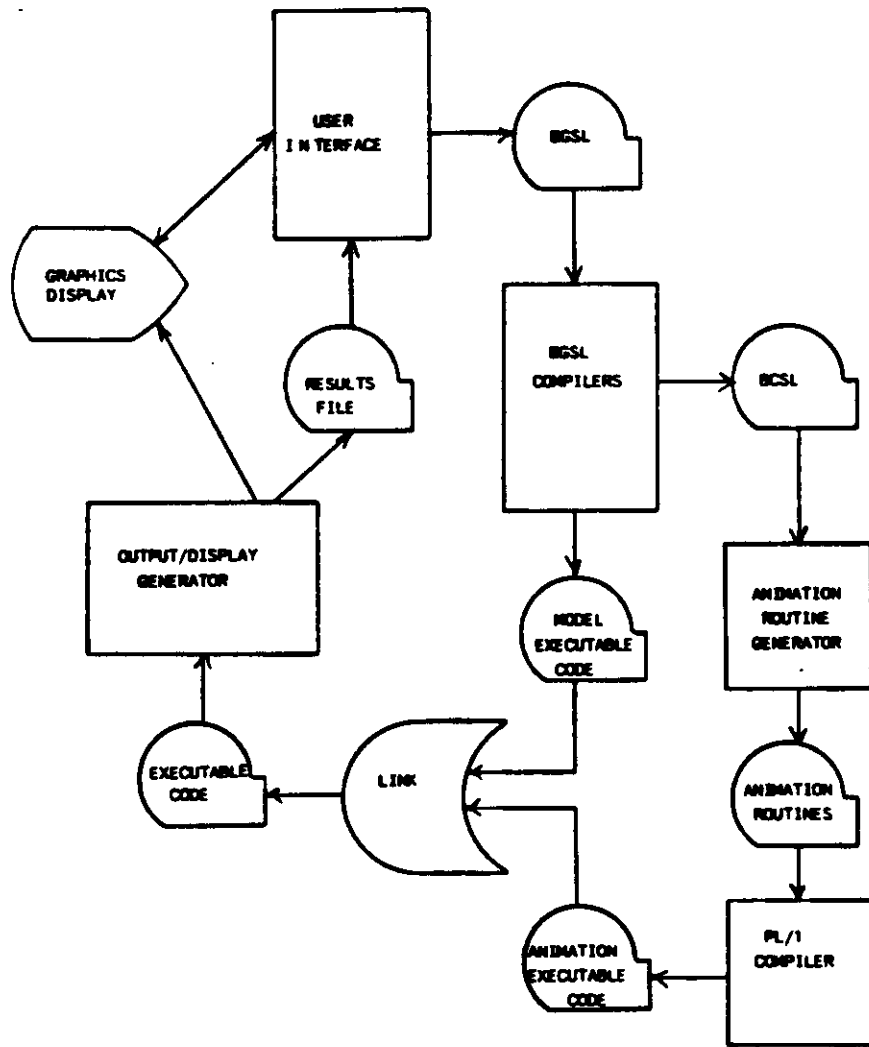


Figure 5-3. An Expert Simulation System

5.4 Limitations And Potential Improvements

The original objective of this research has been to develop a prototype for graphic simulation which is upward compatible with GPSS and capable of modeling manufacturing production shops. About 20 blocks of GPSS were selected to constitute the backbone of the BCSL. By the end of this research the number of implemented blocks has grown to 40 blocks (including almost all of the elementary and intermediate GPSS blocks) and the simulation capabilities of the GSS has been extended far beyond manufacturing modeling.

The generated Graphical Simulation System is not intended for commercial use and certainly does not cover all the aspects and utilities developed for such systems which require large manpower and several years of development.

Since June of 1985, where the GSS has been fully operational, several areas have been improved. Several features have been added to the user interface to facilitate and speed up the model building process, in addition the BCSL compiler has been optimized to reduce CPU time requirements. Still, some improvements could be made in both areas. The user interface can be improved to provide an easy tool for the user to define

new iconic shapes in BGS L for every new block in BCS L.

One of the inherent problems with a system such as GSS is that the user needs to be familiar with SIMSCRIPT in order to be able to develop and debug complicated models. In fact, the more complicated the model is, the more the user needs to know about the internals of the simulation system. In this regard, more debugging tools and messages can be added to the system to facilitate error detection and traceback.

