RADC TR-61-254

THE UNIVERSITY OF MICHIGAN
OFFICE OF RESEARCH ADMINISTRATION
ANN ARBOR

Final Report

June 27, 1960, to August 31, 1961

A SIMULATION OF THE AN/FSQ-27 DATA-PROCESSING SYSTEM

Cooley Electronics Laboratory
Department of Electrical Engineering

By: R. A. Carlsen
M. G. Feingold
D. W. Fife

Approved by: B. F. Barton

Project 03767

Contract No. AF 30(602)-2337
Rome Air Development Center
Griffiss Air Force Base
New York

August 1961

ABSTRACT

This report discusses a simulation program, and its utility, for the AN/FSQ-27 Data-Processing System. A specific simulation run is discussed.

A queuing theory model is used to obtain the length of the simulation run required to obtain statistically significant results for a number of measures of system performance. These data support the conclusion that a complex simulation, as described in this report, is an expensive general-purpose research tool, although not necessarily an expensive technique for obtaining specific desired results. For research purposes, one should rather, employ analytical tools entirely and/or supplement a simulation technique by drawing upon the results obtained from analytical models for significant portions of the over-all system model.

TABLE OF CONTENTS

	Page
LIST OF TABLES	viii
LIST OF FIGURES	xi
1. INTRODUCTION	1
2. DESCRIPTION OF THE AN/FSQ-27 SYSTEM	3
2.1. Description of Equipment Characteristics	3
2.2. The Master Program	4
3. DESCRIPTION OF THE SIMULATION OF THE AN/FSQ-27	7
3.1. Fundamental Concepts	7
3.1.1. The Time Log	7
3.1.2. Representation of Computer Activity	8
3.1.3. Simulation Parameters	10
3.2. Functional Description of the Modeled System	11
3.2.1. Processing a Job Program	11
3.2.2. The Master Computer	14
3.2.3. Job Computers	17
3.3. Mechanics and Techniques	19
3.3.1. Simulation Parameters	19
3.3.2. The Time Log	19
3.3.3. The Input Generating Program	21
3.3.4. Minor Joints	24
3.3.5. Joint Number Bookkeeping	27
3.3.6. Alerts and Interrupts	29
3.3.7. Modeling of the Central Exchange	31
3.3.8. Master Computer Joints	32
3.3.9. Job Computer Functions	34
3.3.10. Major Joints	35
4. QUEUEING THEORY STUDIES	37
4.1. Introduction	37
4.2. Fundamentals of Queueing Theory	38
4.2.1. Equations of Detailed Balance	39
4.2.2. Cascading and Paralleling of Exponential Phases	47
4.2.3. Summary	51

Page

4.3. Queueing in Exponential Service Facilities 51

4.3.1. Single Channel (M = 1) 52
4.3.2. Multiple Channel (M > 1) 68
4.3.3. Summary 83

4.4. Application of the Exponential Model to the Polymorphic Computer 83
4.5. First Passage on the n^{th} State 91

4.5.1. Solution for First-Passage Probability Distribution 92
4.5.2. Additional Problems 94
4.5.3. Summary 99

4.6. Job Computer Queueing for Use of Shared Modules 99

4.6.1. Solution for Constant Arrival and Service Times 100
4.6.2. Exponential Arrival- and Service-Time Distributions 101

4.7. The Effect of Arrival- and Service-Time Distributions on System Behavior 104

4.7.1. Average Number in the Single-Channel Queue System Arbitrary Arrival- and Service-Time Distributions 107
4.7.2. Transient Solution for the State Probabilities 113
4.7.3. Summary 113

5. STATISTICAL ACCURACY OF MEASUREMENTS FROM SIMULATION DATA 117

5.1. Introduction 117
5.2. General Discussion of Statistical Errors 118
5.3. Calculation of Sample Sizes 120

5.3.1. Mean Arrival and Service Time 120
5.3.2. Mean Throughput Time 121
5.3.3. Measurement of Mean Number in the System 123
5.3.4. Measurement of State Probabilities 124
5.3.5. Measurement of the Mean First-Passage Time to the n^{th} State 126

5.4. Length of Simulation Run 126
5.5. Summary 128

6. SUMMARY AND CONCLUSIONS 131

6.1. Capability of the Simulation 131
6.2. Limitations of the Simulation 132
6.3. The Queueing Theory Model 133

Page

7. REFERENCES 135

APPENDIX A. DETAILS OF THE SIMULATION PROGRAM 137

A.1. Operating Instructions 137

A.1.1. The Program Deck 137
A.1.2. Distribution Parameters 139
A.1.3. Input 139
A.1.4. Output 164
A.1.5. Timing 169
A.1.6. Scheduled Error Stops 172

A.2. Flow Charts and Coding 174

A.2.2. The Start Subroutine 174
A.2.3. The TIMELG Subroutine 174
A.2.4. The MAIN2 Routine 183
A.2.5. The ALERT Subroutine 183
A.2.6. The DIST Subroutine 183
A.2.7. The IGP Subroutine 195
A.2.8. The JPMAJ Subroutine 195
A.2.9. The MPMIN Subroutine 195
A.2.10. The NOTE Subroutine 195
A.2.11. The ONEMD Subroutine 225
A.2.12. The SETMD Subroutine 225

APPENDIX B. EXAMPLE OF A SIMULATION RUN 231

B.1. Introduction 231
B.2. Description of Simulation Parameters 231

B.2.1. Master Program 231
B.2.2. Job Programs and Arrivals 231
B.2.3. Equipment Parameters 233

B.3. Results of the Run 235

B.3.1. General 235
B.3.2. Job Arrivals 236
B.3.3. Job Execution Time 236
B.3.4. Throughput Time 241
B.3.5. Average Number of Jobs in the System, and State Probabilities 241

B.4. Conclusions 242

Page

APPENDIX C. AUXILIARY PROGRAMS 245

C.1. Transient Solutions for the Exponential Model 245
C.2. First-Passage Time Distribution 250
C.3. Computation of Roots 250
C.4. Transient Solution for Erlang 2 Service Time 251
C.5. Transient Solution for Hyper-Exponential Service-Time Distribution 255

DISTRIBUTION LIST 261

LIST OF TABLES

Table		Page
I.	Simulation Tables	20
II.	State Probabilities for a Single-Channel System	54
III.	Transient Solutions for a Single-Channel System With $\rho = 0.25$, $P_0(0) = 1.0$	70
IV.	State Probabilities for a Two-Channel System	72
V.	State Probabilities for a Seven-Channel System	73
VI.	Transient Solutions for a One- and a Two-Channel System ($M = 1$, $N = 10$; $M = 2$, $N = 10$)	81
VII.	System Performance Chart	85
VIII.	The Probability of Immediate Service	90
IX.	System Performance Chart. $P(t_w \geq t) = .01$	90
X.	Steady-State State Probabilities for an Infinite-Queue, Seven-Channel System	94
XI.	Ratio of Hyper-Exponential Variance and Exponential Variance	111
XII.	Expected Number in the Single-Channel System With Exponential Arrival Time	112
XIII.	Sample Sizes for Measurements of Exponentially Distributed Arrival or Service Time	121
XIV.	Transient Values for Q_M, $E(t_p)$, and $Var(t_p)$	123
XV.	Calculated Values of Sample Size for Different Sampling Intervals in Measurement of Mean Throughput Time	124
XVI.	Sample Sizes and Sampling Intervals for Measurements of Mean Number in the System	125
XVII.	Sample Sizes for Measurement of State Probabilities	125

Table		Page
XVIII.	Sample Sizes for the Measurement of the Mean First-Passage to the m^{th} State	127
XIX.	Average Interval Between Samples	127
XX.	Average Duration of Simulation Run for Measurements	129
XXI.	The Distribution Parameters	140
XXII.	The MISC Table	156
XXIII.	The Identification of a Job's Progress on a Program Progress Card	171
XXIV.	Simulation Tables	172
XXV.	MAIN1 Routine	175
XXVI.	START Subroutine	176
XXVII.	TIMELG Subroutine	181
XXVIII.	MAIN2 Routine	184
XXIX.	ALERT Subroutine	185
XXX.	DIST Subroutine	189
XXXI.	IGP Subroutine	196
XXXII.	JPMAJ Subroutine	200
XXXIII.	MPMIN Subroutine	203
XXXIV.	NOTE Subroutine	223
XXXV.	ONEMD Subroutine	226
XXXVI.	SETMD Subroutine	228
XXXVII.	Master Program Description	232

LIST OF TABLES (Concluded)

Table		Page
XXXVIII.	Summary Description of the Run	233
XXXIX.	Calculation of Predicted Run Time	235
XL.	Comparison of Arrival Time Distribution	236
XLI.	Results for Job Execution Time	239
XLII.	Results for the Number of Jobs in Progress and in the System	241
XLIII.	State Probabilities Obtained from Data	242
XLIV.	Makeup of Input Deck for One Solution	247
XLV.	Printed Output Format	248
XLVI.	$\dot{P}] = [U]\ P]$ Program	249
XLVII.	Input Format for Root Computation	251
XLVIII.	Root Computation Program	252
XLIX.	Input Format for Transient Solutions With Erlang 2 Service Time	254
L.	$\dot{P}] = [U]\ P]$ (Erlang 2) Program	256
LI.	Input Format for Transient Solutions for Hyper-Exponential Service Time	258
LII.	$\dot{P}] = [U]\ P]$ (Hyper-Exponential) Program	259

LIST OF FIGURES

Figure		Page
1.	Basic elements of the polymorphic computer system.	1
2.	Master computer functions.	5
3.	Job computer functions.	6
4.	Master computer events.	8
5.	The simulation mechanism.	9
6.	Job program representation.	10
7.	Processing a job program.	12
8a.	Master computer functions.	15
8b.	Master computer functions.	16
9.	Job computer functions.	18
10.	Program flow diagram.	22
11.	Generating independent job requests.	23
12.	Generating job request by use of sequences.	25
13.	Minor joint action.	26
14.	Queueing theory model for a polymorphic computer.	37
15.	Transitions to the n^{th} state.	40
16.	Transitions for a single-channel system.	43
17.	Transitions for a two-channel system.	45
18.	Transient for a single-channel system. Erlang Type Two service distribution.	49
19.	Queue length (L_Q) for a single-channel system ($M=1$, $N=\infty$).	57

Figure		Page
20.	Transient state probabilities for a single-channel system (M=1, N=3).	66
21.	Transient solution, $P_0(t)$, for a single-channel system (M=1, N=∞).	69
22.	Queue length (L_q) for a seven-channel system (M=7, N=∞).	75
23.	Root locus for a two-channel system.	78
24.	Transient state probabilities for a seven-channel system (M=7, N=20).	79
25.	Transient state probabilities for a seven-channel system (M=7, N=20).	80
26.	Mean waiting time vs. mean number in service (N=∞).	84
27a.	Waiting-time distributions for multiple-channel systems.	86
27b.	Waiting-time distributions for multiple-channel systems.	87
27c.	Waiting-time distributions for multiple-channel systems.	88
27d.	Waiting-time distributions for multiple-channel systems.	89
28.	First-passage to the n^{th} state.	91
29.	First-passage probability to state n = 7 for ρ = .5, .7, .9.	95
30.	First-passage probability to state n = 10 for ρ = .5, .7, .9.	96
31.	First-passage probability to state n = 20 for ρ = .9, .7.	97
32.	First-passage probability to state n = 20 and state n = 40 for ρ = .9.	98
33.	Representation of shared module queueing.	100
34.	Mean number in queue vs. ρ (M = 7).	102

Figure		Page
35.	Queue length (L_q) and throughput time T_s (normalized to T_a) vs. the shared module use factor.	105
36.	Job-computer vs. shared-module utilization.	106
37.	$P_o(t)$ for the single-channel system, $\rho = 0.25$.	114
38.	$P_o(t)$ for the single-channel system, $\rho = 0.5$.	114
39.	Distribution of a measured mean value.	119
40.	The program deck.	138
41.	Basic card format.	141
42.	An IGPLOG table.	144
43.	An IGPLOG card set.	145
44.	Numerical coding of module types.	146
45.	An IMPLOG table.	148
46.	An IMPLOG card set.	149
47.	An MCT table.	150
48.	A PURVUE table.	151
49.	A JCR card set.	153
50.	An MAT table.	154
51.	A MISC card set.	155
52.	A PERBUF table.	157
53.	A QUEUE table.	159
54.	A SEQQ table.	159
55.	A SEQQ card set.	161

Figure		Page
56.	Computer module TIMELG entries.	162
57.	Job request TIMELG entries.	163
58.	A TIMELG table.	165
59.	A TIMELG table.	166
60.	Job processing stages.	168
61.	Program progress cards (primary simulation output).	170
62.	Debugging aids.	173
63.	START subroutine—entry to START.	179
64.	START subroutine—entry to RESTAR.	180
65.	ALERT subroutine—entry to ALERTJ.	187
66.	ALERT subroutine—entry to ALERTM.	188
67.	DIST subroutine.	192
68.	IGP subroutine.	199
69.	JPMAJ subroutine.	202
70.	MPMIN subroutine—common section.	210
71.	MPMIN subroutine.	211
72.	MPMIN subroutine.	211
73.	MPMIN subroutine.	212
74.	MPMIN subroutine.	213
75.	MPMIN subroutine.	214
76.	MPMIN subroutine.	215
77.	MPMIN subroutine.	215

Figure		Page
78.	MPMIN subroutine.	216
79.	MPMIN subroutine.	217
80.	MPMIN subroutine.	217
81.	MPMIN subroutine.	218
82.	MPMIN subroutine.	218
83.	MPMIN subroutine.	219
84.	MPMIN subroutine.	219
85.	MPMIN subroutine.	220
86.	MPMIN subroutine.	220
87.	MPMIN subroutine.	221
88.	MPMIN subroutine.	221
89.	MPMIN subroutine.	222
90.	ONEMD subroutine.	227
91.	SETMD subroutine.	229
92.	Probability distribution of major joint durations.	234
93.	Comparison of distribution on arrival intervals.	237
94.	Scatter diagram of 100 uniform numbers generating the exponential arrival intervals.	238
95.	Comparison of distribution of job execution time with normal distribution.	240

1. INTRODUCTION

The purpose of this report is to describe the work completed under Air Force Contract No. AF 30(602)-2337 at The University of Michigan's Cooley Electronics Laboratory. The objective of this contract is the evaluation of the polymorphic computer concept as realized in the design of the Data Processing Center AN/FSQ-27.

The AN/FSQ-27 system is a large-scale, general-purpose, high-speed digital computer system. It is composed of a number of equipment elements or modules, categorized as either controlling elements or subordinate elements. Controlling elements may request connections and communicate with subordinate elements through a central exchange switching unit as shown in Figure 1.

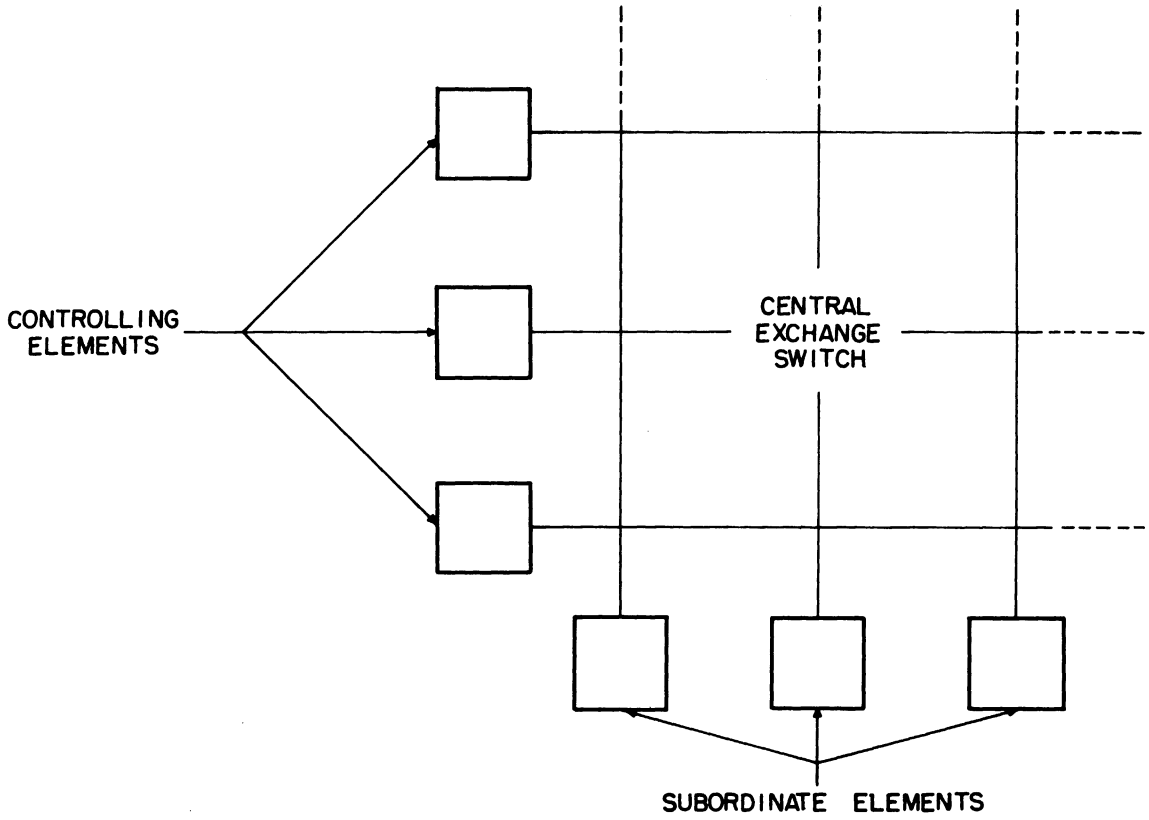

Figure 1. Basic elements of the polymorphic computer system.

During operation of the system, the available units can be grouped into a number of configurations of computing equipment, which may work simultaneously and independently in processing different computing jobs. At comple-

tion of any job, the modules which have been used in its processing then become available for assignment to another configuration, to begin processing a new computing request. The ability to change the equipment configuration into various "forms" or "shapes" gives rise to the term "polymorphic." The control and supervisory functions necessary to implement and coorindate this activity are incorporated into an executive or master control program. In the AN/FSQ-27 system, one configuration of equipment is assigned to the master control functions.

After an initial familiarization period was completed, it was decided that a simulation would provide the best approach to the evaluation of the polymorphic system, if the simulation could be made an order of magnitude simpler than the system being modeled. A Monte Carlo technique has the advantage of making it possible to model system detail which cannot be obtained by analytical methods. However, it became apparent during the contract period that a simulation approach must be augmented by an understanding of some simplified analytical models. Therefore, queueing theory has been applied, first, to determine the length of the simulation run required to obtain statistically significant results and, second, to gain insight into some of the basic problems encountered in a polymorphic system.

The detailed description of the AN/FSQ-27 and the simulation program are discussed in Section 2 and 3. The analysis of a queueing theory model and its direct application to the AN/FSQ-27 is covered in Section 4. In Section 5 the objections to a simulation approach are illustrated by an application of the queueing theory to the utilization of the present simulation. Finally, Section 6, brings together all the major conclusions from Sections 3-5. Although judgment of the value of simulation is, in the last analysis, subjective, the data support the conclusion that a complex simulation, as described in Section 3, is a very expensive analytical tool.

2. DESCRIPTION OF THE AN/FSQ-27 SYSTEM

The AN/FSQ-27 is best described by its hardware and software characteristics. It is assumed that a reader unfamiliar with this system will supplement the discussion in the following sections with the material in Ref. 1.

2.1. Description of Equipment Characteristics

The characteristics of the modular units, or modules, can be summarized as follows.

(1) Computer Module (CM). A computer with magnetic core storage. This module is the only unit which is capable of arithmetical and logical operations. It is the principal controlling module.

(2) Buffer Module (BM). A module composed of two independent magnetic-core-storage buffer units. Each buffer unit is internally programmed and serves principally as a data-transmitting unit. A buffer unit (BU) acts as a controlling unit when connected to subordinate units. It is the only other type of controlling module in the system.

(3) Drum Module (DM). A magnetic drum storage unit.

(4) Tape Module (TM). A magnetic tape storage unit.

(5) Peripheral Buffer (PB). A magnetic drum storage unit which serves as an input/output buffer between the system and operator consoles.

(6) Central Exchange (CX). A transfluxor switch matrix providing for a physical connection between any pair of controlling and subordinate modules, subject to program control. A large number of connections of module pairs may exist simultaneously.

(7) Miscellaneous input/output devices such as User Modules (UM), Printers (PR), Plotters (PL), and Display and Analysis consoles.

(8) Display Buffer (DB). A magnetic drum storage unit which services cathode-ray tube for display consoles.

Since the CX module is in many respects the heart of the system, it deserives further discussion. The path of communication between a controlling module and a subordinate module is through the matrix of the CX switch. When a controlling module wishes to transmit or receive data from a subordinate module, it must request a connection to that module via the CX. In the memory unit of the CX, a table of module assignments is maintained by which it is possible to restrict connections for each controlling module to a specified group of subordinate modules. Before a connection is established, the CX consults this table to verify that a requested connection is allowable. The listings in the module assignment table are under control of the master program (to be discussed in the next section). The process of establishing a connection also includes a test to insure that a requested subordinate module is not being used by some other controlling module.

The means of communication between controlling modules is via an "alert" signal generated at the CX and directed to a controlling module upon command of another controlling module. The allowable alert signals are also listed in the module assignment table, so it is possible to restrict communication between controlling units according to master program control. Upon receipt of an alert signal, the controlling module has the option, according to its program, of responding immediately or at a later time. If the number of alternative responses desired exceeds the number of different alert signals which can be sent to a controlling module, the polymorphic organization allows for additional communication between controlling units by giving two or more controlling modules access to the same subordinate modules, via the CX module assignment table.

2.2. The Master Program

Responsibility for control and coordination of the modules in the system is given to a master program, an executive routine that supervises the progress of all jobs through the system and monitors the status of the modules. Descriptions of two proposed master programs for the AN/FSQ-27 system were made available to this research group by Ramo-Wooldridge. The following paragraphs are an abstract of the basic features found in both proposals.

The master program is responsible for accepting the problems to be run on the system and for assigning to them the modules needed for their operation. One entire computer module, designated the master computer, is reserved for the operation of the master program. However, for proper communication, a part of the master program is also stored in every other computer module (that is, in every job computer), in the entire system. The master computer communicates with these job computers via the alert system and by the use of shared subordinate modules. Certain of these shared modules are used primarily by the master program for storage of information that cannot be kept

in the master computer memory. A drum module stores lists and descriptions of the condition of the problems being processed and the modules in the system. A tape unit may be needed for storage of programs.

After cycling through a basic loop of monitoring operations (Figure 2), the master program in the master computer checks the peripheral buffer to see whether there are any new requests. All such requests are placed in a queue, or list, of waiting jobs, in order of priority. The priority of a particular job may be assigned by the system itself or by an external agent.

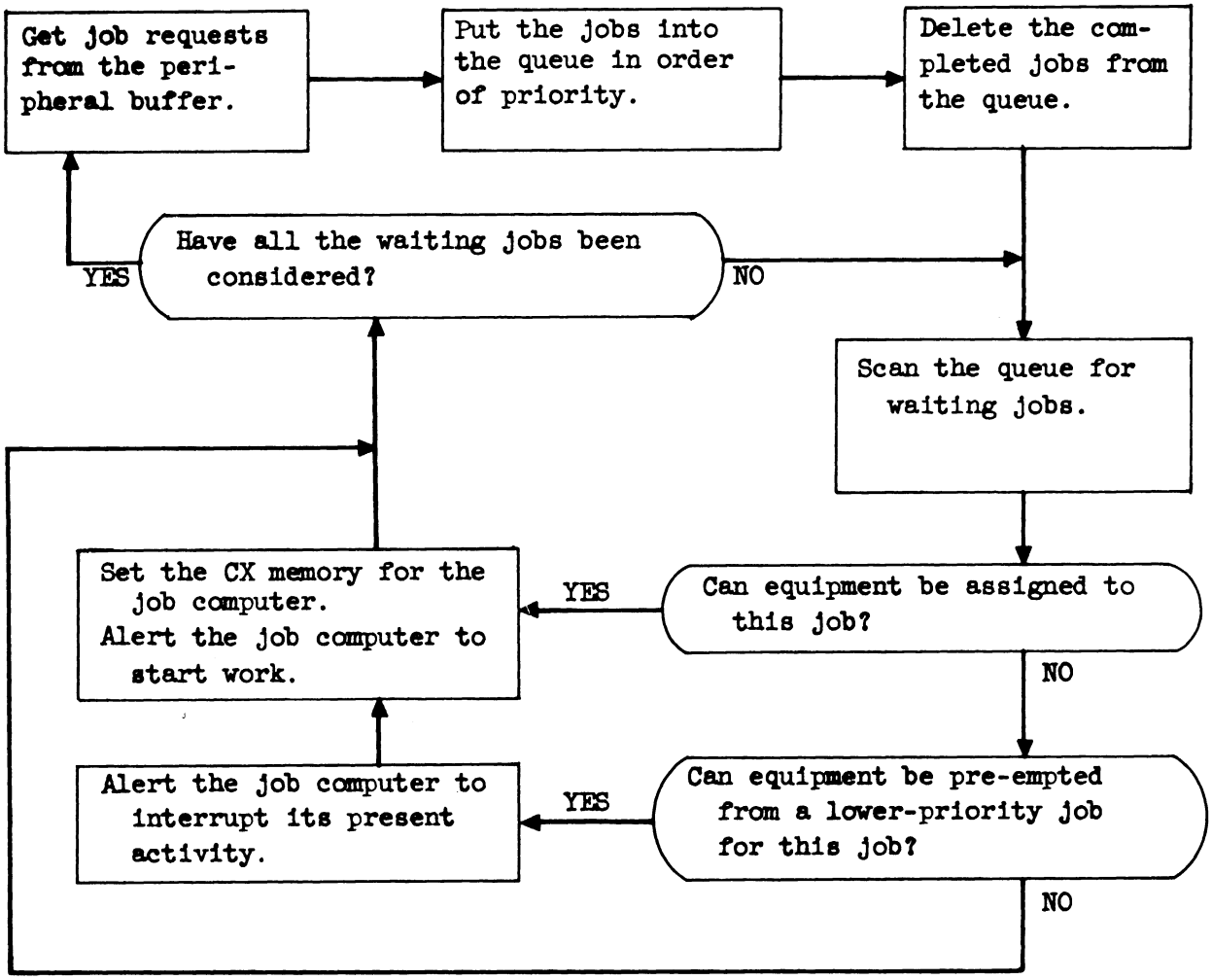

Figure 2. Master computer functions.

The queue is examined periodically to see if there are waiting jobs which can be assigned equipment and started. The alert system is used to interrupt the execution of jobs of low priority to allow early completion of more urgent requests. Just before a job computer starts working on a

new job, the master computer must note the equipment assignment in the CX memory and then alert the assigned computer module to start execution of that job.

Although the master computer is responsible for most of the master program functions, job computers perform some of these functions (Figure 3) as well. Input and output for job programs are controlled by the job computers through the peripheral buffer. When a job computer has executed a job, it notes this fact on the record of the job in the queue, and can then initiate action on any other job it finds in the queue which is ready to be started. It is the responsibility of the master computer to remove completed jobs from the queue.

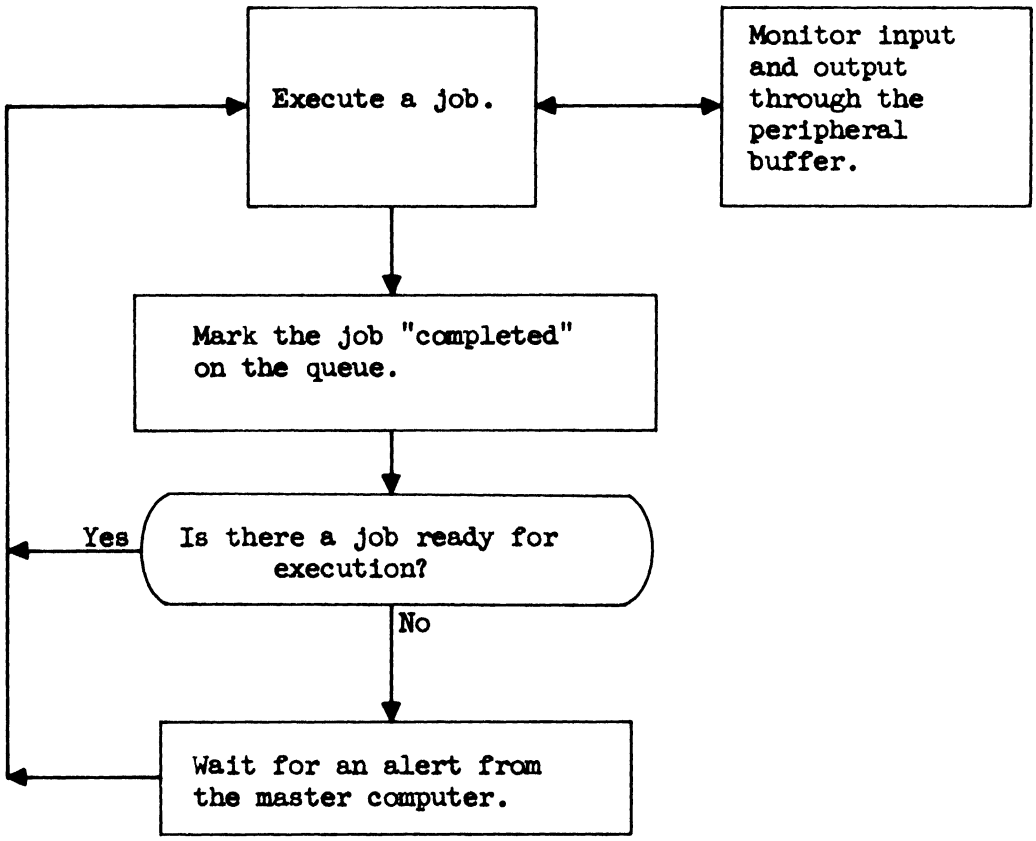

Figure 3. Job computer functions.

3. DESCRIPTION OF THE SIMULATION OF THE AN/FSQ-27

A program has been written to simulate, on the IBM-704, the AN/FSQ-27 data-processing system. The purpose of this section is to outline the features and flexibility of this program.

The equipment features of the system which were modeled are described in Section 2.1. Because the master program for the AN/FSQ-27 was not fully detailed at the time this simulation was written, the master program modeled is defined as consisting of precisely those features included in the simulation program and described in Section 3.2. This organization was chosen primarily for two reasons: (1) it includes the most basic characteristics of the master programs that Ramo-Wooldridge was developing, and (2) it is the least complicated organization that would allow a simulation of the system in the detail best suited for the study.

3.1. Fundamental Concepts

3.1.1. THE TIME LOG

The AN/FSQ-27 system can be viewed as a number of independently and simultaneously operating activities: the master program execution by the master computer, job requests arriving as system input, and processing of the job requests by the job computers. The history of each such element in the system which is operating in parallel with other elements can be depicted by a series of time intervals, each one marking the occurrence of a particular event. For example, we consider the activities of the master computer as a series of events representing the functions which the master computer performs while connected to a particular subordinate module (Figure 4). This time scale is not a part of the simulation program as such. Rather, the program operates through the use of a time log which is the superposition of the time scales for all the independent activities in the system. At the start of a simulation run, the time log consists of the initial events for each part of the system. The program then picks up the first entry in the log and operates on it; that is, it simulates the activities of the system element which are to start at that time, and it generates (usually) another entry in the time log (Figure 5). The intervals between successive events are generated by the use of input parameters. These intervals need not be of equal duration, either in the individual time scales or in the superimposed time log.

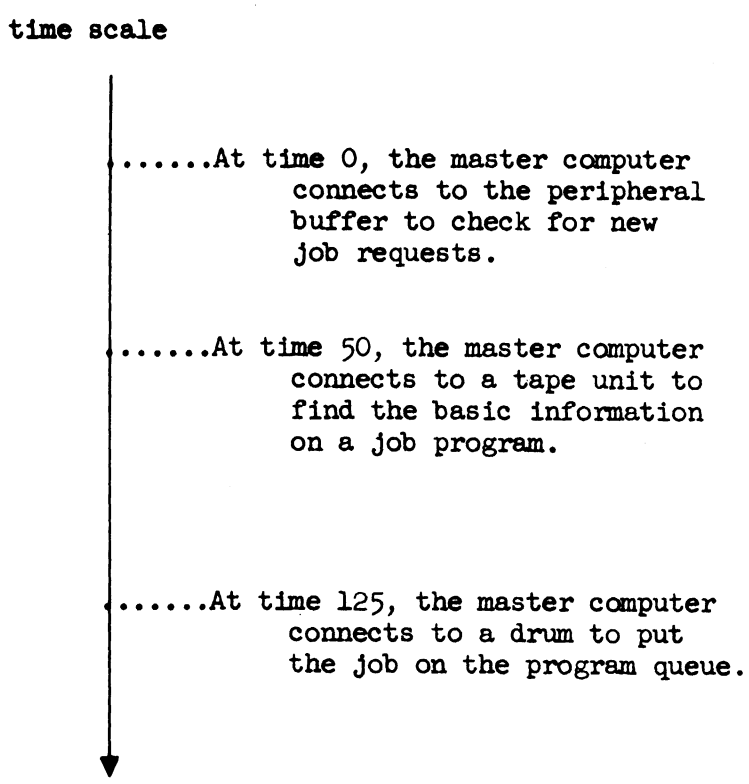

Figure 4. Master computer events.

3.1.2. REPRESENTATION OF COMPUTER ACTIVITY

The activities of computer modules that are modeled by the simulation are designated as "joints," periods of time in which the computer module is connected to a particular subordinate module ("minor" joints) or to a series of subordinate modules ("major" joints). The master computer's activities are represented by minor joints only. Associated with each joint is a particular master program function. The function is simulated when the joint appears in the time log.

Each job program can also be considered as a series of connections between a computer module and subordinate modules, each connection lasting a certain time. The modules concerned are those assigned exclusively to that job and the shared modules (e.g., the PB). The simulation program models these connections in different depths. Any consecutive series of connections to a number of exclusive subordinate modules is modeled as one interval of time, i.e., as one major joint. Any connection in which the subordinate module is a shared module as a minor joint. Moreover, each job program is presumed to be an alternating sequence of minor and major joints (Figure 6).

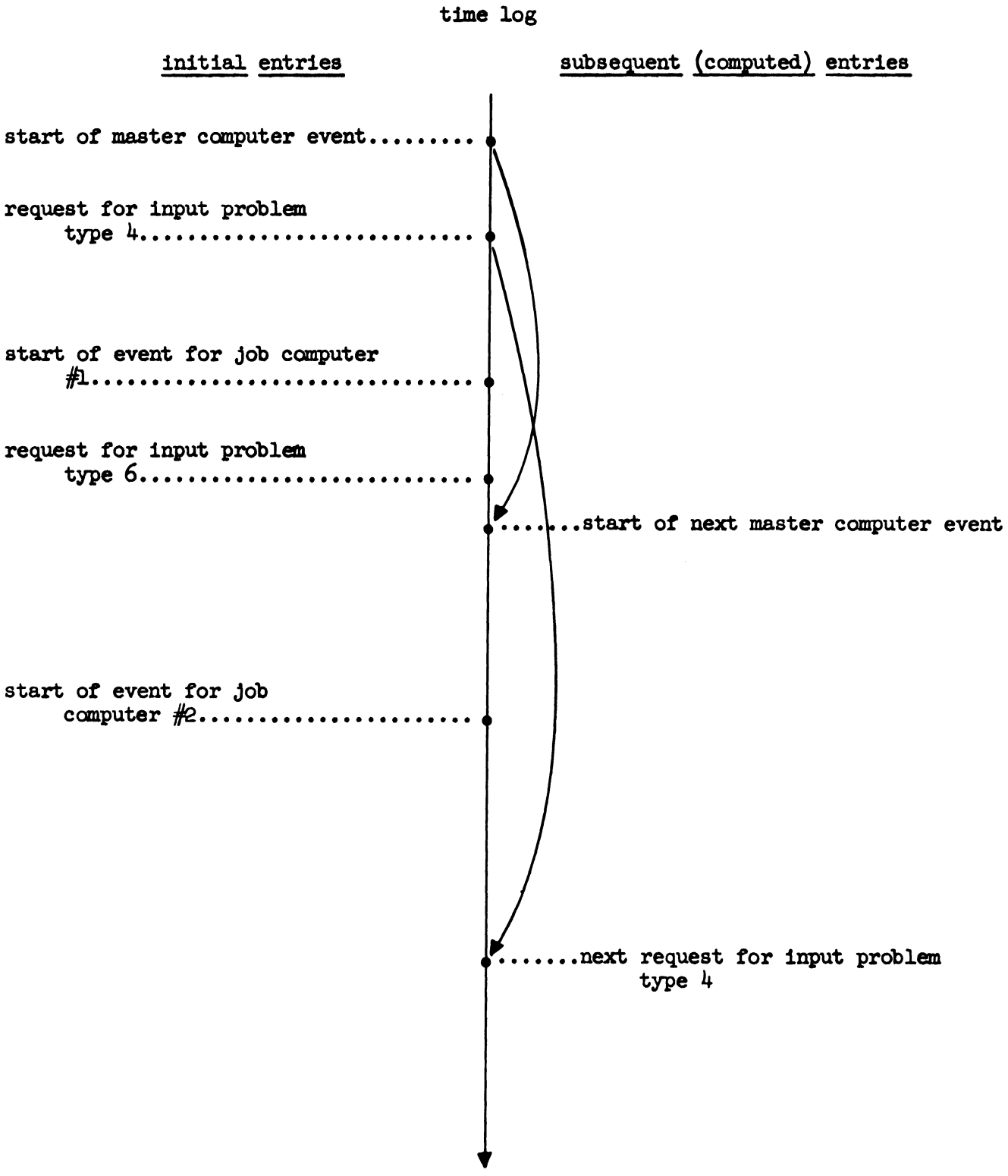

Figure 5. The simulation mechanism.

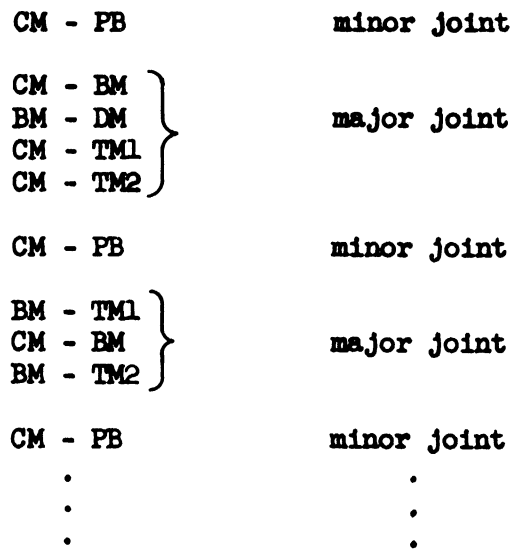

Figure 6. Job program representation.

3.1.3. SIMULATION PARAMETERS

Certain characteristics of the hardware are variables in the simulation. The system components—the number of modules of each type—can be specified. In addition, one can enter parameters which specify the access time to each module type used by the master program. (Access time to modules and exclusively by job programs is not used in the simulation.) Two aspects of the central exchange are variable: the switching time—the time to connect or disconnect two modules or to send an alert; and the time it takes the master computer to load the central exchange memory, during which period the CX is unavailable for any other purpose.

Several different problem types may comprise the input to the AN/FSQ-27 system, with each problem type being defined by the following characteristics:

(1) the rate of arrival of the problems;

(2) the number of major joints (which is also the number of minor joints);

(3) the length of the major joints;

(4) the length of the minor joints;

(5) the priority of the problem type;

(6) the modules required for the job; and

(7) the interrupt disposition of the job; that is, the time delay between the arrival of an alert from the master computer and this job's recognition of that alert (which is the time at which this job will be interrupted).

One may also specify a certain interdependance between jobs with respect to arrivals, so that the simulated request for job type a will always occur at a given time after the completion of job type b.

The master program is portrayed as a series of functions each of which is preceded by a connection to a subordinate module. Although the functions themselves are an integral part of the simulation program, it is possible to specify arbitrarily which controlled modules are involved and how long each function takes to perform. There is also a certain flexibility in the master program's ability to accept alerts from other computer modules.

Almost all the parameters listed above are given as random variables to be computed by the program. A code is given which tells the simulation what distribution function is required. Four such codes are incorporated in the program: a constant, an exponential distribution, a uniform distribution over two intervals, and a normal distribution. It is relatively easy to add other functions, and these additional distributions automatically become available for use for any simulation parameter.

Although it is simplest to start the simulation run from a condition of rest (with no jobs in progress), this is not necessary. The initial conditions are completely arbitrary within the limitation that the run must start at the beginning of an event on the time log.

3.2. Functional Description of the Modeled System

The following sections describe the events in an operating AN/FSQ-27 system from three different points of view: the passage of a job program through the system from its request to its completion, the activity of the master computer in processing jobs and monitoring the system, and the responsibilities and activities of the job computers.

3.2.1. PROCESSING A JOB PROGRAM

3.2.1.1. The Backlog

The input and output devices which communicate with the rest of the system via the peripheral buffer (PB) or the display buffer (DB) are not sim-

ulated as such; the simulation program models only the appearance of input in the PB and the placing of information by the system into the PB. The master program (MP) first becomes aware of the entry of a job into the system by the appearance of a job request in the peripheral buffer (Figure 7).

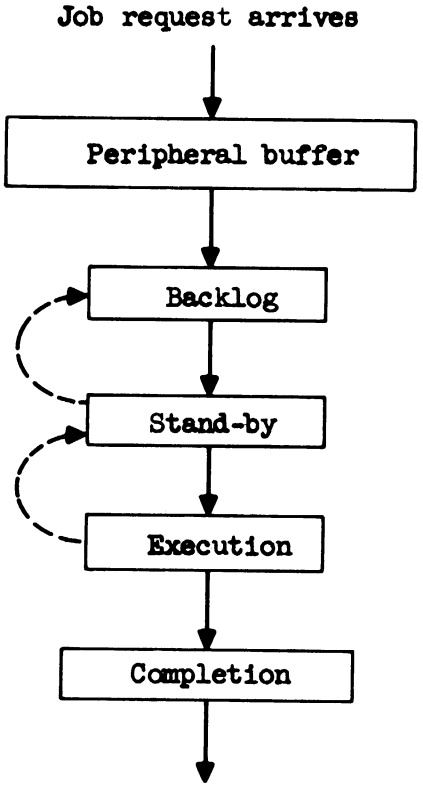

Figure 7. Processing a job program. Dotted lines are used to indicate the steps a job may be forced to take backwards due to the obtrusion of a higher-ranking job.

The master computer (MC), during its periodic inspection of the PB, removes the job from the PB and inserts it into a queue, the "program queue," which holds all the job programs in the system. At this point, the job is said to be in the backlog. The program queue is in order, first by priority number [determined by the input parameters and applicable to every problem of a given type (Section 3.3.3.)], and second by the time at which the job was requested. Rank will be used to refer to this double ordering. The highest rank belongs to the earliest job with the smallest priority number.

3.2.1.2. Stand-by status

The MC then tries to assign to this job those modules, exclusive of shared equipment and a computer module, which the job requires for execution. When these modules have been assigned, the job is said to be in the stand-by list. The master computer looks first at the list of available equipment, that is, those modules which are neither shared nor assigned to another job. When all the modules that a job needs are available, those modules are then marked as assigned to that job.

If all the necessary equipment is not available, the MC looks first for a lower-ranking job in the stand-by list whose assigned equipment, when added to that presently available, will satisfy the requirements of the higher-ranking job. In such a case, the job in stand-by loses all its modules and is returned to the backlog, whereupon the job in the backlog is assigned the modules it needs and placed in the stand-by list. As a second resort, the MC looks for a lower-ranking job already in execution, whose equipment could be pre-empted. If such a job is found, its job computer is alerted and the master computer waits for a response. Upon receiving that response (in the form of an alert, which indicates that the lower-ranking job has been demoted by its job computer from execution phase to stand-by), the master computer demotes that job still further to the backlog, and then proceeds to assign to the stand-by list the job it was originally working on.