A few features of GPSS V could not be completely implemented by the SIMSCRIPT equivalent models, primarily because the underlying differences between the SIMSCRIPT and GPSS internal mechanisms. Following is a list of missing or mismatched features of GPSS V;

1. Deterministic functions are not implemented (this requires an extensive table-lookup mechanism).
2. Variables used for distribution functions and supplied functions must be real (integer values will look like zeros to SIMSCRIPT in certain places).
3. A variable can not be used more than once in each segment by the GATE or TEST blocks in order to suspend that segment (this can be improved by a more complicated left-monitored routine for the GATE and

TEST variable).

4. The GENERATE block priorities can be broken through by monitor routines belonging to lower priority segments.

The following is a list of limitations for the advance features of BCSL.

1. The number of blocks allowed in a MACRO block is limited because the maximum number of parameters for the MACRO block is limited to 10 (this limits the number of attributes available for the internal block).
2. Advanced GPSS blocks are not implemented.
3. TRANSFER BOTH option is not implemented

BIBLIOGRAPHY

1. Abed, Seraj Yousef, "A Comparative Study of Three Simulation Languages as Applied to Manufacturing Facility Simulation", PhD Dissertation, Iowa State University, 1982.
2. Alan, A., Pritsker, B., "Introduction of Simulation and SLAM", 2nd Edition, 1984.
3. Banks, J. and Carson, J.S., "Discrete-Event System Simulation", Prentice-Hall, Inc., Englewood Cliffs, N.J., 1984.
4. Banks, Jerry and Carson, John S., "Process-Interaction Simulation Languages", Simulation, May, 1985.
5. Berry, Robert, et al, "PAWS 2.0 Performance Analyst's Workbench System User's Manual", December 1982.
6. Bobiller, P.A., "Simulation with GPSS and GPSS V", 1975.
7. Bratley, P., et al, "A Guide to Simulation", Springer-Verlay, New York, N.Y., 1983.
8. Braum, J.E., et al, "SIMSCRIPT II.5 Reference Handbook", CACI, 1983.
9. Cheny, T.C.E., "Simulation of Flexible Manufacturing Systems", Simulation, Dec. 1985.
10. Chin, Shim-Miao, et al, "A Graphical Approach to Simulation", Computer Graphic World, September 1981.
11. Christy, D.P. and Watson, H.J., "The Application of Simulation: A Survey of Industry Practices", Interfaces, 13.5, 1983.
12. Duersh, Ralph R., and Laymon, Marc A., "Programming-Free Graphic Factory Simulation with GEFMS/PC (Graphically Enhanced Flexible Modeling System)", Proceedings for the 1985 Winter Simulation

Conference.

13. El Maraghy, H.A., et al, "Simulation and Graphical Animation of Advanced Manufacturing Systems", Journal of Manufacturing Systems, Volume 1, Number 1, 1983.
14. Emshoff, J.R., and Sisson, R.L., "Design and Use of Computer Simulation Models", 1970.
15. Gilman, Andrew, "Interactive Control of Models; A Natural Companion to Animated Simulation Graphics", Proceedings of 1985 Winger Simulation Conference.
16. Goldern, Donald G., "Software Engineering Considerations for the Design of Simulation Languages", Simulation, October 1985.
17. Henriken, J.D., and Crain, "GPSS/H User's Manual", Wolverino Software Corp., 1983.
18. Hills, P.R., "An Introduction to Simulation using SIMULA", Publication No.S55, Norwegian Computing Center, Oslo, 1973.
19. Holly, Michael Alvern, "An On-Line Graphical System for Digital Design", PhD Dissertation, UCLA, 1968.
20. Hutchinson, George K., "The Desing of an Automated Material Handling System for a Job Shop", Computer Industry, June 1983.
21. Kerchoffs, E.J.H., et al, "The Impact of Advanced Information Processing on Simulation", Simulation, Jan. 1986.
22. Kiviat, P.J., "Development of Discrete Digital Simulation Languages", Simulation, Vol. VIII, No. 2, 1967.
23. Kiviat, P.J., et. al, " The SIMSCRIPT II programming language", Printice Hall, 1969.
24. Law, Averill M., et. al, "On-Line Demonstration of Simulation Software", Winter Simulation Conference, 1982.
25. Law, Averill M., and Larmey C.S., "An Intorudction to Simulation Using SIMSCRIPT II.5", CACI, 1984