If an input problem cannot be assigned to stand-by, it remains in the backlog until the MC again attempts this assignment (Section 3.2.2).

3.2.1.3. Execution Phase

After a job has reached the stand-by list, the master computer tries to find a computer module (CM) that can start executing the job. If there is a CM free, it is immediately appointed to the job and execution begins shortly. Otherwise, the MC looks for a job computer that is executing a job of lower rank. If it finds one, it alerts that job computer to stop and again waits for a response. At the response, the job computer is assigned the new job.

If a job is interrupted to yield its computer module to a higher-ranking job it is returned to the stand-by list. If it must also surrender its other equipment, it retrogresses to the backlog. In either event, it is then treated by the master computer exactly the same as any other job in the program queue. If the interrupted job is left in stand-by, the master computer does not attempt to find another job computer for it. When a job computer picks up an interrupted job from the program queue, that job may be restarted from exactly the point at which it was interrupted, or a restart penalty may be assessed. This restart penalty does not include the additional time which must be spent by the master program in again assigning equipment to the job, etc.

3.2.1.4. Completion

When the job is completed, at the end of its last major joint, it is so marked in the program queue. The master program, in its periodic inspection of the queue, reports the completion of the job to the system operator (via a message through the peripheral buffer) and at this time the problem is removed from the queue.

3.2.2. THE MASTER COMPUTER

The master program in the master computer cyles through a basic computation loop, occasionally answering an alert from a job computer, and then returning to the point at which it was interrupted (Figure 8).

The peripheral buffer is examined periodically to see if any job requests have arrived. Each job is removed from the PB, its priority number and equipment requirements are obtained from the purivew table, which contains the specifications of all the job types, and it is then inserted into the program queue. All the job requests in the PB are processed, with each job put into the queue before the next is removed from the buffer.

The master program then examines the program queue to see if any jobs have been marked by the job computers as completed. These are removed from the queue, their completion reported to the system operator, and their assigned modules put back into the pool of available equipment. Again, all completed jobs are removed from the queue at this time, and processing is carried out on one job at a time.

If there were no new job requests and no released modules were made available, the basic cycle is completed at this point and the master computer loops back to re-examine the peripheral buffer. Otherwise, the MP scans the program queue attempting to assign equipment to jobs so that they may be executed. This scanning starts with the highest-ranking job in the queue and covers all the jobs in the queue in backlog status. The MP first tries to find the equipment required for the job from among those modules currently available. If sufficient equipment cannot be found, the MP scans the queue in reverse order of rank to find a single job whose modules, if released, would enable the higher-ranking job to be moved to stand-by. The queue is inspected first for jobs in stand-by and then for jobs already in progress: that is, a stand-by job is selected for demotion rather than a job being executed, regardless of their relative rank. In either case, when the master program demotes a job from stand-by to backlog, it replaces <u>all</u> its modules into the pool. If a job in the backlog cannot be assigned to stand-by, the master computer selects the next (in rank) backlog item in the program queue and proceeds to work on it. If a job in progress is interrupted, its priority number is decreased (made higher) by 1 and it is re-ranked in the queue accordingly.

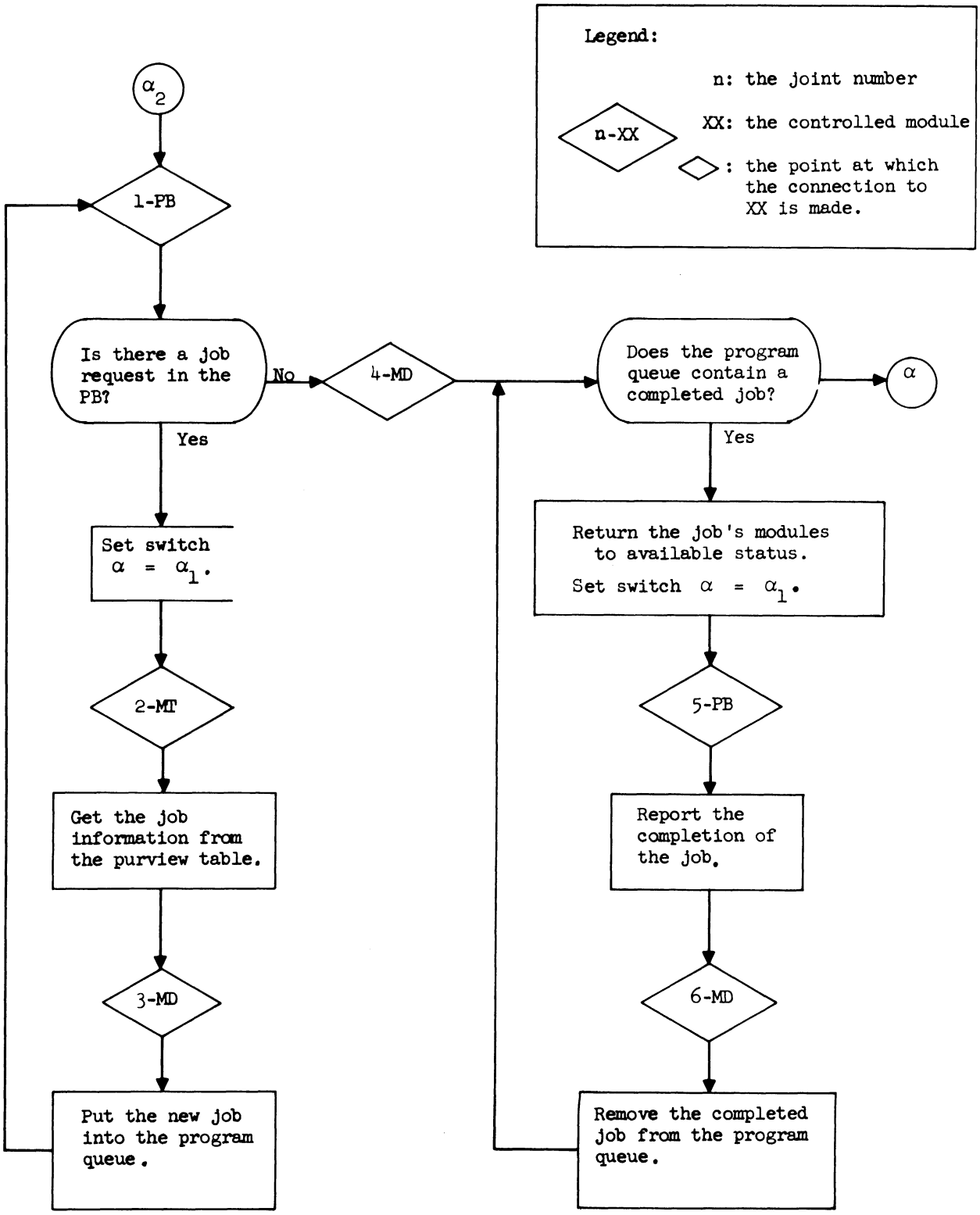

Figure 8a. Master computer functions.

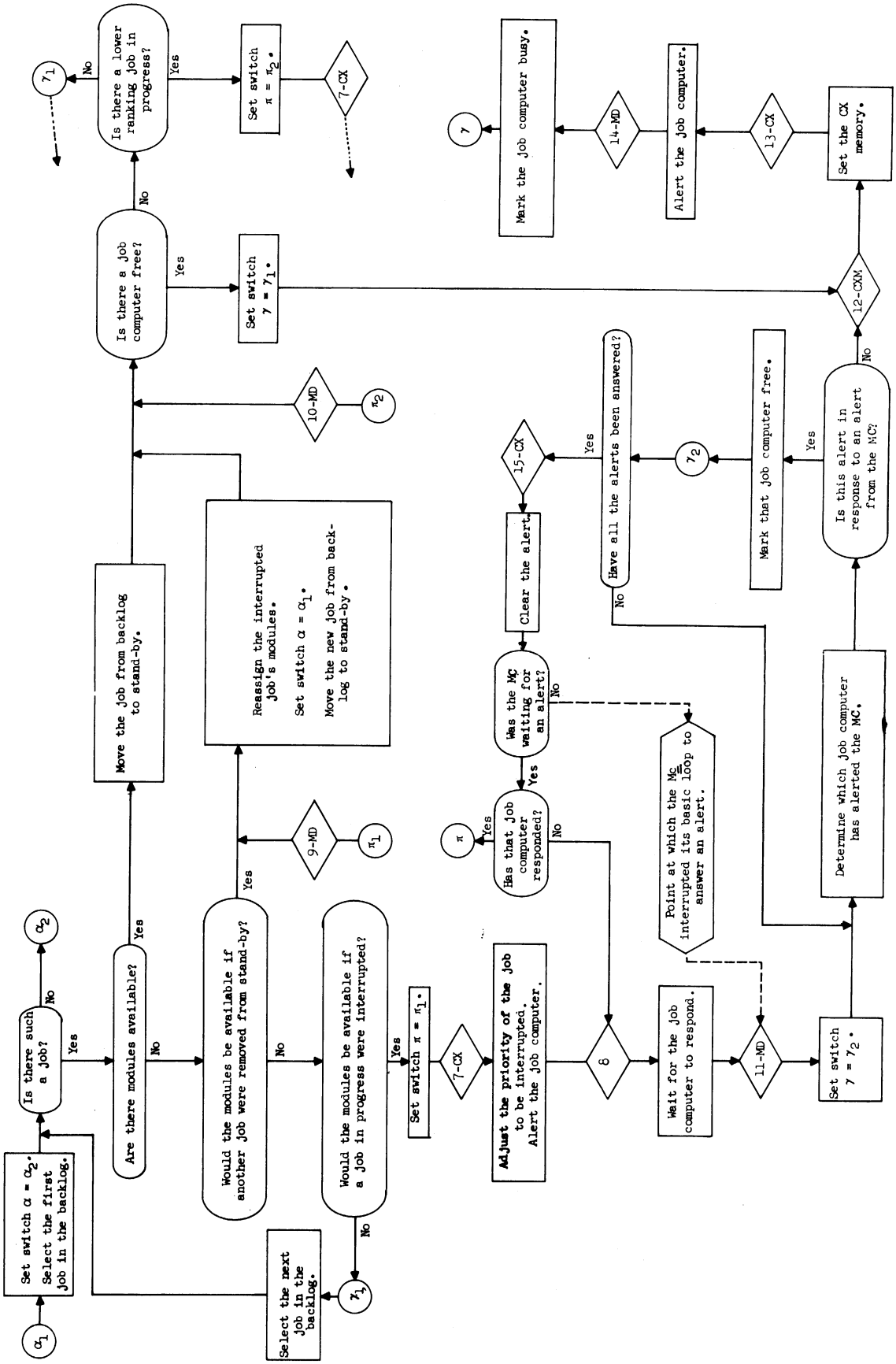

Figure 8b. Master computer functions.

If it is necessary for the MC to alert a job computer which is executing a job, the MC issues the alert and then waits for a response. In effect, the master computer module is engaged in a loop which will be broken only by the arrival of an alert from a job computer. Upon receipt of an alert from any computer in the system, the master computer leaves its loop and acts upon the alert. If that alert was not the one for which the MC was waiting, the MC will re-enter the waiting loop.

Only the master computer can set the CX memory for the job computers. There are two points in the master program at which this is done. First, if a job can be assigned to stand-by and if there is a job computer which is not working on another problem, the MC arranges for the job to be started immediately by loading the job program's equipment assignment list into the CX memory and alerting the idle job computer to start work. The MC then drops further consideration of that job and immediately proceeds to the next item in the backlog. Second, when a job computer completes its present work and finds another problem in the queue to start on, it alerts the MC. The master program can accept and act on these alerts at several points—before the beginning of certain specified joints in the program, and at any time when the MC is in a loop waiting for a response to an alert which it has sent out itself. The CX memory is set for the job computer when the MC accepts the alert.

After all the backlog items have been considered in this way, the master program restarts its basic computational cycle by re-examining the contents of the peripheral buffer.

3.2.3. JOB COMPUTERS

Each job computer (JC) is responsible for some master program functions: that is, a part of the master program is stored in each job-computer module (Figure 9).

At the end of a job program, the JC finds the job's record in the program queue and marks it complete. At this point, the JC no longer has any responsibility for that job. The JC then searches the program queue, in order of rank, to find a job which is ready to be executed. If there is no such job, the JC waits for further action from the master computer. If, on the other hand, there is a job in stand-by, the job computer alerts the MC so that the MC can set the CX memory for it. After the alert is issued, the job computer waits for a response. Upon receipt of that response, the job computer starts work on the new job, after noting on the program queue that the job is now in progress.

When a job is interrupted, the JC notes this fact on the job's record in the program queue by marking that job as being back in the stand-by list.

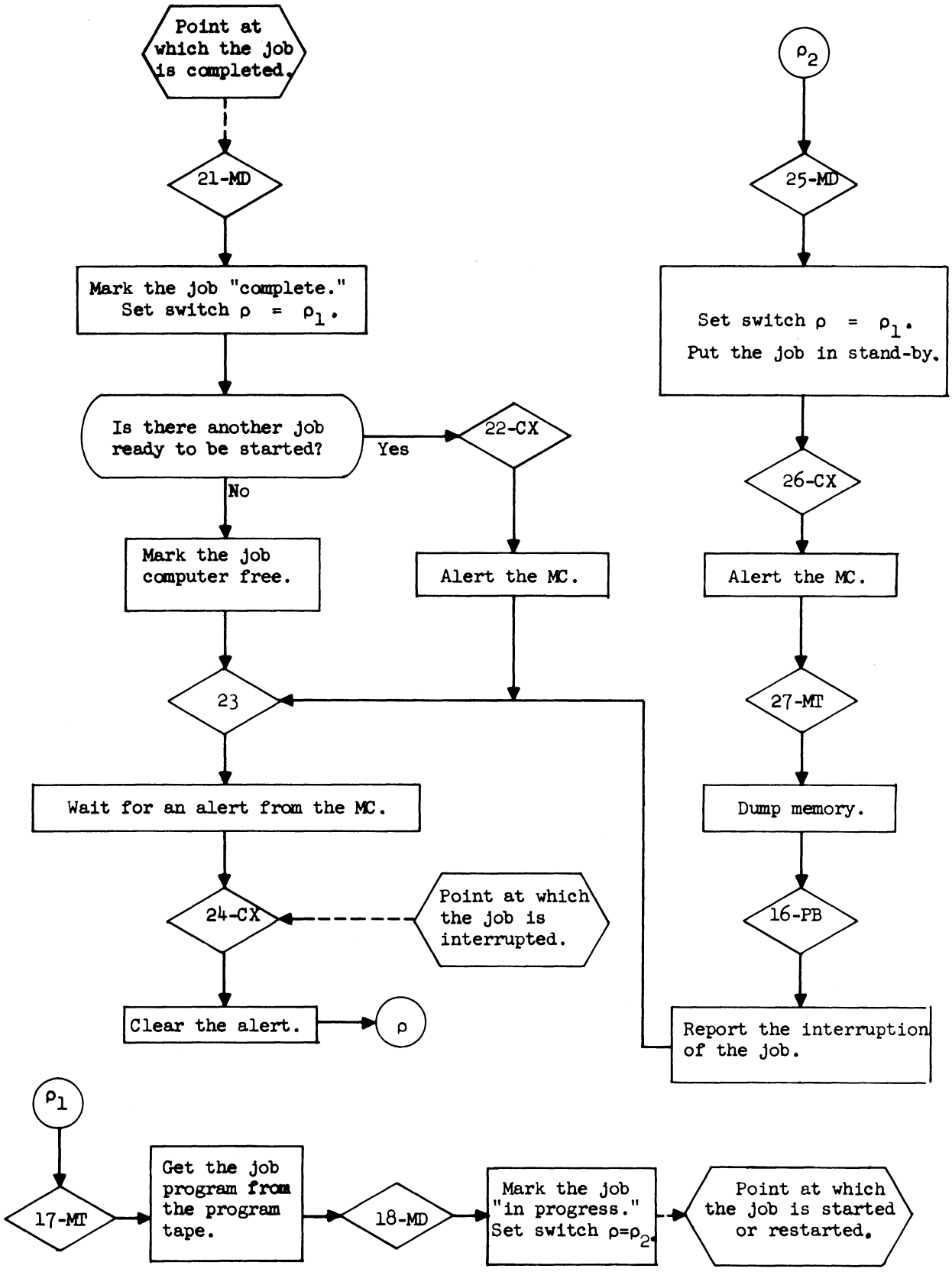

Figure 9. Job computer functions.

Also noted is the exact point at which the job was interrupted. This information, plus a memory dump, permits proper restart procedures. The job computer then issues an alert to the MC to tell it that this action has been taken. The job computer also issues a message to the system operator stating that the job is no longer in progress. Then the JC waits for further action from the MC.

3.3. Mechanics and Techniques

These sections describe in detail the technique by which the simulation program implements the required functions.

3.3.1. SIMULATION PARAMETERS

The simulation program proceeds by means of data which are grouped by function and stored in tables in memory. The data in these tables are completely set by the input cards for the simulation, which are read into the 704 at the start of the program. As the program runs, these data are changed to indicate the status of those aspects of the AN/FSQ-27 that are being modeled.

The first part of the program reads in the input data cards and checks the correctness of the format. When this has been done, one page is printed giving the run number and other incidental information. Then the second part of the program is brought into core memory and the actual simulation begins, using and changing the data in the tables. At the user's option, the first part of the program may be read in at various times to punch out the contents of the simulation tables. These tables are also punched out at the end of the program, thus allowing one to restart the program where a previous run has been stopped or at earlier points, in case of machine difficulties. This input and output is primarily the function of the START subroutine.

The parameters in the simulation tables are given either as constants or as random variables of specified distribution functions (Appendix A.1.2). A brief list of the tables is given in Table I with functional descriptions given throughout this section, and a detailed description in Appendix A.1.3.

3.3.2. THE TIME LOG

The principal mechanism of the simulation program is the time log, a table which consists of entries representing the next occurrences of all the independent activities in the AN/FSQ-27 system. These entries (rows in the TIMELG table) are items of several words each, containing the simulation

TABLE I

SIMULATION TABLES

704 Name	Name	Contents
IGPLOG	Input generating program sequence log	A list of job programs, by type, which are to be input to the AN/FSQ-27 in a prescribed sequence.
IMPLOG	Master program event log	A list of both the controlled modules to which the master computer connects to perform its functions (Figures 8 & 9) and the parameters giving the times to perform these functions.
JRC	Job computer record	The status of each of the computer modules working on job programs.
MAT	Module availability table	The status of each module in the system--whether it is busy or not, and whether it is available or has been assigned to some particular job.
MCT	Module characteristic table	The access time to each type of shared module.
MISC	Miscellaneous array	A collection of assorted variables impossible to describe briefly. The parameter it holds which is of chief concern is the switching time of the central exchange.
PERBUF	Peripheral buffer queue	A list of those jobs which have arrived as input to the AN/FSW-27 but which the master program has not yet started processing.
PURVUE	Purview table	The specifications of each type of job which is input to the system: the length and number of its major joints, its interrupt disposition, its equipment requirements, it priority number, and its arrival date.
QUEUE	Program queue	The status of those jobs which are being processed through the system. After a job is picked up from the peripheral buffer queue, it is placed into the program queue until it is completed and has been reported out to the system operator.
SEQQ	Sequence queue	The status of those IGPLOG sequences which are being processed through the system.
TIMELG	Time log	The over-all status of the simulation. Roughly speaking, the next occurrence of each type of event is listed in the time log.

time and information about the type of activity which that entry represents. The main activities initiated by time log entries are minor joints (for both the job computers and master computer), job-program major points, job-program arrivals, and the end of the simulation run (Figure 10). The job-program minor joint, representing as it does a connection to a shared module, is considered here to be a part of the master program.

The items in the time log are always kept in order by time by the use of the TIMLG subroutine. The simulation program operates by selecting the first entry in the log. Control is then transferred to the subroutine, which performs the simulation function indicated by that entry. This subroutine may change some words in the time log entry, including the simulation time, after which that entry is moved to a new position in the log. In effect, the old item is erased and a new one created. Whenever a new item is inserted into the log, it is placed after all the entries due to occur either earlier or at the same time. This is of particular significance in the modeling of the CX (Section 3.3.7).

A full 704 word is used to hold simulation time. Therefore, the largest time quantity which can be used in the simulation is 2^{35}-1, or 34,359,738,367. If, for example, a run is started at time -(2^{35}-1) microseconds, the program could simulate 19 hours of real time.

3.3.3. THE INPUT GENERATING PROGRAM

Each job program entering the AN/FSQ-27 is considered to be of a given class, or type, p. The characteristics of each job type are specified in the purview, or purvue, table. As each job arrival is simulated, the job request is placed into a queue in the simulated peripheral buffer, where it stays until the master computer removes it for processing. Each new arrival is assigned a consecutive job number k, regardless of the type of job or the method of arrival generation.