26. Lodding, Kenneth N., "Iconic Interfacing", IEEE Computer Graphics and Applications, April 1983.
27. Markowitz, H., "SIMSCRIPT, Encyclopedia of Computer Science and Technology", J. Belzer, et al, Inc, 1978
28. Mesrobian, Edmond, "An User Interface for a Graphical Simulation System", Unpublished Masters Thesis, UCLA Computer Science Department, 1986.
29. Miner, R.J., et al, "Decision Support for Manufacturing", Proceedings, 1981 Winter Simulation Conference, IEEE, Dec. 1981.
30. Miner, Robert J., et al, "Map/1 User's Manual", April 1983.
31. Mortenson, R.E., "Maintenance Planning and Scheduling using Network Simulation", Proceedings, 1981 Winter Simulation Conference, IEEE, Dec. 1981.
32. Mullarney, A., "SIMSCRIPT II.5 Programming Language", CACI, 1983.
33. Nance, R.E., et al, "Simulation Model Management: Resolving the Technological Gaps", Proceedings, 1981 Winter Simulation Conference, IEEE, Dec. 1981.
34. O'Keefe, Robert, "Simulation and Expert Systems - A Taxonomy and Some Examples", Simulation, January 1986.
35. Oren, T.J., Zeigler B.P., "Concept for Advance Simulation Methodologies", Simulation, March 1979.
36. Pegden, Dennis C., et al, "Introduction to SIMAN", 1983.
37. Pegden, Dennis C., "Introduction to SIMAN", Proceedings of the 1985 Winter Simulation Conference.
38. Pritsker, A.A., "Application of SLAM", IIE Transaction, 1982.
39. Rundgren, W. P. and Standrige, C.R., "A Database Support Discrete Parts Manufacturing Simulation", Proceedings, 1981 Winter Simulation Conference, IEEE, Dec. 1981.

40. Russell, Edward C., "Building Simulation Models with SIMSCRIPT II.5", CACI, 1981.
41. Schriber, Thomas J., "Simulation using GPSS", Wiley, 1974.
42. Schroer, B.J., et al, "Just In Time Manufacturing System Simulation on a Microcomputer", Simulation, August 1985.
43. Shannon, R.E., "System Simulation: The Art and Science", Prentice-Hall, 1975.
44. Shannon, Robert W., et al, "Comparison of Modeling Languages for Simulation of Automated Manufacturing Systems", AUTOFACT 5 Conference Proceedings, November 1983.
45. Standridge, C.R., "Using the Simulation Data Language", Simulation, Sept. 1981.
46. Standridge, Charles R., "Performing Simulation Projects with the Extended Simulation System (TESS)", Simulation, Dec. 1985.
47. Swezey, Robert W., et al, "A Case Study of Human Factors Guidelines in Computer Graphics and Applications, November, 1983.
48. Torn, Aimo A., "Simulation Graphics: A General Tool for Modeling Simulation Design", Simulation, Volume 37, December 1981.
49. Yancey, D.P., et al, "ICAM Decision Support System (IDSS), Third Interim Technical Report", June, 1983.
50. Zeigler, B.P., "Concepts and Software for Advanced Simulation Methodologies", Proceedings, 1980 Winter Simulation Conference, IEEE, Dec. 1980.

APPENDIX A
SUMMARY OF BCSL MODEL BLOCKS

This appendix consists of a list of all the various BCSL model Blocks implemented so far in this language, arranged in alphabetical order according to Block Operation. This information is provided for each Block.

1. A picture of the Block is shown.
2. The Block Operation is given and the available choices of Auxiliary Operators are listed.
3. List of block's operands and role of each operand is indicated.

Each operand is represented in following format;
Operand = Explanation : Available choices list

Table A-1 shows the abbreviations used in supplying The range of choices available for the Operands.