There are two methods of generating problem arrivals. In the first, there must be an initial entry in the time log with the job type, and, of course, the time of arrival of that job. Each such entry in the log generates a succeeding entry for that job type, at a time interval given in the purview table (Figure 11). A second method of generating arrivals allows the user to represent related sequences of job programs. Each of these sequences is composed of a set of "events," where an event may be either the execution of a job of a given type, or a time interval (which might be used to simulate off-line activity between computer operations). One can then specify that the initiation of a set of one or more events will occur upon the completion of another set of one or more events. For example, one might want to describe a sequence of job arrivals in the following manner:

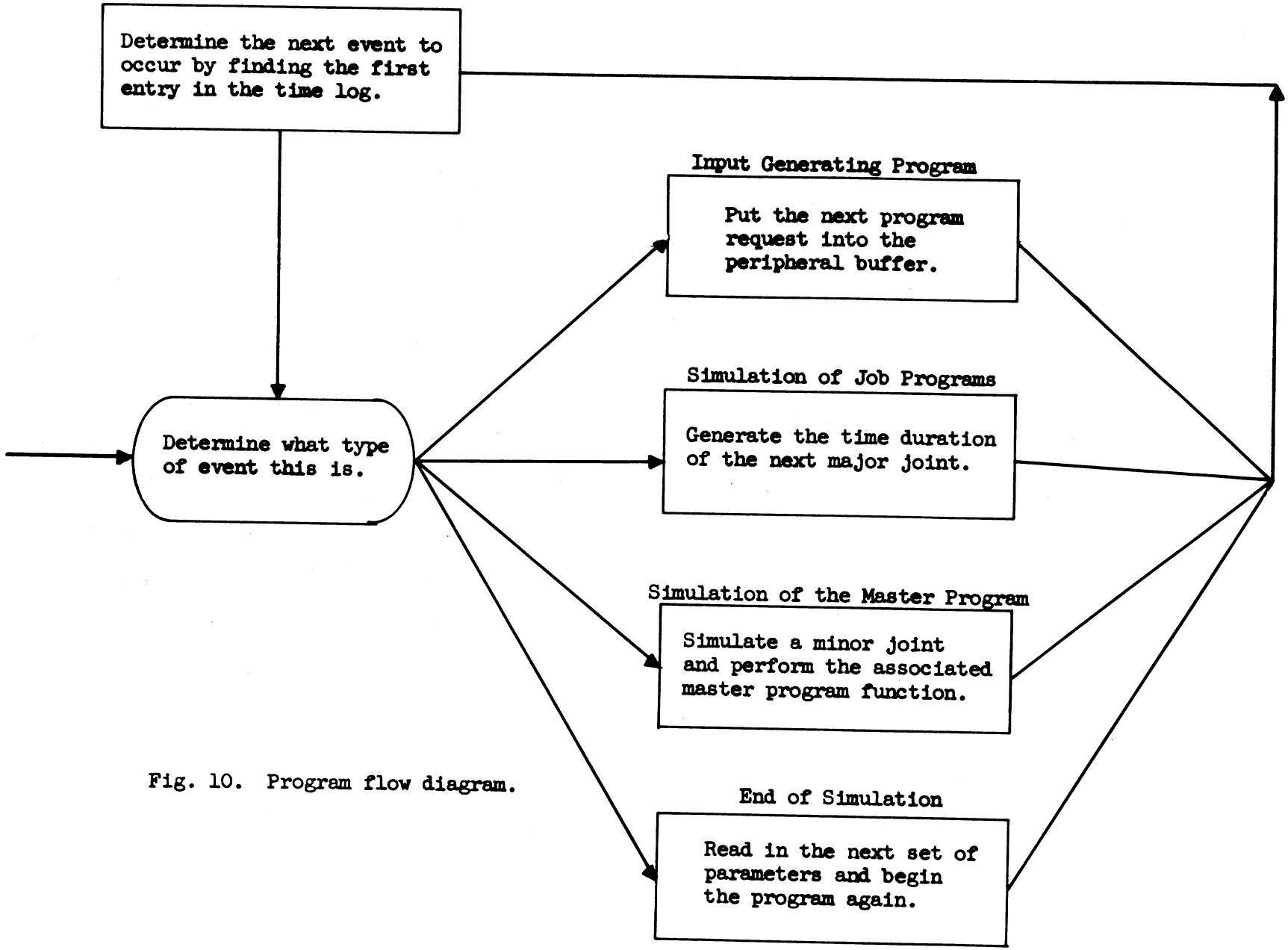

Fig. 10. Program flow diagram.

Figure 10. Program flow diagram.

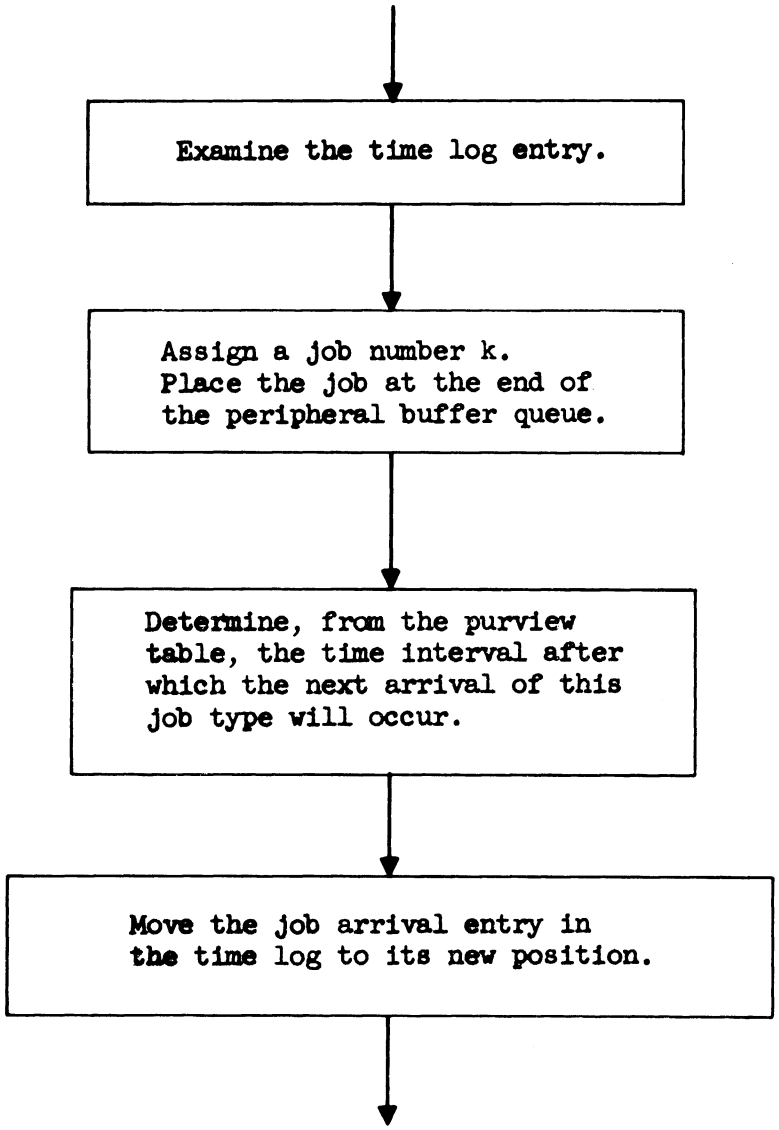

Figure 11. Generating independent job requests.

(a) Problem type 3 is begun.[1]

(b) At the completion of 3, there is to be a time delay of 1,000 seconds.

(c) At the end of that delay, problem type 4 is to be started, and, at the same time, a time interval of 2,000 seconds will be begun.

(d) When both events started in (c) are completed, another arrival of problem type 1 is to be generated.

[1]When we say in this discussion that a job is begun, we mean that an arrival of that type is placed into the PB queue.

Each such sequence is considered to be of a given type r, with the characteristics of each sequence type specified in the IGPLOG table. As in the first method of generating job-program arrivals, the initial time log entry for each sequence generates a succeeding entry, at a time interval given for that sequence in the IGPLOG. As each sequence is initiated, it is assigned a consecutive sequence number L. Using a sequence to describe job-program arrivals may create additional entries in the time log: when an event is initiated which represents a time interval between jobs, entries will be made in the time log, at the time of completion of this event, for each event that follows this time-interval event (Figure 12).

With the first method of arrival generation, the initiation of the next job of that type does not depend on the completion of the preceding job of that type. With the second method, the initiation of the next sequence does not depend on the completion of the previous sequence.

Both these methods of specifying job arrivals may be used during one simulation run, with several different sequences being used and several different problem types arriving independently. The same job type may appear both independently and as parts of sequences. The same job type may appear more than once in the same sequence, and there may be more than one time log entry for the same job or sequence type. Each initial entry in the time log, and the set of subsequent entries it generates, are completely independent from all other entries in the log.

3.3.4. MINOR JOINTS

There is one and only one entry in the time log for each computer module in the system. The job computers perform both major and minor joints; the master computer performs only minor joints. One word in the time log entry for each computer module states whether the next function to be performed by that computer module, starting at the time given in the time log, is to be a minor joint or not. The IMPLOG table lists all the minor joints in the simulation program. Joints 1 through 15 and 29 are master computer functions; 16 through 28 are performed by the job computers.

On appearance of a time log entry calling for a minor joint, the MPMIN subroutine takes the following steps (Figure 13):

(a) The joint to be starting is determined. This is given in the time log entry, except when the master computer has received and is accepting an alert (Section 3.3.6).

(b) The computer module is disconnected from its subordinate module, and CX switching time is added to the present time as given in the time log entry. The time log entry contains the number of

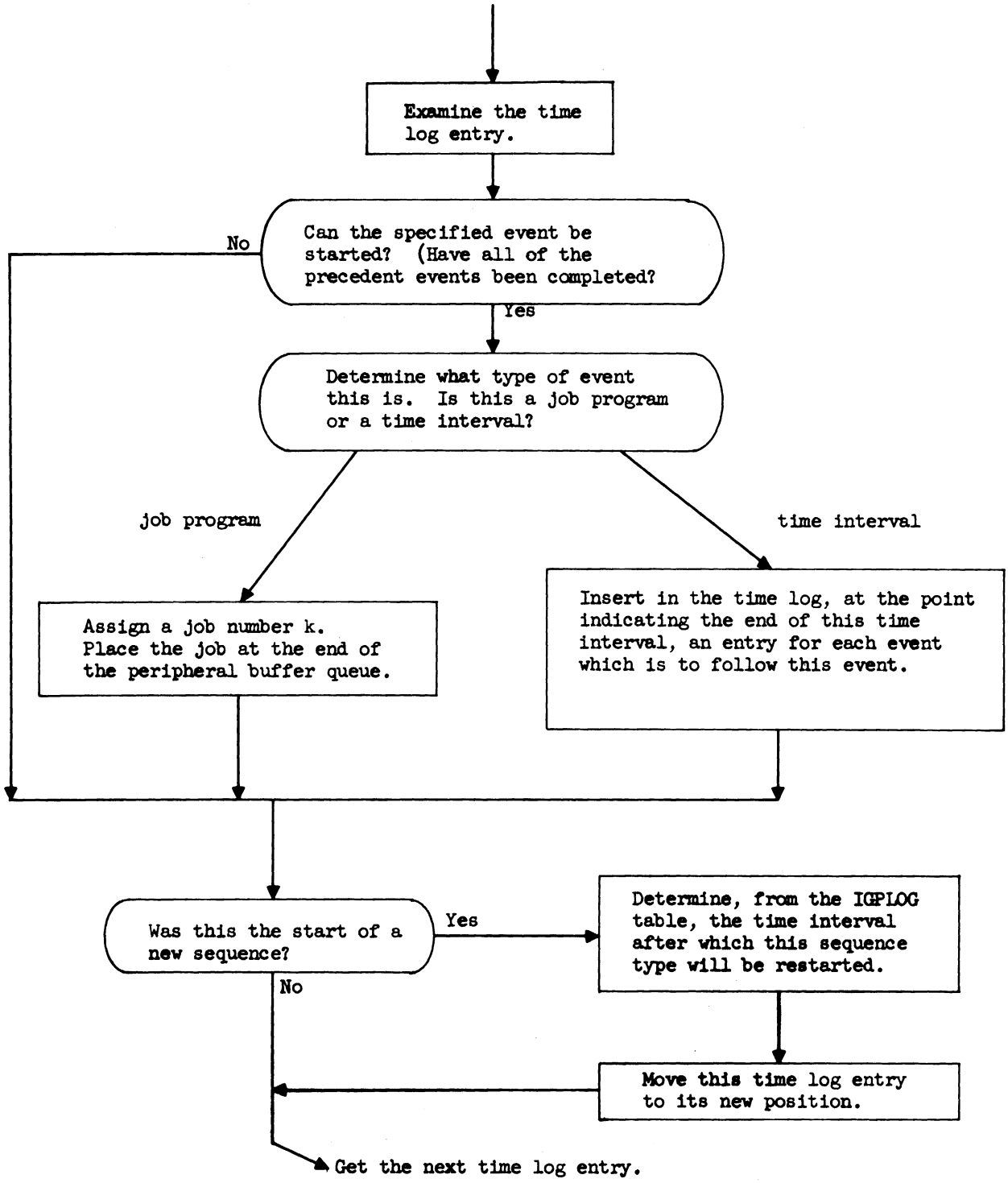

Figure 12. Generating job request by use of sequences.

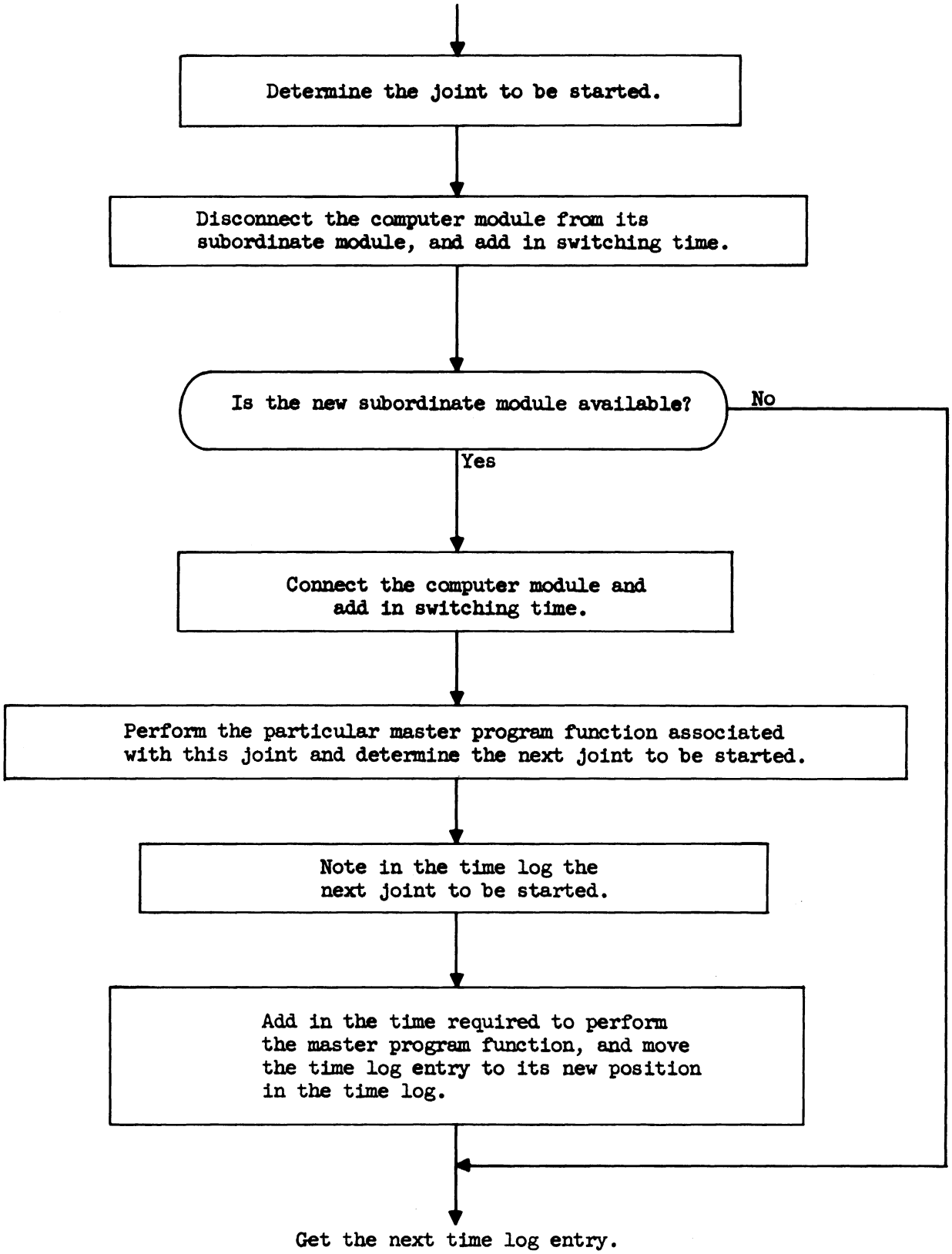

Figure 13. Minor joint action.

the subordinate module to which this computer module is now connected. The MAT table lists all the modules in the system, by type only (i.e., if there are two or more modules of the same type in the system, all are listed in MAT, but nothing differentiates one from another). The disconnection here is modeled by finding any module in MAT of the given type which is marked busy and not disabled, and re-marking that module as free, or available. The appropriate word in the time log item is now set to zero, indicating that the computer module is no longer connected to a subordinate module.

(c) The subordinate module for the next joint, if any, is determined from the IMPLOG, and the MAT is searched to see if any module of this type is available and not disabled. [This searching does not take place if the subordinate module is the central exchange "module" (Section 3.3.7), since this pseudo module is considered always free.] If no such module is free at the moment, the time log is searched to find the time t when the first module of the necessary type will become free. The entry for this joint is then moved in the time log to time t, and, as always, is placed in the time log after all the entries at time t. There is no more activity on this joint at this time.

(d) If the subordinate module is available, it will be marked as busy in MAT and the time log entry for this computer module will be marked to show that the CM is now connected to a subordinate module of the given type. The access time to the subordinate module, computed from the parameters in the MCT table, is added to the present time as given in the time log entry.

(e) Control is now transferred to the particular section of the MPMIN subroutine which performs the simulated master program function (Figures 8 and 9). These program sections also determine the next joint to be executed by this particular computer module. Any action taken at this point which refers to "present time" considers as present time the time that the joint started plus CX switching time plus access time to the subordinate module. This figure is found in the time log entry for the joint.

(f) The general section of the MPMIN subroutine places the next joint indication into the time log entry, adds the time of the joint (as found in the IMPLOG table) to the present time in the time log entry, and enters the TIMLG subroutine to move the entry to its new position in the log.

3.3.5. JOINT NUMBER BOOKKEEPING

At this point it is advisable to digress to introduce the concept of main routines and subroutines. As used in this simulation program with re-

gard to the activities of the AN/FSQ-27 computer modules, a main routine is the principal activity in a computer module, while a subroutine is an activity performed repetitively, from different points in the main routine.

In job computers, the main routines are defined as major joints of job programs, while all master-program activity, or minor joints, performed by the job computers are considered subroutines. The repetitive use of these minor joints is obvious, when one notes that different job programs are run by a given job computer, but the same minor joints are used for each job program to effect its execution.

In the master computer, the main routine consists of those minor joints which perform the regular tasks of accepting new jobs into the system, assigning them equipment, and finally removing them from the system. There is but one subroutine in the master computer, and that is the set of minor joints which is entered when a job computer alerts the master computer (Section 3.3.6).

It is necessary for the simulation program to keep track of a computer module's progress as it goes from main routine to subroutine and back again. Two words in the time log entry—j and j'—accomplish this. j is the number of the next joint in the main routine to be executed; j' is the number of the next joint in the subroutine to be executed. A computer module enters a subroutine by setting j' equal to the number of a particular minor joint which is an entry to a subroutine. As the subroutine progresses from joint to joint, j' is progressively changed, while j is constant. At the end of the subroutine (during its last joint), j' is set equal to zero. This indicates to the simulation program that the next joint to be done by the computer module will be joint j of the main routine. The action of the MPMIN part of the simulation program is always to do the next joint of the subroutine rather than the main routine.

For example, consider a master computer which is about to start joint 1 when it receives and accepts an alert from a job computer. Upon receipt of an alert, the master computer enters joint 11, and performs joints 11, 12, 13, 14, and 15 before returning to the point at which it was interrupted. The sequence of time log entries which occurs will show j and j' changing as follows:

	j	j'
Before the alert is accepted, the time log entry has:	1	0
When the alert is accepted, joint 11 is started and the time log is marked for the next joint:	1	12
At the end of 12, the time log is marked:	1	13
At the end of 13, the time log is marked:	1	14

	j	j'
At the end of 14, the time log is marked:	1	15
At the end of 15, the MC is allowed to return to its basic activity, and the time log has:	1	0
At the end of joint 1, the MC is scheduled to start joint 2:	2	0

As a second example, consider a job computer which is alternating between major joints of a job program and minor joints (joint 28 in the IMPLOG table). In this case the sequences of j and j' in the successive time log entries for the job computer module looks as follows:

When the next joint to be done is joint 28:	1	28
When the next joint is major joint 1:	1	0
When the next joint is minor joint 28:	2	28
When the next joint is major joint 2:	2	0
When the next joint is minor joint 28:	3	28

3.3.6. ALERTS AND INTERRUPTS

Computer modules communicate with each other via a system of alerts, the impulses sent from the MC to a job computer, or vice versa, via the central exchange. The receiving module can interrupt itself to test for and act on an alert at its option, that is, at any time after the alert has been sent. Although this type of action might have many uses when programming for the AN/FSQ-27, the simulation program models only a few particular uses of alerts.

3.3.6.1. Sending Alerts

The sending of an alert is modeled by the execution of a minor joint in which the subordinate module is a pseudo CX module (Section 3.3.7). The conditions under which CM's alert one another can be seen in the general flow diagram in Figures 8 and 9. The master computer alerts a job computer (joint 7) when it wants that job computer to stop whatever it is doing, if anything, and start work on a particular job. The MC also alerts job computers (minor joint 13) when it has set the CX memory so that that job computer can start work on a new problem.

There are also two specific instances in which the job computers alert the master computer. When a JC has finished one job and, on its own, has found another job in the program queue which is ready to be executed, the job computer alerts the MC (minor joint 22) so that the MC can set the CX memory for the job computer. The second instance of a JC's alerting a MC occurs after a JC has been interrupted in the middle of a job by the MC. At this time, the job computer notifies the MC by means of an alert (minor joint 26) that it has interrupted itself and will soon be ready for further word from the MC.

3.3.6.2. Receiving and Acting on Alerts

The simulation program, in modeling the accepting of an alert, is limited by not allowing most minor joints to be interrupted. The two exceptions to this arise when a CM is specifically waiting for an alert. This waiting procedure, employed by both job computers and the master computer, is modeled by having the computer module enter a minor joint with no subordinate module and of infinite duration (minor joints 8 and 23). This procedure is modeled by causing the time log entry for that module to sink to the bottom of the time log list.