TABLE A-1. Explanation of the Abbreviations

ABBREVIATION	MEANING
k	AN ARITHMETIC EXPRESSION
sn	A TEXT STRING, A SYMBOLIC NAME
SNA	FAMILY NAME OF A STANDARD NUMERICAL ATTRIBUTE (see appendix c for more information)
SNA\$sn	FAMILY NAME OF A STANDARD NUMERICAL ATTRIBUTE, FOLLOWED BY A DOLLAR SIGN (\$) AND A SYMBOLIC ENTITY NAME

ADVANCE A, B, C

A = Mean Time: k, SNA\$sn

B = First Spread Modifier: k, SNA\$sn
or Function Modifier: FN\$sn, DS\$sn

C = Second Function Argument: k

ASSIGN A, B, C

A = Parameter Number [+]: k, SNA\$sn

B = Value to be Assigned: k, SNA\$sn

C = Number of Function Modifier: k, SNA\$sn

DEPART A, B

A = Queue Name: sn, k, SNA\$sn

B = Number of Units: k, SNA\$sn

ENTER A, B

A = Storage Name: sn, k, SNA\$sn

B = Number of Units: k, SNA\$sn

GATE LS A, B
LR

A = Logic Switch Name: sn, k, SNA\$sn

B = Next Block if Condition is False: sn, k, SNA\$sn

GATE NI A, B
I
NU

U

A = Facility Name: sn, k, SNA\$sn

B = Next Block if Condition is False: sn, k, SNA\$sn

GATE SE A, B
SF
SNE
SNF

A = Storage Name: sn, k, SNA\$sn

B = Next Block if Condition is False: sn, k, SNA\$sn

GENERATE A, B, C, D, E, F, G, H, I

A = Mean Time: k, SNA\$sn

B = Spread Modifier: k, SNA\$sn
or Function Modifier: FN\$sn, DS\$sn

C = Offset Interval: k, SNA\$sn

D = Limit Count: k, SNA\$sn

E = Priority Level: k, SNA\$sn

F = Number of Parameters: k, SNA\$sn

G = Type of Parameters: [F]

H = Transaction Name: sn

I = Second Function Argument: k

LEAVE A, B

A = Storage Name: sn, k, SNA\$sn

B = Number of Units: k, SNA\$sn

LOGIC I A
R
S

A = Logic Switch Name: sn, k, SNA\$sn

LOOP A, B

A = Parameter Number: k, SNA\$sn

B = Next Block if Parameter = 0: sn, k, SNA\$sn

MSAVEVALUE A, B, C, D

A = Matrix Name: sn, k, SNA\$sn

B = Row Number: k, SNA\$sn

C = Column Number: k, SNA\$sn

D = Value to be Saved: k, SNA\$sn

PRINT A, B

A = Lower Limit: sn, k, SNA\$sn

B = Upper Limit: sn, k, SNA\$sn

QUEUE A, B

A = Queue Name: sn, k, SNA\$sn

B = Number of Units: k, SNA\$sn

RELEASE A

A = Facility Name: sn, k, SNA\$sn

SAVEVALUE A, B, C

A = Savevalue Name [+]: sn, k, SNA\$sn

B = Value to be Saved: k, SNA\$sn

C = Savevalue Type: Real or Integer

SEIZE A

A = Facility Name: sn, k, SNA\$sn

TABULATE A, B

A = Table Name: sn, k, SNA\$sn

B = Weighting Factor: k, SNA\$sn .

TERMINATE A

A = Terminate Counter Decrement: k, SNA\$sn

TEST G A, B, C
GE
E
NE
LE
L

A = First Value: k, SNA\$sn

B = Second Value: k, SNA\$sn

C = Next Block if Condition is False: sn, k, SNA\$sn

TRANSFER A, B, C
(Statistical)

A = Selection Mode: k, SNA\$sn

B = First Block: sn, k, SNA\$sn
C = Second Block: sn, k, SNA\$sn

TRANSFER: A, B
(Unconditional)

A = Selection Mode: Not Used
B = Next Block Entered: sn, k, SNA\$sn

ENTERSTAT A, B, C, D, E, F

A = Station Name: sn
B = Number of Units: k
C = Mean Time: k
D = Distribution Function: DS\$sn, FN\$sn
E = Parameter 2: k
F = Parameter 3: k

LEAVESTAT A

A = Station Name: sn

REQWSTAT A, B, C, D, E, F

A = Station Name: sn
B = Number of Units: k
C = Mean Time: k
D = Distribution Function: DS\$sn, FN\$sn
E = Parameter 2: k

F = Parameter 3: k

REQTRANS A, B

A = Transport Name: sn

B = Next (Destination) Workstation: sn

APPENDIX B

SUMMARY OF BCSL DEFINITION AND CONTROL BLOCKS

In this appendix definition and control blocks are listed alphabetically, according to their "Operation." The following information appears for each block

1. The role of the field is briefly described.
2. The available choices in supplying the information required in the various fields are then summarized.