When an alert is sent to a CM in this condition, the ALERT subroutine recognizes the situation and immediately arranges for the receiving CM to accept the alert at once. It does this by moving the CM's entry in the time log up to the time at which the alert was sent, and by setting j' (see Section 3.3.5) so that the next joint performed by that CM is the one whose function it is to act on alerts (minor joints 11 and 24).

When an alert is sent to a job computer which is engaged at the time in the execution of a major joint, the following steps are taken to see if and when that JC will allow itself to be interrupted. A time interval is computed from the parameters listed in the purivew table as the interrupt disposition for that particular job type. This time interval is then added to the present time (the time at which the alert was sent). If this computed time of possible interrupt occurs before the end of the major joint (as given in the time log), then that major joint is cut short, by moving the time log entry for the job computer up to the computed time of interrupt, and j' is changed to joint 24, as described in the preceding paragraph.

If an alert is sent to a CM, and the situation is not one of those described above, a marker is put in the time log entry for the module which received the alert. The word "a" in the time log entry, usually zero, is set equal to the time at which the alert was sent. In the case of the master computer, if an alert is sent and the MC is not in joint 8 (i.e., is not in a position to accept the alert immediately), "a" is first checked to see whether or not it is already nonzero. If it is nonzero, indicating the presence of an earlier alert from another job computer, "a" is left at its previous, lower value. This alert indicator "a" is then used as follows:

(a) When a computer module is about to start a minor joint, "a" is checked. If "a" is nonzero and if it is allowable to interrupt that CM before the start of its predesignated minor joint then j' is set to joint 11 or joint 24 as indicated above.[2] Whether or not the CM is allowed to accept an interrupt before the start of a particular minor joint depends on parameters in the IMPLOG, set by the user of the simulation program, which state before which joints alerts will be accepted.

(b) When a job computer is about to start a major joint, "a" is checked. If "a' is nonzero, the time of possible interrupt is computed as described above, adding the interrupt disposition time interval to the present time (or the time at which the alert was sent, whichever is the greater). The present time in this case is not the time the alert was sent but rather the time at which the major joint was scheduled to be started. The length of the next major joint is then computed (Section 3.3.10). If the time of possible interrupt is earlier than the time at which the next major joint would otherwise end, the end of the next joint is set equal to the time of interrupt and j' is set equal to 24.

(c) If an "a" in a time log entry is nonzero, and if the alert is not accepted before or during the next joint, "a" is left as it is and tested again at the beginning of the next joint of that computer module.

After a computer module has received and accepted an alert, it erases the alert signal, i.e., it removes the signal from its input lines. In the simulation program this activity is modeled by the execution of a minor joint (15 and 24) in which the subordinate module is the pseudo CX module. This is not to be confused with the sending of an alert by that CM.

The assumption was made, in this program, that the master computer has only one line on which to receive alert impulses, i.e., that it has no way of knowing, when it receives an alert, which job computer sent that alert. The simulation program solves this problem by assuming that certain information is placed on the master drum, which is a shared subordinate module, by the job computer. This information, contained in the JCR table, is then checked by the master computer upon receipt of the alert impulse to determine which job computers have signaled it and why.

3.3.7. MODELING OF THE CENTRAL EXCHANGE

The central exchange is not modeled precisely in this program. We have assumed that the CX is always available to any computer module which requires

---

[2] A third test is that the time the alert was sent must not be later than the time at which the minor joint is tarting; this test is necessitated by the inaxactitude of the modeling of the CX.

it, thus ignoring the jump commutator actually used in the AN/FSQ-27. At the start of each minor joint, a constant time representing the CX access time and set by the simulation user with an input parameter is added to the present time as given in the current time log entry. This represents the time needed to connect with a subordinate module and/or disconnect from it.

To simulate the sending and clearing of alerts (Section 3.3.6), the artifice of a pseudo CX module is employed—a subordinate module to which a CM connects to send or clear an alert. Actually, this is just a matter of notation to aid in understanding the progress of events, since the minor joints which "connect" to the CX module act exactly as if there were no subordinate module at all, and the entire concept could be dropped out of the program.

The pseudo CX memory module, however, has very real use. To simulate the master computer's loading the CX memory, the device is used of a minor joint (12) which has the CX memory module as a subordinate module. The access time to this module, as listed in the module characteristic table MCT, should be set equal to the time needed by the master computer to load the CX memory. This pseudo access time then determines the time t by which time the memory will have been loaded. At joint 12, the time log is examined and all entries occurring before time t are moved back to time t, thus modeling the fact that no access to the CX is possible during this time period. For ease of programming, there is a slight inaccuracy in that all entries in the time log are so moved, whether or not they represent a future connection to the CX. (IGP entries, for example do not represent such a connection.)

3.3.8. MASTER COMPUTER JOINTS

The master computer executes a series of minor joints as shown in Figure 8. Each minor joint involves the set of general operations described in Section 3.3.4. In addition, certain specific functions are performed in each joint. The following list of activities complements Sections 3.2.2 and 3.2.3.

(1) If there is a job in the peripheral buffer queue, the job is removed from the queue and placed into temporary storage. If there is more than one job in the queue, the first (earliest) is removed.

(2) This joint simulates reading the new job's purview table from its place of storage (such as a magnetic tape file). The simulation program performs no functions.

(3) The new job is placed on the program queue, in the backlog.

(4) If there are jobs on the program queue that have been completed, the master program removes the job with the highest rank, and puts the modules that were used for that job back into the list of available modules. If there is no such job, and if the switch is not set to go back to joint 1 (see Figure 8), then the master program tries to assign equipment to jobs on the program queue in the backlog. In trying to assign modules (including job computers) to various jobs, the master computer simply scans the list of modules in the MAT table from top to bottom and takes the lowest numbered job computer available.

(5) The IGP subroutine is entered here, in case the newly completed job is part of a sequence.

(6) This completes the removal of the completed job from the queue.

(7) This joint alerts a job computer which is executing a job. The function is also performed here of reordering the jobs in the queue, if necessary, because of any change of rank caused by the master computer's changing the priority number of a job after interrupting its execution.

(8) In this joint, the master computer is waiting for an alert from a job computer.

(9) and (10) These joints are entered by the MC whenever a job computer has signaled the master computer that it has interrupted itself.

(11) This joint is entered by the MC when it is interrupted. If a job computer has signaled the master computer to set the CX memory, this procedure is initiated and the record of this request, a part of the JCR table, is erased by the master computer. If the JC that sent the alert was the one that the MC had previously signaled, that JC is also marked in the JCR table as being free.

(12) The CX memory is loaded (Section 3.3.7).

(13) The job computer is alerted.

(14) The job computer is marked as busy.

(15) The alert signal to the master computer is erased, by setting "a" equal to zero (Section 3.3.6).

It should be noted that the master program will probably spend the major part of its time oscillating between joints 1 and 4, essentially doing nothing. The user of the simulation has the option of speeding up the operation of the program by effectively removing the MC from the time log when there is nothing for it to do (see Appendix, Section A.1.1).

3.3.9. JOB COMPUTER FUNCTIONS

The execution of a job program is expressed as an alternation of major and minor joints, beginning with a minor joint. The number of major joints and the length of each are peculiar to each problem type and are defined in the purview table. However, the minor joint is the same for all problem types, both in length and the subordinate module. Each job computer must also perform some of the executive functions of the system, using minor joints, to administer the succession of job programs. The minor joints listed below and depicted in Figure 9 are performed by all job computers in the same manner, regardless of the particular job program being processed.

(16) The job computer marks itself as busy.

(17) No simulation function is performed.

(18) No simulation function is performed.

(19) The words in the time log entry for this job computer are set to the proper values for the new job. An interrupted job will be restarted at the major joint in which it was interrupted. That joint will end at a time interval after the present time which is equal to that part of the major joint that was not completed plus the restart penalty, if any.

(20) This is a simulation mechanism.

(21) The job is marked as completed on the program queue and the queue is searched for a job which is ready to be executed. If there is none, the job computer marks itself as free.

(22) The master computer is alerted.

(23) The job computer waits for an alert from the master computer.

(24) This joint is entered when the job computer is interrupted in the middle of a job.

(25) The amount of time left in the unfinished major joint is noted in the program queue, so that the job can be restarted properly.

(26) The master computer is alerted.

(27) No simulation function is performed.

(28) This is the minor joint for all job programs. No simulation function is performed.

(29) This joint is a simulation mechanism.

3.3.10. MAJOR JOINTS

There are two sets of parameters in the purview table for each job type, giving the number of major joints and the length of each major joint for any job of that type. When a job first enters the system, the number of major joints for that particular job is generated from the input parameters. This number is unchanged for that job until the job is completed. At the time of starting each major joint, the length of that particular joint is generated.

If a job is interrupted in the middle of a major joint, the amount of time t left in that joint (not executed because of the interruption) is saved. When the job is restarted, it begins with a (partial) major joint of duration t, plus whatever additional time interval is assessed as a restart penalty (as given in the purivew table) if this is being used, at the option of the simulation user. The computation of joint lengths is done in the subroutine JPMAJ. Complete details of the simulation program are given in Appendix A.

4. QUEUEING THEORY STUDIES

4.1. Introduction

It became apparent during the contract period that a simulation approach must be augmented by an understanding of simplified analytical models. The mathematical theory of queues, i.e., the study of waiting lines in service facilities, furnishes a very useful analytical basis for study of simple models of the polymorphic system. For example, in the AN/FSQ-27 system one of the major functions of the master computer is the assignment of waiting job requests to the job computers, the rules of assignment constituting a queue discipline. Hence, the master computer may be modeled as the maintainer of queue discipline. The job computers, augmented by the required numbers and types of subordinate modules, may be considered as single service channels. The resulting model of the AN/FSQ-27, then, is a multiple-service-channel facility with a single waiting line. If one also reduces the characteristics of the job programs to a distribution on the service time in a channel (job computation time) and a distribution on time between arrivals of job requests, the model becomes one of the simplest to analyze in terms of queueing theory. The analogies between the multiple-channel service facility and the polymorphic computer are summarized in Figure 14. Job re-

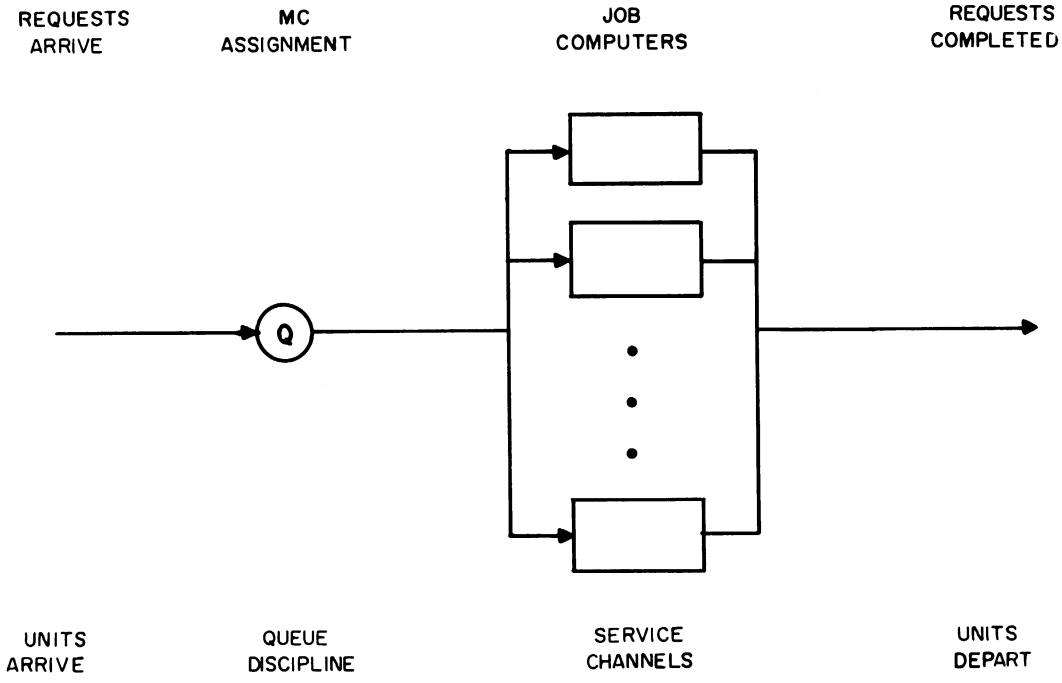

Figure 14. Queueing theory model for a polymorphic computer.

quests (units) arrive at the system queue, i.e., the backlog and stand-by lists, and are subsequently assigned to the job computers (service channels) as they become available.

This model has been used for two purposes: to determine the length of the simulation run required to obtain statistically significant results, and to gain insight into some of the basic problems encountered in a polymorphic data-processing system. The purpose of this section is to present a logical development of queueing theory and to apply the results to the AN/FSQ-27 as modeled in Figure 14. The application of queueing theory to the problems associated with the utility of the simulation are covered in Section 5.

The essential features of the system can be reduced to a waiting-line problem. That is, the measures of system performance of particular interest are the statistics describing the number of requests in the system and the waiting and throughput-time distributions. The emphasis upon the processing time per job request is justified because the system must keep up with the raw data received from the satellite readout stations.

Sections 4.3 and 4.4 are devoted to the single- and multiple-channel service facility with exponential arrival and service-time distributions. The direct implications of Section 4.3 for the polymorphic computer are summarized in Section 4.4. Sections 4.5 and 4.6 show particular applications of queueing theory to the storage saturation and shared module queueing problems. Finally, the effect of arbitrary arrival and service-time distributions are discussed in Section 4.7.

4.2. Fundamentals of Queueing Theory[3]

Basic to the remainder of our queueing theory studies is an understanding of the equations of detailed balance for single- and multiple-channel systems with exponential service-time and arrival-time distributions. The approach taken in the theory of queues is to consider a system, at any instant of time, as being in one of a number of possible states. For example, in the simplest queueing problem, each state represents a certain number of units in the system. Knowing the probability of transition between states, one can write a set of linear, constant-coefficient, first-order differential equations in terms of the state probabilities.

If the concept of system states is extended to include the division of a given channel into series or parallel phases, as opposed to the number of units in the system, the arrival and service-time distributions can be approximated by a class or Erlang and/or Hyper-Exponential distributions.

[3]The best single reference for this section is Morse (Ref. 1).

The equations of detailed balance for a one- and two-channel system are discussed in Section 4.2.1. In Section 4.2.2 the Erlang and Hyper-Exponential distributions are introduced and the technique for writing the equations of detailed balance when phases are introduced is illustrated by a single-channel system with an Erlang Type Two or Hyper-Exponential service-time distribution. The only queue discipline considered is first come, first served.

4.2.1. EQUATIONS OF DETAILED BALANCE

A single- or multiple-service facility is characterized by the number of allowed states and the probability of being in a particular state $P_n(t)$. The state of a system is, by definition, equal to the number of units in the system waiting and/or in the service channels. If length of the queue is finite, then the number of allowed states N is determined by the number of channels M plus the maximum number of units allowed in the waiting line. The system has an infinite number of states when the waiting-line length is unbounded.

The equations of detailed balance are stated in terms of the state probabilities, $P_n(t)$. The t dependence indicates that this probability can, under nonsteady-state conditions, be a function of time. A knowledge of the state probabilities is fundamental to any consideration of system performance, since many statistics, e.g., the mean number of units in the system, are a function of the state probabilities. The analogies between the service-facility problem and the AN/FSQ-27 will be discussed as the particular queueing problems are considered. However, it is apparent that, for the simplest system representation, the units can be considered as requests for computation, and service channels as analogous to job computers.

Now, changes of state result from either the arrival of units at the system input or the completion of services. Consider the transitions to the n^{th} state shown in Figure 15. The probability of the system's being in state n at time t+dt is equal to the sum of the transition probabilities to the n^{th} state plus the probability of remaining in the n^{th} state, all weighted by the probability of being in the source state.

$$P_n(t+dt) = T_{n-1,n}(dt)P_{n-1}(t) + T_{n,n}(dt)P_n(t) + T_{n+1,n}(dt)P_{n+1}(t) \qquad (1)$$

where:

$T_{i,n}(dt)$ = the transition probability from the i^{th} state to the n^{th} state during the interval dt, $i \neq n$.

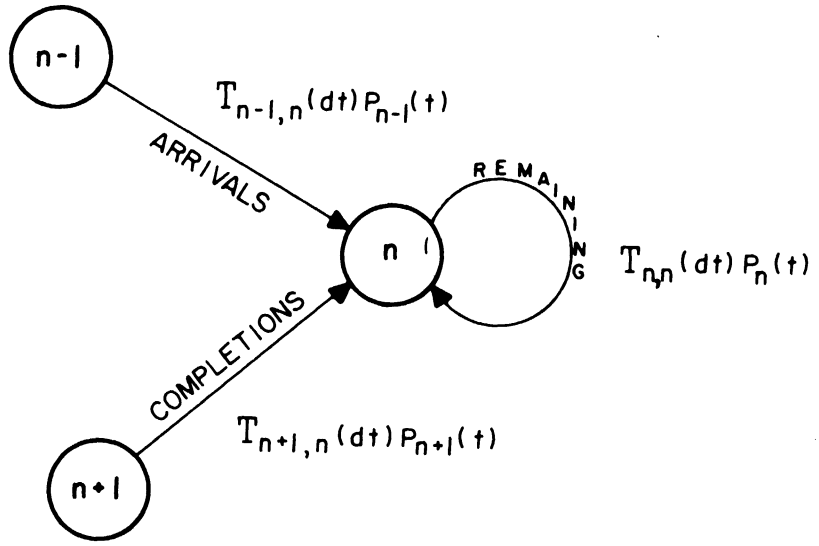

Figure 15. Transitions to the n^{th} state.

$T_{n,n}(dt)$ = the transition probability of remaining in the n^{th} state during the interval dt.

In general, the transition probabilities are functions of the time interval since the last change of state. That is, $T_{n-1,n}(dt)$ is dependent upon the time since the last unit arrival at the system input. Only when the arrival and service distributions are exponentially distributed can this time dependence be eliminated.

Since the number of exponentially distributed events in a given interval are Poisson distributed, the probability of n transitions in time dt is given by

$$T_n(dt) = \frac{(\lambda dt)^n}{n!} e^{-\lambda dt}$$

$$T_n(dt) \cong \frac{(\lambda dt)^n}{n!} \qquad (2)$$

where

n = the number of events in the interval dt;

λ = average rate of events per unit time.

The probability of more than one event is proportional to dt raised to higher powers and, as a result, the probability of more than one event in the interval dt can be neglected. Therefore it is necessary only to consider transitions from adjacent states. The transition probabilities can be shown to be given by

$$\begin{aligned} T_{n-1,n}(dt) &= A_1(dt) \\ T_{n+1,n}(dt) &= S_1(dt) \\ T_{n,n}(dt) &= A_0(dt)S_0(dt) \end{aligned} \qquad (3)$$

where:

$A_1(dt)$ = λdt, the probability of one arrival;

$S_1(dt)$ = μdt, the probability of one service completion for a single channel;

$S_1(dt)$ = $n\mu dt$, $n < M$

$S_1(dt)$ = $M\mu dt$, $n \geq M$

} the probability of one service completion for a multiple-channel system;

$A_0(dt)$ = $1-A_1(dt)$, the probability of no arrival;

$S_0(dt)$ = $1-S_1(dt)$, the probability of no service completions;

λ = $1/T_a$, average arrival rate;

μ = $1/T_s$, average service rate for a single channel;

T_a = average interval between arrivals;

T_s = average interval between service completions for a single channel;

n = the system state;

M = number of channels in the system;

ρ = λ/μ for a single channel system; and

ρ = $\lambda/M\mu$ for a multiple channel system.

When the system has more than one channel, the probability of one completion is equal to the sum of the probabilities that any filled channel will have a service completion, as the channels are assumed to be independent. Again the probability of more than one completion is negligible. If that system is in a state n which is equal to or greater than the number of channels in the system, $S_1(dt)$ is determined by the number of channels M in the system. Otherwise, $S_1(dt)$ is determined by the number of channels n which are filled. Notice that the probability of remaining in the n^{th} state is equal to the joint probability that there will be no arrivals and no service completions during the interval dt.

The parameter ρ is referred to in the literature as the utilization factor. This however, is true only when the queue is unbounded (see Section 4.3.1.2). When the queue is finite, the system utilization is approximately equal to ρ for small ρ and a large number of allowable states (N). It can be shown, by finding the expected value of the arrival and service times, that λ and μ are, respectively, equal to the reciprocal of the average interval between arrivals T_a and the average service time for a single channel T_s. It is certainly reasonable that the ratio of the arrival rate and the average service (for a channel constantly busy) would be the utilization factor for an infinite queue system. For a multiple-channel system, the equivalent service rate for the entire system (all channels again continuously busy) is equal to $M\mu$. Thus, for a single-channel system, ρ is equal to λ/μ while for a multiple-channel system ρ is equal to $\lambda/M\mu$.