The definition and control blocks do not fit into a uniform scheme for summary purposes. Except for the common concept of "location, Operation, and Operands" fields. Each operand is represented in following format;

Operand = Explanation : Available choices list

CLEAR A, B

A = Savevalue(s) not to be cleared: Xj, S\$sn

B = Delimiter if Multiple Entries: ,

LOC EQ A,B

LOC = Symbolic Name of Entry: sn

A = Numeric Equivalent value of Symbolic name: k

B = Mnemonic for Entity type:

LOC FUNCTION A, B, C,

LOC = Name of Function

A = Function Argument: SNA\$sn, except MX

B = Function type and No. of Points: C,D,E,L,M

C = Variable Type: Real, Integer

INITIAL A, B

A = Logic Switch(s) to be set: LS\$2n

B = Delimeter if Multiple Entries: /

INITIAL A, B, C

A = Matrix Savevalue(s): X\$sn

B = Initial Value: k

C = Delimiter if Multiple Entries: /

INITIAL A, B, C

A = Savevalue(s): Xj, S\$sn

B = Initial Value: k

C = Delimiter if Multiple Entries: /

LOC MATRIX A, B, C,

LOC = Name of Matrix: k, sn

A = Matrix type: x

B = Number of Rows: k

C = Number of Columns" k

LOC QTABLE A, B, C, D,

LOC = Name of Table

A = Name of Queue: sn, k

B = Inclusive Upper Limit of Lowest Freq. Class: k

C = Width of Intermediate Frequency: k

D = Number of Frequency Classes: k

LOC ATABLE A, B, C, D

LOC = Name of Table

A = Name of Internal Variable: sn

B = Inclusive Upper Limit of Lowest Freq. Class: k

C = Width of Intermediate Frequency: k

D = Number of Frequency Classes: k

START A, B, C, D, E

A = Initial Value of Termination Counter: k

B = Printout Suppression: NP

C = Initial Value of Snap Interval Counter: k

D = Signal for Chain Printouts: 1

LOC STORAGE A

LOC = Name of Storage: k, sn

A = Storage Capacity: k

STORAGE A, B, C

A = Reference to Storage: S\$sn

B = Storage Capacity: k

C = Delimiter if Multiple Entries: /

LOC TABLE A, B, C, D

LOC = Name of Table

A = Table Arguments: k, SNA\$sn

B = Inclusive Upper Limit of Lowest Freq. Class: k

C = Width of Intermediate Frequency: k

D = Number of Frequency Classes: k

LOC VARIABLE A

FVARIABLE

LOC = Name of Variable: k, sn

A = COMBINATION OF NUMERIC DATA SPECIFICATIONS
AND ARITHMETIC OPERATORS

LOC OPERATION A, B, C, D, E

LOC = Define Transport

A = Name: sn

B = Speed: k

C = Number of Units: k

D = Distance Table Name: sn

E = Starting Workstation Number: k

LOC DEFWSTAT A, B, C

LOC = Define Workstation

A = Name: sn

B = Number: k

C = Number of Units: k

APPENDIX C

SUMMARY OF BCSL STANDARD NUMERICAL ATTRIBUTES

The list of BCSL Standard Numerical Attributes implemented in this language are summarized below, listed alphabetically under the various entity headings. Expanded definitions and discussions can be found in the GPSS book by Schriber.

ENTITY -----	SNA ---	BRIEF DESCRIPTION -----
BLOCKS	N	TOTAL COUNT
	W	CURRENT COUNT
CLOCK	CI	VALUE OF RELATIVE CLOCK
FACILITIES	F	FACILITY STATUS (1=BUSY; 0=AVAILABLE)
	FC	CAPTURE COUNT
	FR	FRACTIONAL UTILIZATION
	FT	AVERAGE HOLDING TIME PER CAPTURE (INTEGERIZED)
MATRIX SAVEVALUES	MX(A,B)	VALUE OF ELEMENT IN ROW A, COLUMN B, FULLWORD MATRIX
QUEUES	Q	CURRENT CONTENT
	QA	AVERAGE CONTENT (INTEGERIZED)
	QC	ENTRY COUNT (TOTAL ENTRIES)
	QM	MAXIMUM CONTENT
	QT	AVERAGE RESIDENCED TIME (BASED ON QC)
	QX	AVERAGE RESIDENCED TIME (BASED ON QZ)
	QZ	ENTRY COUNT (ZERO ENTRIES)
RANDOM NUMBERS	RN	A NUMBER BETWEEN 0 AND 1