Having specified the equations for the transition probability, we can write down the equations of detailed balance for some typical problems. First the equations for a single channel, infinite and finite queue, will be considered. The weighted transition probabilities are shown in Figure 16. Considering the 0^{th} state first, Eq. (1) becomes

$$P_0(t+dt) = T_{-1,0}(dt)P_{-1}(t) + T_{0,0}(dt)P_0(t) + T_{1,0}(dt)P_1(t) \quad (4)$$

where:

$T_{-1,0}(dt)P_{-1}(t)$ is omitted

$T_{0,0}(dt) = [1 - A_1(dt)]S_0(dt) = 1 - \lambda dt$

$T_{1,0}(dt) = S_1(dt) = \mu dt.$

Notice that the probability of no service completion is equal to one, as the system is already in state zero. Thus Eq. (4) becomes

$$P_0(t) + dP_0(t) = (1-\lambda dt)P_0(t) + (\mu dt)P_1(t) \quad (5)$$

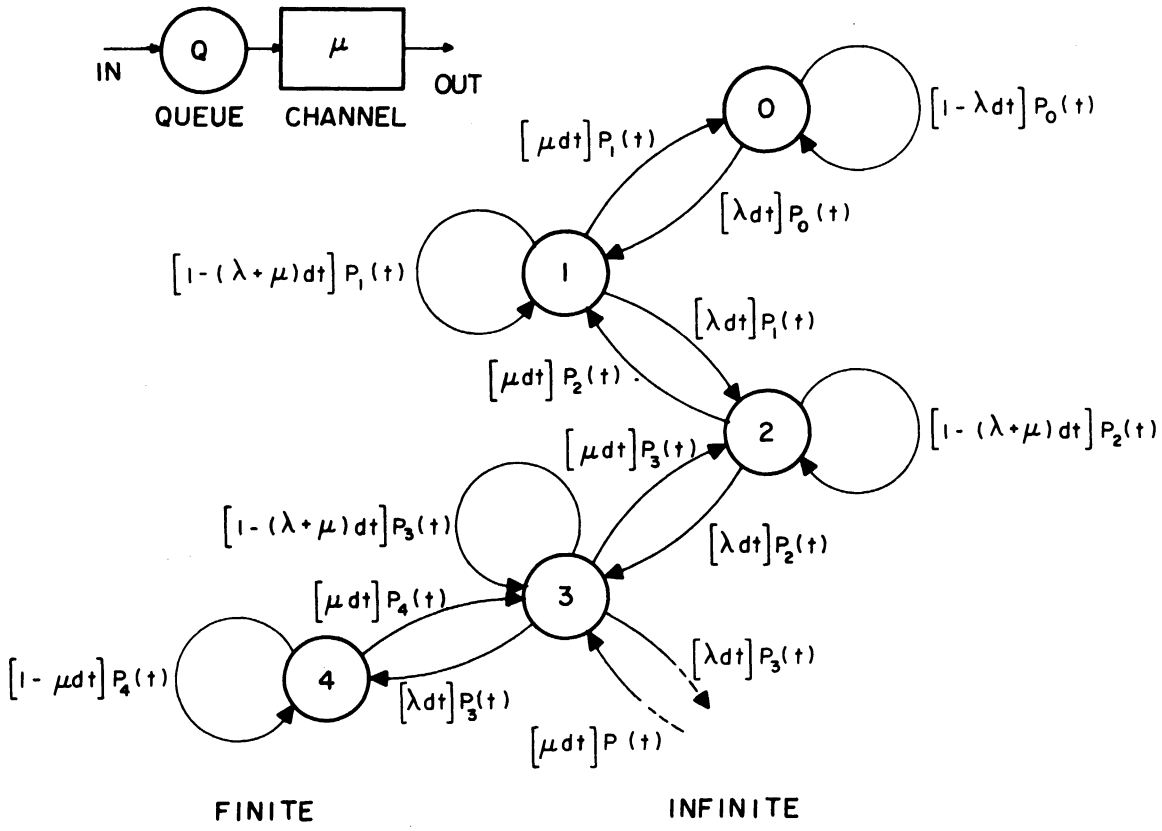

Figure 16. Transitions for a single-channel system.

For the remainder of the states for the infinite queue case, the n^{th} equation of detailed balance is

$$P_n(t+dt) = (\lambda dt)P_{n-1}(t) + [1 - (\lambda+\mu)dt]P_n(t) + (\mu dt)P_{n+1}(t) \quad (6)$$

$n > 0$ for N infinite

$0 < n < N$ for N finite.

For the N^{th} state the required equation is given by

$$P_N(t+dt) = (\lambda dt)P_{N-1}(t) + T_{N,N}(dt)P_N(dt) \quad (7)$$

where:

$$T_{N,N}(dt) = A_0(dt)[1 - S_1(dt)] = 1 - \mu dt.$$

$A_0(dt)$ is equal to one because the system is already in the highest allowed state. That is, it is certain that there will not be any unit arrivals during the interval dt, because any unit which would normally enter the queue will, in this case, go away since the queue is already at its maximum allowable length. The following set of linear, first-order, constant-coefficient, differential equations is obtained for a finite queue of maximum length 3, by dividing Eqs. (5), (6), and (7) by dt:

$$\begin{aligned}
\dot{P}_0(t) &= -\lambda P_0(t) + \mu P_1(t) \\
\dot{P}_1(t) &= \lambda P_0(t) - (\lambda+\mu) P_1(t) + \mu P_2(t) \\
\dot{P}_2(t) &= \lambda P_1(t) - (\lambda+\mu) P_2(t) + \mu P_3(t) \\
\dot{P}_3(t) &= \lambda P_2(t) - (\lambda+\mu) P_3(t) + \mu P_4(t) \\
\dot{P}_4(t) &= \lambda P_3(t) - \mu P_4(t)
\end{aligned} \tag{8}$$

If the queue is infinite, the number of equations is also infinite. In this case, Eq. (7) is not considered.

When one considers a multiple-channel system, the transitions, $T_{n+1,n}$, are modified in accordance with the number of occupied channels. The transitions for a two-channel system are shown in Figure 17. The derivation of the equations of detailed balance is essentially identical to that of those of the single channel, and therefore will not be repeated. For a finite queue (N = 4) the system of equations is given by

$$\begin{aligned}
\dot{P}_0(t) &= -\lambda P_0(t) + \mu P_1(t) \\
\dot{P}_1(t) &= \lambda P_0(t) - (\lambda+\mu) P_1(t) + 2\mu P_2(t) \\
\dot{P}_2(t) &= \lambda P_1(t) - (\lambda+2\mu) P_2(t) + 2\mu P_3(t) \\
\dot{P}_3(t) &= \lambda P_2(t) - (\lambda+2\mu) P_3(t) + 2\mu P_4(t) \\
\dot{P}_4(t) &= \lambda P_3(t) - 2\mu P_4(t)
\end{aligned} \tag{9}$$

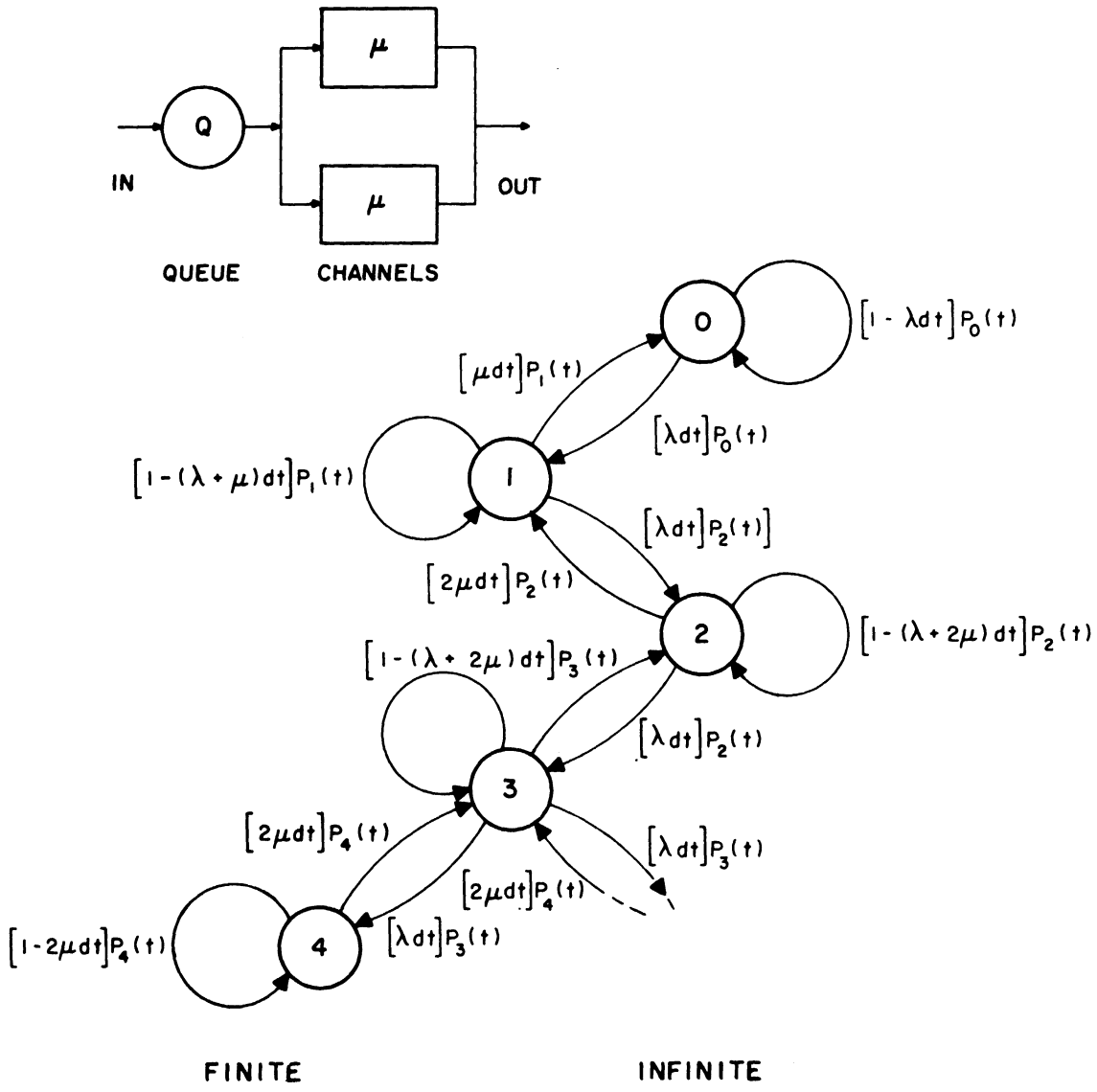

Figure 17. Transitions for a two-channel system.

If the independent variable t is scaled according to $t' = \mu t$, if λ is written in terms of the parameter ρ, and if all equations are divided through by μ, the above equations of detailed balance can be written as

$$
\begin{aligned}
\dot{P}_0(t) &= -2\rho P_0(t') + P_1(t') \\
\dot{P}_1(t) &= 2\rho P_0(t') - (2\rho+1)P_1(t') + 2P_2(t') \\
\dot{P}_2(t) &= 2\rho P_1(t') - (2\rho+2)P_2(t') + 2P_3(t') \\
\dot{P}_3(t) &= 2\rho P_2(t') - (2\rho+2)P_3(t') + 2P_4(t') \\
\dot{P}_4(t) &= 2\rho P_3(t') - 2P_4(t')
\end{aligned}
\qquad (10)
$$

Observe that t' (i.e., $\mu = 1/T_s$, therefore $t' = t/T_s$) is equal to t normalized to the mean service time T_s of a single service channel and, as a result, all solutions are stated in terms of the mean service time for a single channel. In the remaining equations t' will be replaced by t to simplify the notation.

The introduction of a matrix notation can simplify the notation to a considerable extent. For example, Eqs. (10) can become

$$
\dot{P}] = [U]\,P] \qquad (11)
$$

where:

$$
\dot{P}(t)] = \begin{bmatrix} \dot{P}_0(t) \\ \dot{P}_1(t) \\ \dot{P}_2(t) \\ \dot{P}_3(t) \\ \dot{P}_4(t) \end{bmatrix}
\qquad
P(t)] = \begin{bmatrix} P_0(t) \\ P_1(t) \\ P_2(t) \\ P_3(t) \\ P_4(t) \end{bmatrix}
$$

$$
[U] = \begin{bmatrix}
-2\rho & 1 & & & \\
2\rho & -(2\rho+1) & 2 & & \\
 & 2\rho & -(2\rho+2) & 2 & \\
 & & 2\rho & -(2\rho+2) & 2 \\
 & & & 2\rho & -2
\end{bmatrix}
$$

$\dot{P}]$ and $P]$ are column vectors and [U] is a square matrix with constant coefficients. The elements of the [U] matrix are the transition probabilities for the system. Therefore [U] will be referred to as the transition matrix.

4.2.2. CASCADING AND PARALLELING OF EXPONENTIAL PHASES

It was pointed out in the previous section that exponential arrival and service-time distributions given rise to a system of linear, constant-coefficient, differential equations for the state probabilities. However, the arrival and service-time distributions to be studied are often nonexponential. In this case, the above-mentioned method of obtaining the equations of detailed balance is no longer adequate because the transition probabilities are a function of the time since the last change of state (Ref. 1). Nevertheless, there are two types of nonexponential distributions which may provide satisfactory approximations to the actual distributions and have the advantage that the equations can be written with constant coefficients: the Erlang and Hyper-Exponential distributions. The derivation of these distributions and the equations of detailed balance obtained when the service-time distribution is either Erlang or Hyper-Exponential and the arrival-time distribution is exponential are discussed in this section. A similar analysis holds when the arrival-interval distribution is either Erlang or Hyper-Exponential. These cases are extensively discussed in Ref. 1.

4.2.2.1. Erlang Service-Time Distribution

If a service channel is considered as a cascading of exponential phases, the service-time distribution for the over-all channel is of the Erlang type. A unit being served must pass through every phase in the channel before another unit may enter service.

Assuming a cascade of two exponential phases (each having an average service time equal to one-half of the service time for the entire channel), the probability that service for a single unit will be completed in time t+dt is given by

$$s(t)dt = \int_0^t [2\mu e^{-2\mu y} dy][2\mu e^{-2\mu(t-y)} dt] \qquad (12)$$

The first term in the integral is the probability that service will be completed by the first phase by time y, while the second term is the probability that the second phase of service will be completed in the remaining time, t-y. To consider all combinations of y and t-y which will sum to the time of interest t, the above equation must be integrated with respect to y. Thus

$$s(t)dt = 2\mu(2\mu t)e^{-2\mu t}dt \qquad (13)$$

where:

$s(t)$ = the probability density for an Erlang Type Two distribution; and

2μ = the service rate for a single phase.

The class of distributions generated in this way are "less random" in the sense that the variance of the new distribution is less than that for the exponential distribution. Erlang density functions of type k are obtained by higher-order convolutions of the basic exponential density functions. The general form is

$$s(t)dt = \frac{(k\mu)^k}{(k-1)!} t^{k-1} e^{-k\mu t}dt \qquad (14)$$

The equations of detailed balance will now be considered for a single-channel system with an exponential arrival distribution and an Erlang Type Two service-time distribution. The major modification, with respect to the exponential facilities discussed in the previous section, is that the states of the system are no longer defined simply by the number of units in the system, but must include a consideration of the two phases making up the service channel. Consider the transitions shown in Figure 18. Each state, with the exception of the zero state, is described by two subscripts. The first indicates the number of units in the system, n, and the second subscript, s, describes which phase of the service channel is occupied. The units enter from the queue into the s = 2 phase, with service rate 2μ, and then progress into the s = 1 phase. Notice that transition from state n,1 to n+1,2 is not allowed. That is, arrivals cause states to change from 1,1 to 2,1 or 1,2 to 2,2 and service phase completions allow transitions from 2,1 to 1,2 (the last phase completes its service and a unit that was waiting enters the first phase). The transitions in Figure 18 illustrate that a unit is either in the first or the last phase, but not both, because the two phases define the one service channel. A unit enters the channel only when the previous unit has completed the last phase of service. Finally, the probability of remaining in the state n,s is determined by the probability of no arrivals during the interval dt and the probability of no service completions by a service phase. The equations of detailed balance follow from Eq. (1). The first equation of the set is for state zero. The remaining equations follow the sequence 1,1; 1,2; 2,1; 2,2; etc.

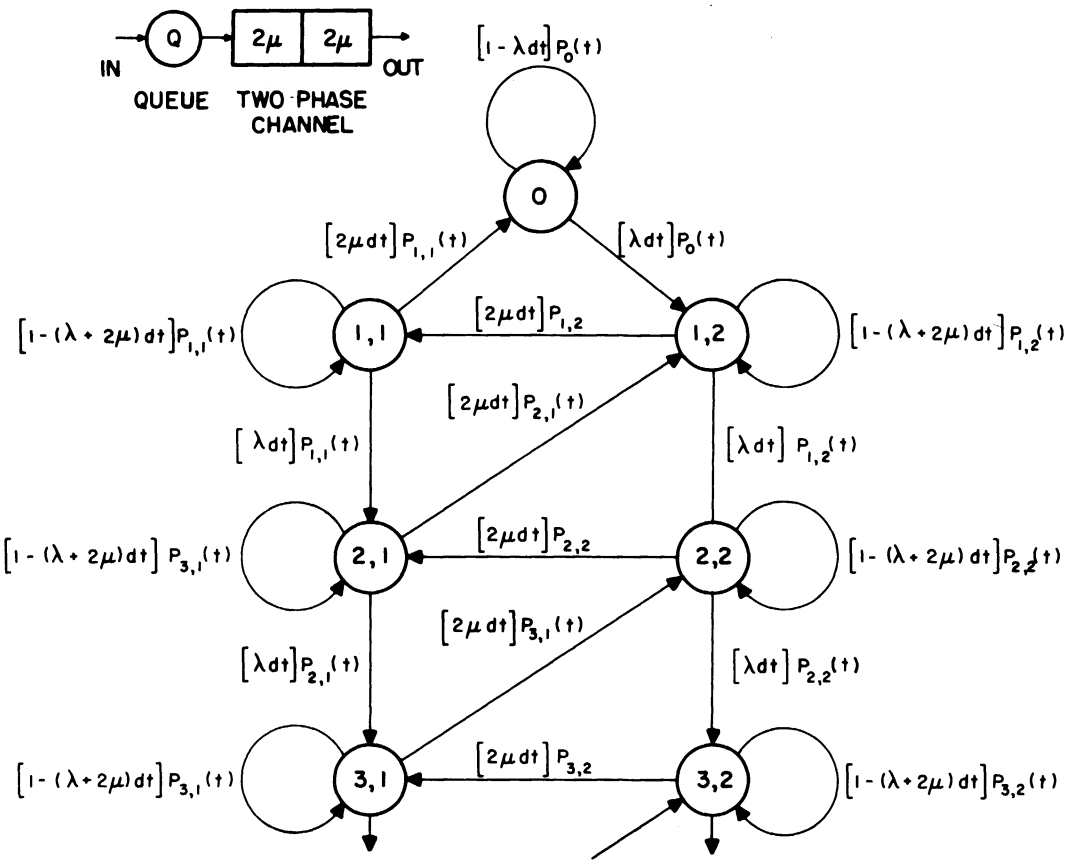

Figure 18. Transient for a single-channel system. Erlang Type Two service distribution.

$$
\begin{aligned}
\dot{P}_0(t) &= -\lambda P_0(t) + 2\mu P_{1,1}(t) \\
\dot{P}_{1,1}(t) &= -(\lambda+2\mu) P_{1,1}(t) + 2\mu P_{1,2}(t) \\
\dot{P}_{1,2}(t) &= \lambda P_0(t) - (\lambda+2\mu) P_{1,2}(t) + 2\mu P_{2,1}(t) \\
\dot{P}_{2,1}(t) &= \lambda P_{1,1}(t) - (\lambda+2\mu) P_{2,1}(t) + 2\mu P_{2,2}(t) \\
\dot{P}_{2,2}(t) &= \lambda P_{1,2}(t) - (\lambda+2\mu) P_{2,2}(t) + 2\mu P_{3,1}(t) \\
\dot{P}_{3,1}(t) &= \lambda P_{2,1}(t) - (\lambda+2\mu) P_{3,1}(t) + 2\mu P_{3,2}(t) \\
\dot{P}_{3,2}(t) &= \lambda P_{2,2}(t) - (\lambda+2\mu) P_{3,2}(t) + 2\mu P_{4,1}(t) \\
&\vdots
\end{aligned}
\qquad (15)
$$

After the solutions for the above set of equations have been obtained, the probability of n units in the system is found by summing the probabilities for the states n,1 and n,2.

4.2.2.2. Hyper-Exponential Service-Time Distribution

The Hyper-Exponential service-time distribution arises when each service channel is considered as being in reality, two parallel exponential phases with different rates. Each unit which enters service chooses one or the other phase. A unit with probability σ will enter a phase with mean rate $2\sigma\mu$, or, with probability $1-\sigma$, will enter a phase with rate $2(1-\sigma)\mu$. Service of the unit is completed when the service time in the selected phase is completed. In addition, only one of the two phases may be occupied at any given time. The resulting probability distribution on service time for a single unit is

$$s(t)dt = 2\sigma^2\mu e^{-2\sigma\mu t}dt + 2(1-\sigma)^2\mu e^{-2(1-\sigma)\mu t}dt \qquad (16)$$

where:

σ = the probability of entering the fast phase; and

μ = the over-all service rate for the channel.

This distribution has a greater variance than the exponential distribution.

The equations of detailed balance are derived in a manner similar to those for the Erlang distribution. The state of the system is specified by the total number of units present, n, and the service phase which is occupied in the single-channel case. The latter is characterized by a second subscript s = 1 when the phase with rate $2\sigma\mu$ is occupied (otherwise s = 2). The equations are

$$\begin{aligned}
\dot{P}_o &= -\lambda P_o + 2\sigma\mu P_{1,1} + 2(1-\sigma)\mu P_{1,2} \\
\dot{P}_{1,1} &= -[\lambda+2\sigma\mu]P_{1,1} + \sigma[2\sigma\mu P_{2,1} + 2(1-\sigma)\mu P_{2,2}] + \sigma\lambda P_o \\
\dot{P}_{1,2} &= -[\lambda+2(1-\sigma)\mu]P_{1,2} + (1-\sigma)[2\sigma\mu P_{2,1} + 2(1-\sigma)\mu P_{2,2}] + (1-\sigma)\lambda P_o \qquad (17) \\
\dot{P}_{n,1} &= -[\lambda+2\sigma\mu]P_{n,1} + \sigma[2\sigma\mu P_{n+1,1} + 2(1-\sigma)\mu P_{n+1,2}] + \lambda P_{n-1,1} \\
\dot{P}_{n,2} &= -[\lambda+2(1-\sigma)\mu]P_{n,2} + (1-\sigma)[2\sigma\mu P_{n+1,1} + 2(1-\sigma)\mu P_{n+1,2}] + \lambda P_{n-1,2}
\end{aligned}$$

These equations emphasize that the transition from the state n+1 to the state n,i depend not only on the rate of service completion in the n+1 state $(2\sigma\mu P_{n+1,1}+2(1-\sigma)\mu P_{n+1,2})$, but also on the probability that the next unit will enter phase i.

Although the modification of the service time has been emphasized in this discussion, the same technique can be applied to obtain Erland and Hyper-Exponential arrival-time distributions. In thise case, arrival intervals are considered to be exponentially distributed, but arrivals must pass through a pre-processing channel composed of series or parallel exponential phases before arriving at the system input.