SAVEVALUES	X	VALUE OF FULLWORD SAVEVALUE
STORAGES	R	REMAINING CAPACITY
	S	CURRENT CONTENT
	SA	AVERAGE CONTENT (INTEGRIZED)
	SC	ENTRY COUNT
	SR	FRACTIONAL UTILIZATION
	SM	MAXIMUM CONTENT
	ST	AVERAGE HOLDING TIME PER UNIT
TABLES	TB	AVERAGE VALUE OF NONWEIGHTED ENTRIES
	TC	NUMBER OF NONWEIGHTED ENTRIES
	TD	STANDARD DEVIATION OF NONWEIGHTED ENTRIES
TRANS-ACTIONS	P	PARAMETER VALUE
	M1	RESIDENCE TIME IN MODEL
VARIABLES	BV	CURRENT VALUE OF BOOLEAN VARIABLE
	V	CURRENT VALUE OF ARITHMETIC VARIABLE

APPENDIX D
SYSTEM CONFIGURATION AND USERS GUIDE

The Graphical Simulation System, runs on an IBM 4341 computer on top of CMS operating system. The PL/1 and SIMSCRIPT compilers are required for running the Graphical Simulation System in addition to GDDM graphics packages. Figure D-1 shows the GSS operations environment.

Three types of terminals are supported by GSS user interface. The IBM 3279 terminal is a graphic display with color graphics. In this case the user interface splits the display in two sections; graphic zone and menu zone. The light pen is used to select menu items and the key board is used to move the cursor within the graphic region. The IBM 3277-GA terminal is a combination of the IBM 3277 for alphanumeric display and a Tektronix 614 high resolution graphic tube. The light pen is used to select the menus and the joy stick is used to move the cursor within the graphic region.