4.2.3. SUMMARY

As long as the transitions between states are exponentially distributed, the equations of detailed balance are first-order, constant-coefficient, differential equations. It is important to emphasize that the states can be defined with considerable generality. Specifically, the arrival- and service-time distributions can be extended beyond the exponential case by including the phase concept. Sequential queues and systems with priority can be handled by further refinements in the definition of a system state.

4.3. Queueing in Exponential Service Facilities

The steady-state and transient solutions for a single- and multiple-channel service facility are summarized in this section. The arrival intervals and service times are assumed to be exponentially distributed and the queue discipline is first come, first served. It has been shown in Section 4.2.1 that these queueing problems can be described by a set of linear, first-order, differential equations.

$$\dot{P}] = [U]\,P] \qquad (18)$$

where:

$\dot{P}]$ = a column vector of the state probability derivatives;

$[U]$ = the transition matrix; and

$P]$ = a column vector of state probabilities.

The steady-state state probabilities P_n are statistical descriptions of the system under normal conditions, i.e., normal in the sense that the sta-

tistics are not time dependent. The transient solutions for the state probabilities $P_n(t)$, on the other hand, are functions of the observation time from a given initial state. For example, assume that a service facility is empty at the beginning of each working day. The state probability $P_n(t)$ could be obtained experimentally from the ensemble of observations taken at time t each morning.

The transient solutions are obtained by solving the set of differential equations, for a given set of initial conditions on the state probabilities, indicated in Eq. (18). On the other hand, the steady-state solutions are obtained by setting $\dot{P}]$ equal to zero and solving the resulting set of algebraic equations.

The following discussion is divided into two major sections. The single-channel (M=1) system is considered in Section 4.3.1 and the multiple-channel (M > 1) system is covered in Section 4.3.2. Recall that M is, by definition, equal to the number of channels and N designates the highest allowable state. These parameters are used throughout the following sections to emphasize the particular type of service facility under study. Although the topic discussed under each section will differ in emphasis and content, the general organization of each section will be as follows:

Steady-State Solutions

Steady-State Statistics

Transient Solutions

4.3.1. SINGLE CHANNEL (M = 1)

The single-channel system is of interest (a) as an introduction to the multiple-channel facility and (b) as an analogies of the conventional computer. The steady-state solutions, steady-state statistics, and transient solutions for the finite and infinite queue single-channel service facility are summarized in Sections 4.3.1.1-4.3.1.3. Since the objective is to present the results so as to aid in understanding the basic problems, the formulas are not always written in their most sophisticated form. An attempt is made to indicate the important steps for each derivation in the discussion accompanying each equation.

4.3.1.1. Steady-State Solutions

Setting $\dot{P}]$ equal to zero, the steady-state equations of detailed balance for a single-channel system having a maximum queue of length two (N = 3) is given by

$$
\begin{aligned}
0 &= -\rho P_0 + P_1 \\
0 &= \rho P_0 - (\rho+1)P_1 + P_2 \\
0 &= \rho P_1 - (\rho+1)P_2 + P_3 \\
0 &= \rho P_2 - P_3
\end{aligned}
\tag{19}
$$

where:

$$\rho = \lambda/\mu.$$

From the first equation it is apparent that P_1 is related to P_0 by the parameter ρ and, in general, we have the following relationship between P_n and P_{n-1}.

$$P_n = \rho P_{n-1}$$

or

$$P_n = \rho^n P_0 \qquad 0 \leqslant n \leqslant N \tag{20}$$

P_0 is determined by one additional constraint on the state probabilities. The sum of all P_n's is equal to one. Thus, P_0 is given by

$$P_0 = 1/\sum_{n=0}^{N} \rho^n \tag{21}$$

For the infinite queue Eq. (20) still holds. This follows from the fact that the n^{th} equation relating P_{n-1}, P_n, and P_{n+1} is the same for both the finite and infinite queue case. That is,

$$P_n = \rho^n P_0 \qquad 0 \leqslant n < \infty \tag{22}$$

Again the state probabilities must still sum to one and, as a result, P_0 is given by

$$P_0 = 1 - \rho \qquad N = \infty \text{ only} \tag{23}$$

which follows from Eq. (21) by recognizing $\sum_{n=o}^{\infty} \rho^n$ as the power series expansion for $1/1-\rho$ about $\rho = 0$, $\rho < 1$. The interpretation of the utilization factor ρ is apparent from a consideration Eq. (23). The probability of the system being empty is P_o. Therefore, the average system utilization is $1-P_o$ or ρ for the single-channel, infinite-queue case.

It is instructive to compare the state probabilities for the finite and infinite queue system. A few values of the state probabilities for various values of ρ and N are given in Table II. Notice that the P_n's for a finite queue are slightly larger than for the infinite queue case. In addition, the state probabilities tend to become evenly distributed over the n states as ρ is increased.

TABLE II

STATE PROBABILITIES FOR A SINGLE-CHANNEL SYSTEM

State Probabilities	ρ = .25 N = 3	ρ = .50 N = 7	ρ = .70 N = 10	ρ = .25 N = ∞	ρ = .50 N = ∞	ρ = .70 N = ∞
P_o	.75294	.50394	.30873	.75000	.50	.3
P_1	.18822	.25197	.21611	.18750	.250	.210
P_2	.04706	.12599	.15128	.04688	.1250	.14700
P_3	.01177	.96300	.10590	.01172	.06250	.10290
P_4	---	.03150	.074130	.002930	.03125	.072030
P_5	---	.015750	.051891	.0007325	.015625	.050421
P_6	---	.0078750	.036324	---	.0078125	.035295
P_7	---	---	.025427	---	.0039063	.024707
P_8	---	---	.017799	---	.0019532	.017295
P_9	---	---	.012459	---	.00097660	.012107

4.3.1.2. Steady-State Statistics

The steady-state measures of system performance fall into three categories: direct interpretation of the state probabilities, the moments obtained from the state probabilities, and the distributions for the waiting

and throughput time. It is evident that the system performance is measured in terms of some criterion, i.e., statistic, which is appropriate for the particular system application. For example, mean throughput time is one measure for a service facility. It is important to realize that the following relationships are valid only under steady-state conditions.

From a direct interpretation of the state probabilities one can determine the following probabilities:

P_o = probability that the system is empty;

$\sum_{n=2}^{N} P_n$ = the average fraction of the time that the queue is occupied; and

P_N = the average fraction of time for which the system is saturated ($N < \infty$) and/or the average fraction of the units which do not enter the queue.

In general, the quantities indicated above are important if the reduction of system idle time is a prime consideration. However, as ρ is increased to effect this decrease in idle time, other statistics (mean throughput time, etc.) increase to intolerable levels.

There are a number of measures of system performance which are functions of the state probabilities. Several of the following equations expressing these measures are applicable to both the finite and infinite queue system, i.e., N in the equations is simply replaced by $N = \infty$. Many of these series can be easily reduced to closed form for the infinite queue case, and these formulas will be qualified by "$N = \infty$ only." Otherwise, it can be assumed that the relationship is valid for both cases.

The mean number of units in the system and in the queue are given by

$$L = \sum_{n=1}^{N} nP_n$$

$$L = P_o \sum_{n=1}^{N} n\rho^n \qquad (24)$$

$$L_q = \sum_{n=1}^{N} (n-1)P_n$$

$$L_q = L - 1 + P_o \qquad (25)$$

where:

L = mean number of units in the system; and

L_q = mean number of units in the queue.

When the queue is infinite, L and L_q are given by

$$L = \rho/1 - \rho \qquad N = \infty \text{ only} \tag{26}$$

$$L_q = \frac{\rho^2}{1-\rho} \qquad N = \infty \text{ only} \tag{27}$$

The mean queue length L_q is plotted in Figure 19 as a function of ρ. The significance of the first-order pole at $\rho = 1.0$ is simply that the queue grows infinitely long (in infinite time) when ρ is equal to one.

The system utilization factor, s.u.f, is always of general interest. It is clear that the mean number of units in service

$$L_s = 1P_1 + 1 \sum_{m=1}^{N-1} P_{1+m}$$

$$L_s = 1 - P_o \tag{28}$$

where:

L_s = mean number in service

cannot exceed the number of channels. Therefore, system utilization, which is equivalent to channel utilization since queued units are assumed to enter the first channel that becomes available, is equal to the mean number of units in service per channel.

$$\text{s.u.f} = \frac{L_s}{M} \tag{29}$$

when $M = 1$

$$\text{s.u.f} = \rho \qquad N = \infty \text{ only} \tag{30a}$$

$$\text{s.u.f} = \rho \frac{(1-\rho^N)}{(1-\rho^{N+1})} \qquad N \text{ finite} \tag{30b}$$

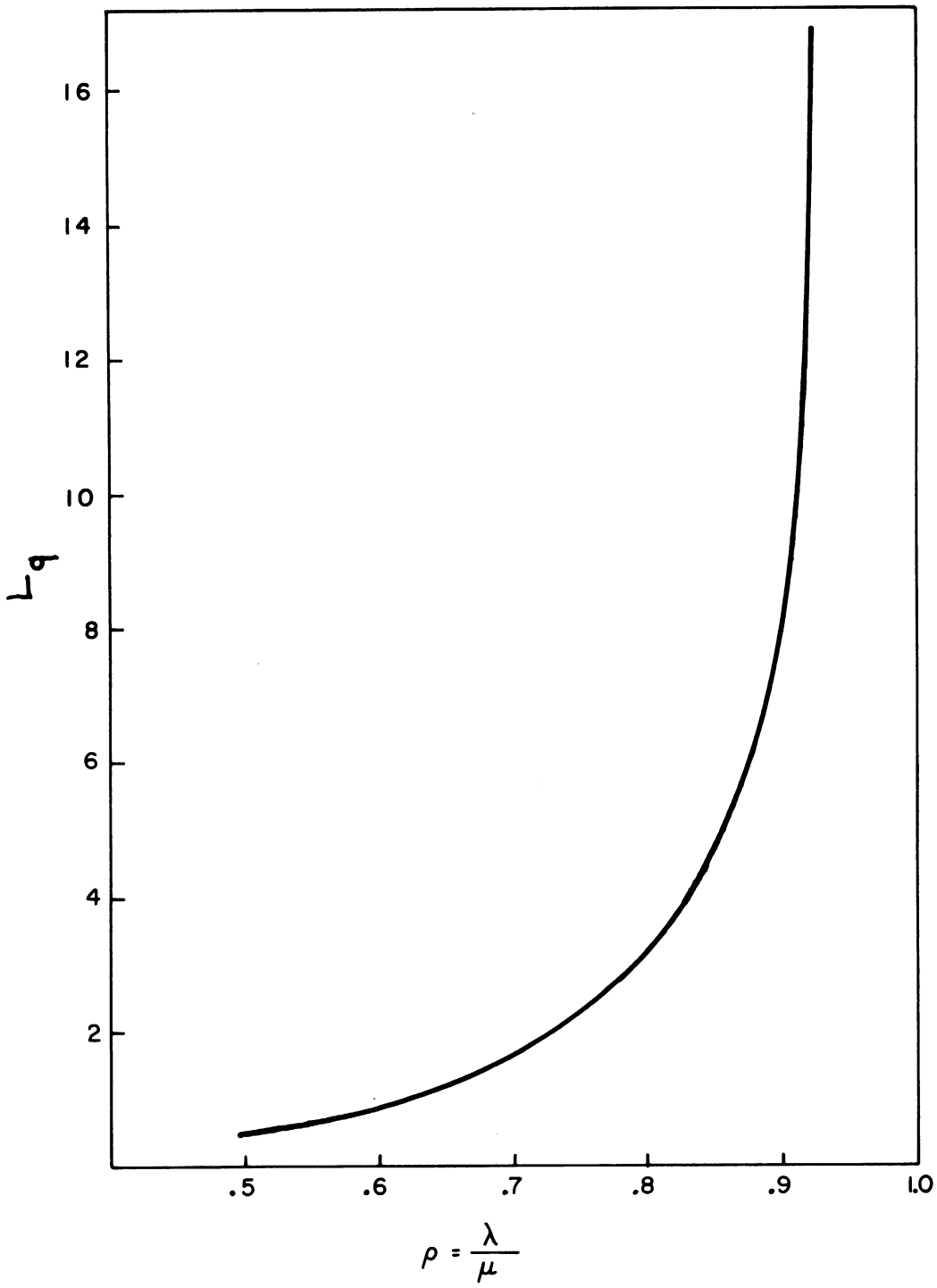

Figure 19. Queue length (L) for a single-channel system (M=1, N=∞).

For a finite-queue system P_o approaches zero for very large values of ρ, and for an infinite-queue system ρ must be less than one for the steady-state solutions to exist. Thus, under steady-state conditions

$$s.u.f \leqslant 1$$

for finite- and infinite-queue systems. This condition establishes the validity of the original definition, Eq. (29). Furthermore, the probability that the system is utilized, or busy, is equal to $1-P_o$.

It is apparent from Eq. (30b) that the s.u.f is not equal to ρ for a finite-queue system but only approximates ρ for large N and small ρ. This point can be emphasized if we recall that ρ can exceed one for a finite-queue system. Nevertheless, ρ is often referred to as the utilization factor for finite- and infinite-queue service facilities.

Although the mean-value statistics are important, higher-order moments are also of interest. For example, the variance of the number of units in the system is

$$\text{Var}\ (L) = \sum_{n=1}^{N} n^2 P_n - (L)^2 \tag{31}$$

$$\text{Var}\ (L) = \frac{\rho}{(1-\rho)^2} \qquad N = \infty \text{ only} \tag{32}$$

Finally, we want to determine the distribution of thw waiting time in the queue and the distribution of the throughput time for a single unit. Throughput time is by definition the interval between the time of arrival and service completion for a given unit. Since the service time is exponentially distributed, the number of completions in a time t_w is Poisson-distributed. Given that the system is in state n when a given unit arrives at the system input, we want to know the joint probability of (n-1) outputs during the interval t_w and one additional service completion during the next interval dt_w. Since the servicing of units is assumed to be independent, the waiting-time incremental probability $p_n(t_w)dt_w$, conditional upon the fact that the system is in state n, is given by:

$$p_n(t_w)dt_w = \left[\frac{(\mu t_w)^{n-1} e^{-\mu t_w}}{(n-1)!}\right]\left[\mu dt_w\right] \tag{33}$$

where:

μ = mean service rate for the channel; and

t_w = waiting time.

The first come, first serve queue discipline is apparent in the derivation of Eq. (33), as the unit which has just arrived waits until all n units have been serviced. It follos that the waiting-time density function, considering all states n, is

$$p(t_w) = \sum_{n=1}^{N} P_n p_n(t_w) + \delta(0) P_o \qquad (34a)$$

$$p(t_w) = P_o \mu e^{-\mu P_o t_w} + \delta(0) P_o \qquad N = \infty \text{ only} \qquad (34b)$$

where:

$\delta(0)$ = Dirac delta function.

Thus the probability that the waiting time will be greater than or equal to some specified time t is equal to the integral of the density function between t and infinity. The delta function accounts for the probability that the system is empty.

$$P(t_w \geqslant t) = \int_{t_w}^{\infty} p(t_w) dt_w \qquad (35)$$

where:

$P(t_w \geqslant t)$ = the waiting-time probability distribution function.

The derivation of the throughput time is analogous to that of the waiting time. In addition to considering the waiting which occurs when the arriving unit finds the system in state $n \geqslant 0$, it is necessary to consider the service time for the unit. The probability density function for the throughput time t_p is equal to the convolution (*) of the waiting and service-time density functions. That is, we want to find the probability density of waiting time t_w and a service time $t_s = t_p - t_w$ for all t_w's between 0 and t_p.

$$p(t_p) = p(t_w) * p(t_s) \qquad (36)$$

For the infinite-queue case, $p(t_w)$ can be expressed in closed form and, as a result, the following relationship can be easily verified.

$$p(t_p) = P_o\mu e^{-\mu P_o t_p} \qquad N = \infty \text{ only} \qquad (37)$$

with the following substitution in Eq. (36).

$$p(t_w) = P_o\mu e^{-\mu P_o t_w} + \delta(0)P_o$$

$$p(t_s) = \mu e^{-\mu t_s}$$

$$P_o = 1 - \rho$$

Again, the probability that the throughput time t_p will exceed a given value t is given by

$$P[t_p \geqslant t] = \int_t^\infty p(t_p)dt_p$$

$$P[t_p \geqslant t] = e^{-\mu[1-\rho]t} \qquad N = \infty \text{ only} \qquad (38)$$

Thus, as ρ approaches unity it is nearly certain that the throughput time, under steady-state conditions, will exceed relatively large values of t. However, it is important to realize that the steady-state condition for the infinite-queue case has no meaning as ρ approaches one.

The mean waiting time and throughput time can always be obtained by finding the first moments of their respective distributions. However, it is apparent that T_w and T_p are given by

$$T_w = L_q T_a \qquad (39a)$$

$$T_p = L_q T_a + T_s = LT_a \qquad (39b)$$

That is, a first come, first serve discipline specifies that a unit advances in the queue, relative to the end of the line, when a new arrival enters the queue [Eq. (39a)]. Therefore, the mean waiting time is equal to the product of the mean queue length and the average interval between arrivals.

4.3.1.3. Transient Solutions for State Probabilities

The steady-state solutions for the state probabilities are generally sufficient for most queueing problems. However, there are certain situations in which the transient solution is of interest. For example, the mean arrival rate λ is often a function of time. It is evident that the arrival rate can be assumed to be constant over the time interval of interest provided the transients in the state-probability solutions decay in a time which is short compared to the fluctuations in the mean arrival rate λ. In such cases the transient solutions must be considered before the steady-state solutions and statistics can be applied.

The equations of detailed balance for a single-channel (N = 3) system are of the form

$$\begin{bmatrix} \dot{P}_0(t) \\ \dot{P}_1(t) \\ \dot{P}_2(t) \\ \dot{P}_3(t) \end{bmatrix} = \begin{bmatrix} -\rho & 1 & & \\ \rho & -\rho-1 & 1 & \\ & \rho & -\rho-1 & 1 \\ & & \rho & -1 \end{bmatrix} \begin{bmatrix} P_0(t) \\ P_1(t) \\ P_2(t) \\ P_3(t) \end{bmatrix} \tag{40}$$

There are, of course, two methods for obtaining solutions for the state probabilities. The equations can be solved directly, using the computer, or the solutions can be obtained by analytical methods. It turns out that the former method is satisfactory for our requirements and, in addition, involves considerably less labor than the analytical approach. Nevertheless, considerable insight can be gained by looking at one typical problem in detail. The analytical solutions for a single-channel system having a finite (N = 3) and infinite (N = ∞) queue will be discussed in the following sections.

4.3.1.3.1. Finite Queue.—The theory required for the solution of systems of linear, constant-coefficient, differential equations is well known (Ref. 2), and therefore only a few of the high points will be summarized here. For example, the equations of detailed balance can be written as

$$\dot{P}] = [U]\,P] \tag{41}$$

Assuming a solution of the form $e^{\gamma t}$, the following set of homogeneous algebraic equations is obtained:

$$[f(\gamma)]\,P] = 0 \tag{42}$$

where:

$[f(\gamma)]$ = $[I]\gamma - [U]$;

$[f(\gamma)]$ = the gamma matrix;

$[I]$ = the identify matrix;

$[U]$ = the transition matrix; and

$P]$ = a column vector of state probabilities.

To obtain nontrivial solutions for the state probabilities, the determinant of the gamma matrix must equal zero. That is, the γ's are equal to the roots of the characteristic equations.

$$\Delta_m(\gamma) = 0 \tag{43}$$

where:

$\Delta_m(\gamma)$ = $|f(\gamma)|$; and

m = N+1, the degree of the characteristic polynomial.

Having determined the roots of the characteristic polynomial $\gamma_1, \gamma_2, \ldots \gamma_i \ldots \gamma_m$, we find that the solutions for the state probabilities are of the form

$$P] = [k][M(t)][k^{-1}]P(0)] \tag{44}$$

where:

$[k]$ = the matrix of eigenvectors;

$[k^{-1}]$ = the inverse of the $[k]$ matrix; and

$P(0)]$ = a column vector specifying the system status at $t = 0$.

$$[M(t)] = \begin{bmatrix} e^{\gamma_1 t} & & & \\ & e^{\gamma_2 t} & & \\ & & \ddots & \\ & & & e^{\gamma_m t} \end{bmatrix} ; \quad P(0)] = \begin{bmatrix} P_0(0) \\ P_1(0) \\ \vdots \\ P_N(0) \end{bmatrix}$$

When the roots are distinct the eigenvector vector $k^i]$ associated with the i^{th} root γ_i is proportional to any column of the adjoint matrix $[F(\gamma_i)]$. (Recall that the adjoint matrix $[F(\gamma_i)]$ is, by definition, the transpose of the gamma matrix co-factors.) When the roots have a multiplicity greater than one, the determination of the eigenvector is more involved. However, the above summary is adequate for this discussion, since the roots have been distinct for all the queueing problems which have been considered.