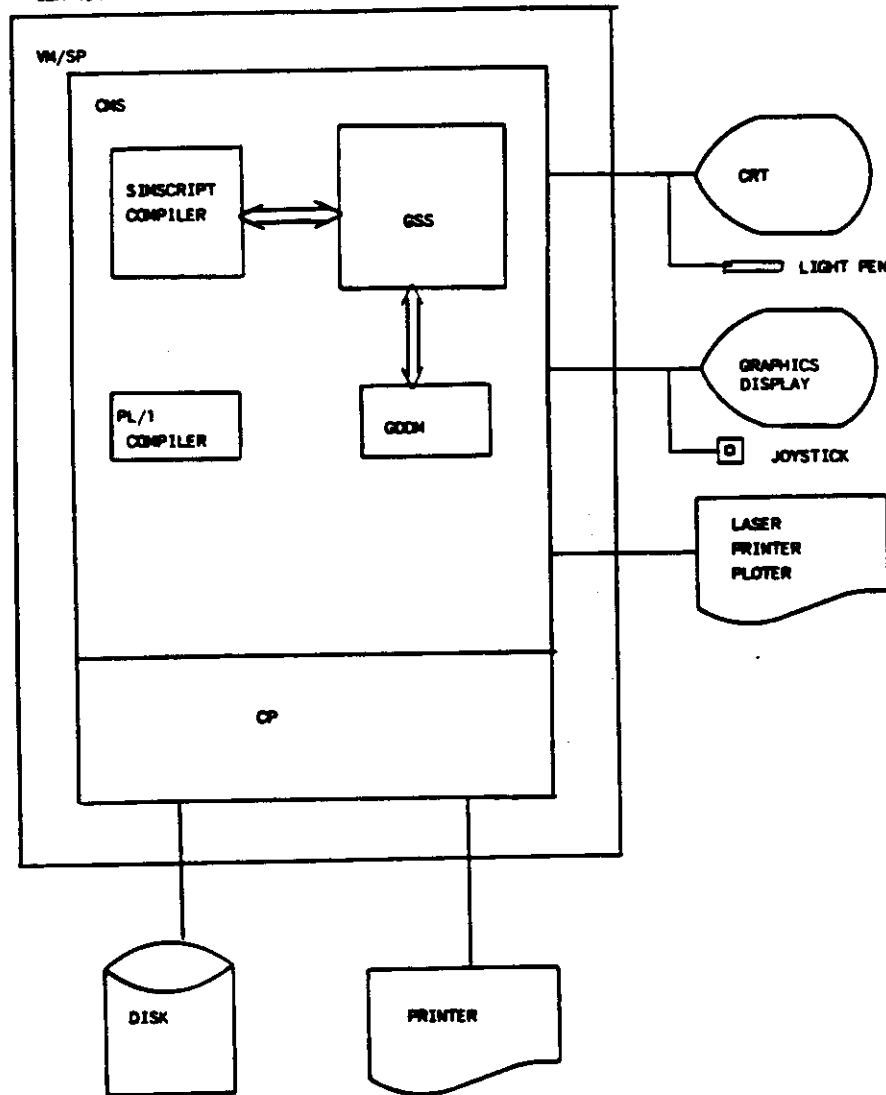


Figure D-1. GSS Operations Environment

The GSS supports the IBM 3290 plasma based terminal, which represents state-of-the-art in graphic terminals and allows two windows to operate in parallel. In addition the IBM 4250 laser printer can generate a hard copy of the graphic region on the 3279 terminal.

As discussed in Chapter 2, the GSS consists of 3 sections. The user interface (GRAPE) is written in PL/1 and relies on the CMS operating system utilities. The graphic support part of the user interface is done by the GDDM package. The BCSL code generation is written also in PL/1. The rest of GSS (BCSL Compiler and Output Generator) only require the SIMSCRIPT compiler and runs on any computer which supports SIMSCRIPT (including the IBM/PC).

In order to use the Graphical Simulation System on the UCLA CAD/CAM lab's IBM 4341; the user activates the user interface by typing GRAPE. The first menu (Figure D-2) allows the user to either start building a model from scratch by selecting the BUILD MODEL option or load an already existing model. The user loads an existing model by selecting SYSTEM UTILITIES on the first menu. The SYSTEM UTILITIES menu (Figure D-3) allows the user to load a model, save the mode, print or display any files and get a plot of the graphic model.

Once an existing model is loaded, the user can select the BUILD MODEL on the first menu to modify the model. Later the user can compile and execute the model by selecting the SIMULATION MODEL option. The SIMULATE MODEL menu (Figure D-4) allows the user to translate the graphical model, then compile the BCSL and finally execute the model. In each step the generated files are displayed.

Through the SYSTEM UTILITIES menu the user can change the model settings (Figure D-5), where the size of the graphical grid, scroll region and default language can be selected. The GSS user interface allows GPSS/PL1 and BCSL to be selected as the simulation base (the GPSS/PL1 is a GPSS compatible language which is PL/1 based). For further information on the User Interface package (GRAPE) and detail operations manual, refer to Mesrobian [28].

USE THE LIGHT PEN TO MAKE A SELECTION FROM THE MENU

CURRENT MENU PATH:

BUILD MODEL

SIMULATE MODEL

SYSTEM UTILITIES

EXIT GRAPE

PF1 = HELP

Figure D-2. Main Menu

USE THE LIGHT PEN TO MAKE A SELECTION FROM THE MENU

CURRENT MENU PATH: /SYSTEM UTILITIES

CLEAR MODEL

PRINT FILES

DISPLAY FILE

REMOVE FILE

LIST MODELS

SAVE MODEL

MODEL SETTINGS

USE MODEL

PLOT MODEL

PF1 = HELP

PF3 = CANCEL/EXIT

Figure D-3. System Utilities

USE THE LIGHT PEN TO MAKE A SELECTION FROM THE MENU

CURRENT MENU PATH: /SIMULATE MODLE

GENERATE CODE

COMPILE MODEL

EXECUTE MODEL

PF1 = HELP

PF3 = EXIT

Figure D-4. Simulate Model

SUPPLY THE FOLLOWING INFORMATION FOR THE COMMON
SELECTED THEN PRESS ENTER

NUMBER OF ROWS IN THE MODEL GRID (10-99) ==> 50
NUMBER OF COLUMNS IN THE MODEL GRID(10-99) ==> 50
PERCENT OF SCREEN TO SCROLL UP (25-75) ==> 50
PERCENT OF SCREEN TO SCROLL DOWN (25-75) ==> 50
OF SCREEN TO SCROLL LEFT (25-75) ==> 25
SIMULATION LANGUAGE (GPSSPLI OR BCSL) ==> BCSL

PF1 = HELP PF3 = CANCEL/EXIT

Figure D-5. Model Settings