If the last two matrices in Eq. (44) are considered, it is apparent that the column vector $P(0)]$ "selects" only certain columns from the $[k^{-1}]$ matrix. For example, for an $N = 3$ system in state zero at time $t = 0$

$$[k^{-1}]P(0)] = \begin{bmatrix} k_{11}^{-1} & \cdots & k_{14}^{-1} \\ \vdots & & \vdots \\ k_{41}^{-1} & \cdots & k_{44}^{-1} \end{bmatrix} \begin{bmatrix} P_1(0) \\ \vdots \\ P_4(0) \end{bmatrix} = \begin{bmatrix} k_{11}^{-1} \\ \vdots \\ k_{41}^{-1} \end{bmatrix} \tag{45}$$

and

$$[k][M(t)] = \begin{bmatrix} k_{11}e^{\gamma_1 t} & \cdots & k_{14}e^{\gamma_4 t} \\ \vdots & & \vdots \\ k_{41}e^{\gamma_1 t} & \cdots & k_{44}e^{\gamma_4 t} \end{bmatrix} \tag{46}$$

Therefore,

$$
\begin{aligned}
P_0(t) &= k_{11}k_{11}^{-1}\, e^{\gamma_1 t} + k_{12}k_{21}^{-1}\, e^{\gamma_2 t} + k_{13}k_{31}^{-1}\, e^{\gamma_3 t} + k_{14}k_{41}^{-1} \\
P_1(t) &= k_{21}k_{11}^{-1}\, e^{\gamma_1 t} + k_{22}k_{21}^{-1}\, e^{\gamma_2 t} + k_{23}k_{31}\, e^{\gamma_3 t} + k_{24}k_{41}^{-1} \\
P_2(t) &= k_{31}k_{11}^{-1}\, e^{\gamma_2 t} + k_{32}k_{21}^{-1}\, e^{\gamma_3 t} + k_{33}k_{31}^{-1}\, e^{\gamma_3 t} + k_{34}k_{41}^{-1} \\
P_3(t) &= k_{41}k_{11}^{-1}\, e^{\gamma_3 t} + k_{42}k_{21}^{-1}\, e^{\gamma_4 t} + k_{43}k_{31}^{-1}\, e^{\gamma_4 t} + k_{44}k_{41}^{-1}
\end{aligned}
\qquad (47)
$$

when:

$$P_0(0) = 1,\ P_1(0) = P_2(0) = P_3(0) = 0$$

If, on the other hand, the system is initially in state one, $[k^{-1}]\ P(0)\ ]$ would be equal to the second column of the $[k^{-1}]$ matrix. It is evident, therefore, that the steady-state values of the state probabilities are

$$P_0 = k_{14}k_{41}^{-1} = k_{14}k_{42}^{-1} = k_{14}k_{43}^{-1} = k_{14}k_{44}^{-1}$$

$$\vdots$$

$$P_3 = k_{44}k_{41}^{-1} = k_{44}k_{42}^{-1} = k_{44}k_{42}^{-1} = k_{44}k_{44}^{-1}$$

For a single-channel system with a maximum queue of length two ($P_0(0) = 1$, $N = 3$) the complete solutions for $\rho = 0.25$ are

$$
\begin{aligned}
P_0(t) &= .0585\, e^{-1.957t} + .144\ e^{-1.25t} + .0498\, e^{-.543t} + .753 \\
P_1(t) &= .0998\, e^{-1.957t} - .144\ e^{-1.25t} - .143\ e^{-.543t} + .188 \\
P_2(t) &= +.0559\, e^{-1.957t} + .0359\, e^{-1.25t} - .139\ e^{-.543t} + .0471 \\
P_3(t) &= .0147\, e^{-1.957t} + .0359\, e^{-1.25t} - .0622\, e^{-.543t} + .0117
\end{aligned}
\qquad (48)
$$

At least two observations can be made from the above equations. First, the coefficients of the time-dependent terms are the same order of magnitude as the steady-state terms and may be of either sign depending upon the initial condition. As a result, $P_0(t)$ is greater than P_0 while $P_1(t)$, $P_2(t)$

and $P_3(t)$ are less than their steady-state values. Second, the separation between the roots γ_1, γ_2, γ_3 is not sufficient to allow one to approximate the transient solution (for large t) by one minimal root and the steady-state probabilities. The solutions, in terms of t normalized to the mean service-time for a single channel, are plotted in Figure 20. The value of $P_n(6.0)$ is indicated on the figure with the steady-state value included, in parenthesis, for comparison purposes. At t = 6.0 service-times, the percentage error from the steady-state value in the computation of the mean number in the system is 2.75% with the following errors in the state probabilities: .6, .7, 4.3, 8.5% for n = 0, 1, 2, 3, respectively. It is apparent that the time required to reach P_3, assuming $P_0(0) = 1$, is greater than the time required to reach, for a specified percent error, the steady-state probabilities P_n $(n < 3)$.

The major advantage of a complete analytical solution over a strictly numerical solution is the added insight which can be gained by a consideration of the coefficients and roots as illustrated by Eq. (48). However, the complete process of root extraction, eigenvalue determination, and matrix inversion in practical only if programmed on a computer; and since The University of Michigan Computing Center had a subroutine available for the solution of linear, first-order, differential equations using the Runge-Kutta iterative technique,[4] this rather than the analytical approach was chosen.

4.3.1.3.2. Infinite Queue.—When the queue is infinite the analytical method indicated in the previous section is not applicable because the [U] matrix is of infinite order. L. Saaty (Ref. 3) circumvents this difficulty by the following technique. There are three major steps in the derivation of the final result.

First, the generating function P(z,t) is obtained from the equations of detailed balance:

$$
\begin{aligned}
\dot{P}_0(t) &= -\rho P_0(t) + P_1(t) \\
\dot{P}_1(t) &= \rho P_0(t) - (\rho+1)P_1(t) + P_2(t) \\
&\vdots \\
\dot{P}_n(t) &= \rho P_{n-1}(t) - (\rho+1)P_n(t) + P_{n+1}(t) \\
&\vdots
\end{aligned}
\qquad (49)
$$

[4]The program for this computation is described in Appendix C.

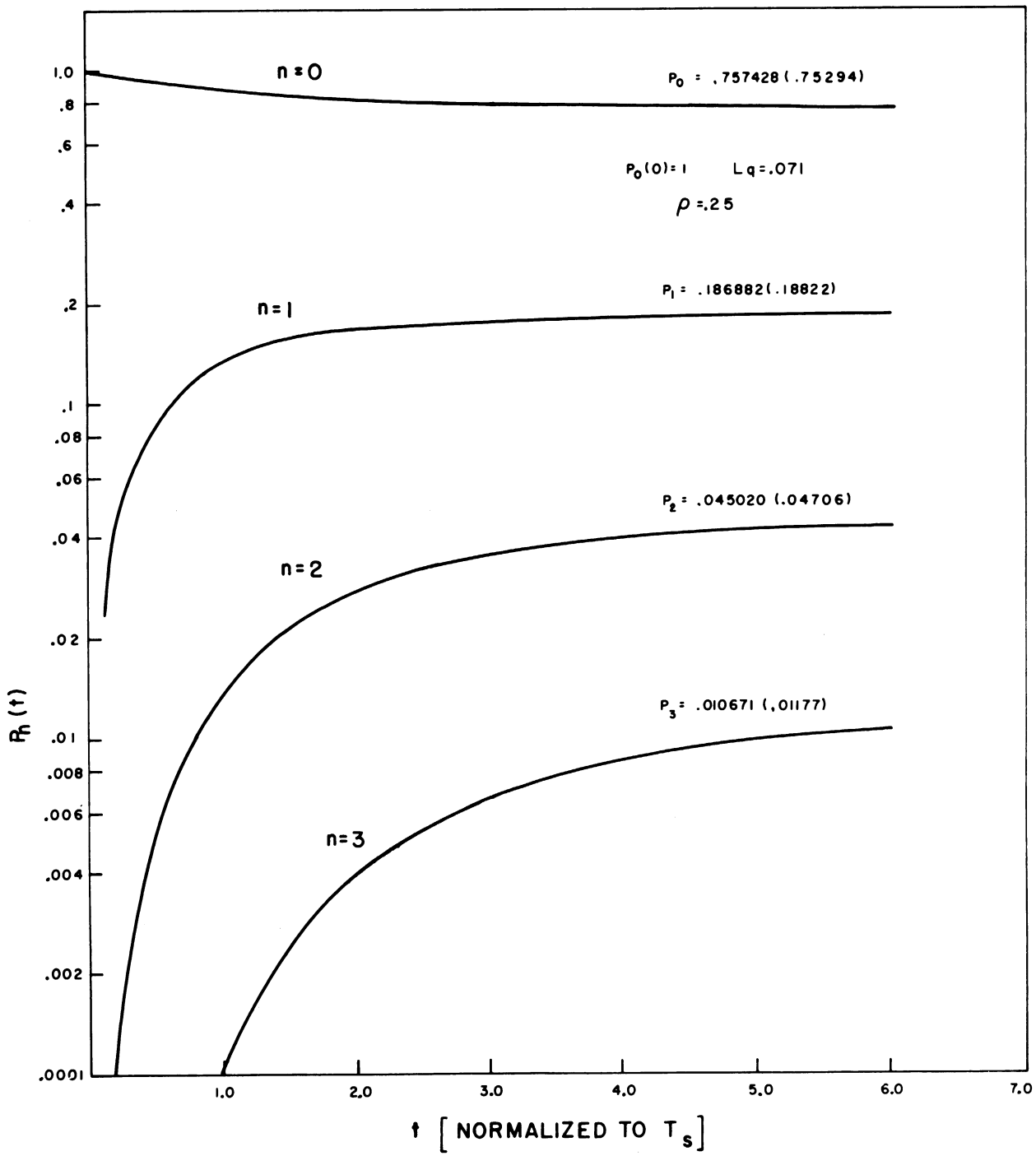

Figure 20. Transient state probabilities for a single-channel system (M=1, N=3).

by summing the above equations. That is,

$$\frac{\partial P}{\partial t}(z,t) = -(\rho+1)P(z,t) + \rho z P(z,t) + \frac{1}{z}P(z,t) - \frac{1}{z}P_o(1) + P_o(1) \tag{50}$$

where:

$$P(z,t) = \sum_{n=0}^{\infty} z^n P_n(t); \text{ and}$$

$$P(z,0) = 1 \text{ (assuming } P_o(0) = 1).$$

The Laplace transform of the generating function $P(z,t)$ is then obtained, and is given by

$$P(z,s) = \frac{z - (1-z)P_o(s)}{-[\rho z^2 - (\rho+1+s)z+1]} \tag{51}$$

Second, the poles, $\alpha_1(s)$ and $\alpha_2(s)$, of this transformed generating function are obtained:

$$\alpha_2(s) = \frac{\rho+1+s}{2\rho} - \frac{1}{2\rho}\sqrt{(\rho+1+s)^2 - 4\rho}$$

where:

$|\alpha_2(s)| < 1$, and $\alpha_1(s)$ is found from the relationship

$$\alpha_1(s) \cdot \alpha_2(s) = \rho$$

Because $P(z,s)$ is known to be an analytic function of z for $|z| \leq 1$, and since $P(z,s)$ has a pole at $z = \alpha_2(s)$, the numerator of $P(z,s)$ must have a zero at $z = \alpha_2(s)$. It follows therefore that

$$\alpha_2(s) - [1 - \alpha_2(s)]P_o(s) = 0 \tag{52a}$$

or

$$P_o(s) = \frac{\alpha_2(s)}{1 - \alpha_2(s)} \tag{52b}$$

The third and final step is to take the inverse transform $L^{-1}\{P_o[\alpha_2(s)]\}$ (Ref. 4), where $\alpha_2(s)$ is a irrational function of s. The result is

$$P_o(t) = \frac{e^{-(\rho+1)t}}{t} \sum_{k=0}^{\infty} (2)^{k+1}(k+1) I_{k+1}(2\sqrt{\rho}\, t) \qquad (53)$$

The expression for $P_o(t)$ is thus equal to the product of a decaying exponential and an infinite sum of modified Bessel functions. The solutions for $\rho = .25$, .5, .8, and .9 are indicated in Figure 21. The principal conclusion to be drawn from this figure is that as ρ approaches one the time required to reach the steady-state probability becomes infinite. It follows, therefore, that for the infinite queue one must use considerable caution in applying steady-state statistics for large value of ρ.

To obtain other state probabilities, e.g., $P_1(t)$, simply take the Laplace transform of the first equation in set (49), $P_o(s)$ from Eq. (52b), and solve for $P_1(s)$.

$$P_1(s) = (s+\rho) \frac{\alpha_2(s)}{1 - \alpha_2(s)} - 1 \qquad (54)$$

The transforms for the remaining $P_n(s)$ can be obtained similarly, and the $P_n(t)$ are of course determined from the inverse transforms.

4.3.1.3.3. The Approximation of an Infinite-Queue System by Finite System.—It is reasonable to theorize that, if ρ is small and N is large, the transient solutions for the finite- and infinite-queue cases should be nearly identical. That is, if the steady-state probabilities are nearly equal, then all the $P_n(t)$'s should be nearly equal. This is in fact the case, as illustrated by Table III. It turns out that, for a $\rho = .25$ and $N = 4$, the errors are less than .01. The error is greatest for small n and small t. The ΔP_o, the difference between the steady-state probabilities,[5] for $\rho = .8$ and $N = 20$ is equal to .002. Thus, by analogy, a system of $N+1 = 21$ equations of detailed balance is satisfactory for our purpose.

4.3.2. MULTIPLE CHANNEL (M > 1)

Since the discussion in this section roughly parallels that of Section 4.3.1, the major results can be stated without repeating the detail of the previous section. Although the seven-channel system is of particular interest, since this is analogous to the AN/FSQ-27 system, the two-channel system will be used to illustrate the multiple-channel system because at least eight

[5]Recall Eqs. (20) and (22).

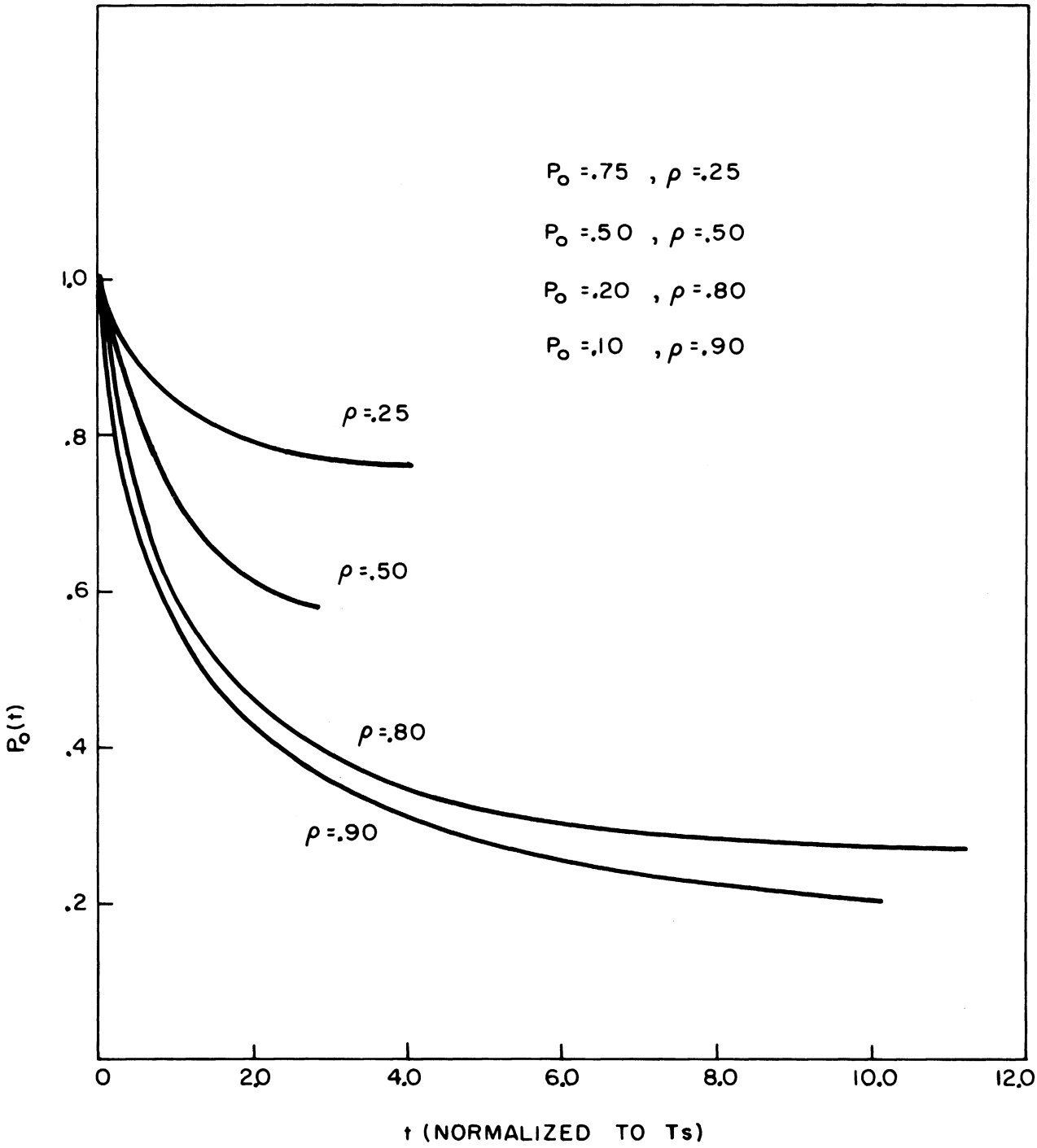

Figure 21. Transient solution, $P_o(t)$, for a single-channel system (M=1, N=∞).

TABLE III

TRANSIENT SOLUTIONS FOR A SINGLE-CHANNEL SYSTEM WITH
$\rho = 0.25$, $P_0(0) = 1.0$

t	State Probabilities P_n	N = 3	N = 4	N = ∞
	P_0	.90623	.90623	.89700
	P_1	.08852	.08852	--
.5	P_2	.00507	.00505	--
	P_3	.00021	.00020	--
	P_4	--	.00001	--
	P_0	.85299	.85299	.84815
	P_1	.13229	.13229	--
1.0	P_2	.01364	.01363	--
	P_3	.00108	.00103	--
	P_4	--	.00006	--
	P_0	.80066	.80065	.78512
	P_1	.16755	.16752	--
2.0	P_2	.02785	.02774	--
	P_3	.00394	.00367	--
	P_4	--	.00042	--
	P_0	.76675	.76651	.76618
	P_1	.18369	.18342	--
4.0	P_2	.04090	.04068	--
	P_3	.00865	.00807	--
	P_4	--	.00152	--
	P_0	.75294	.75294	.75
	P_1	.18822	.18824	.18750
∞	P_2	.04706	.04706	.04688
	P_3	.01177	.01176	.01172
	P_4	--	.00296	.00293

equations are required to describe even the simplest, i.e., a zero-length queue, system. However, the results for the seven-channel case will be included in the discussion. Again, many of the formulas are applicable to both the finite- and infinite-queue systems while others apply to the infinite queue only. The latter will be emphasized by "N = ∞ only."

4.3.2.1. Steady-State Solutions

The steady-state equations in the state probabilities for a two-channel system having a maximum queue of length of two (M = 2, N = 4) are:

$$\begin{aligned} 0 &= -2\rho P_0 + P_1 \\ 0 &= 2\rho P_0 - (2\rho+1)P_1 + 2P_2 \\ 0 &= 2\rho P_1 - (2\rho+1)P_2 + 2P_3 \\ 0 &= 2\rho P_1 - (2\rho+1)P_3 + 2P_4 \\ 0 &= 2\rho P_3 - 2P_4 \end{aligned} \qquad (55)$$

where:

$$\rho = \frac{\lambda}{2\mu} .$$

As for the single channel, the state probabilities are obtained by successive substitutions of P_0, P_1, etc., in the above equations.

$$P_n = (M\rho)^n P_0/n! \qquad 0 \leqslant n < M \qquad (56a)$$

$$P_n = M^M \rho^n P_0/M! \qquad M \leqslant n \leqslant N \qquad (56b)$$

Again Eqs. (56a) and (56b) are applicable when the queue is infinite. However, the values of P_0 for the two cases are given by

$$P_0 = 1\left[\sum_{n=0}^{M-1} \frac{(M\rho)^n}{n!} + \sum_{n=M}^{N} \frac{M^M}{M!}\rho^n\right]^{-1} \qquad (57a)$$

$$P_0 = 1\left[\sum_{n=0}^{M-1} \frac{(M\rho)^n}{n!} + \frac{(M\rho)^M}{M!(1-\rho)}\right]^{-1} \qquad N = \infty \text{ only} \qquad (57b)$$

A few values of the state probabilities for the two-channel system are shown in Table IV. The state probabilities for a seven-channel (M = 7) finite queue (N = 20) are of particular interest in our studies. Values of $P_0 \ldots P_{20}$ are given for a number of values of ρ in Table V. As for the single-channel case, the state probabilities become more nearly uniformly distributed over the N+1 states as ρ becomes large.

TABLE IV

STATE PROBABILITIES FOR A TWO-CHANNEL SYSTEM

State Probabilities	$\rho = .25$ $N = \infty$	$\rho = .50$ $N = \infty$	$\rho = .70$ $N = \infty$
P_0	.60002	.33333	.17647
P_1	.30001	.33333	.24600
P_2	.07500	.16666	.17294
P_3	.01875	.08333	.12106
P_4	.00485	.04166	.08474
P_5	.00121	.02083	.05932
P_6	.00030	.01042	.04152
P_7	.00007	.00521	.02906
P_8	.00002	.00261	.02034
P_9	.00004	.00131	.01424

4.3.2.2. Steady-State Statistics

The three categories for steady-state statistics are: direct interpretation of state probabilities, the moments obtained from the P_n's, and the distribution for waiting and throughput time. The interpretation of the state probabilities will be considered first.

P_0 = probability that the system is empty;

$1-P_0$ = average fraction of the time the system is in use;

$\sum_{n=M+1}^{N} P_n$ = average fraction of time that the queue is occupied; and

P_N = the average fraction of time for which the system is saturated.

TABLE V

STATE PROBABILITIES FOR A SEVEN-CHANNEL SYSTEM

State Probabilities	$\rho = .50$ N = 20	$\rho = .60$ N = 20	$\rho = .70$ N = 20	$\rho = .80$ N = 20
P_0	0.029845	0.0144353	0.0067183	.0028981
P_1	0.104458	0.060628	.032919	.0162294
P_2	0.182801	0.127319	.080653	.0454422
P_3	0.213267	0.178247	.131734	.0848254
P_4	0.186609	0.187159	.161374	.1187556
P_5	0.130626	0.157214	.158146	.133006
P_6	0.076199	0.11005	.129143	.124139
P_7	0.0380993	0.06603	.090407	.099311
P_8	0.0190497	0.0396179	.063285	.079449
P_9	0.0095248	0.023771	.044299	.063559
P_{10}	0.0047624	0.014264	.031009	.050847
P_{11}	0.0023812	0.0085575	.021707	.040678
P_{12}	0.0011906	0.00513448	.0151946	.032542
P_{13}	0.00059532	0.00308069	.0106362	.026034
P_{14}	0.00029766	0.0018484	.0074453	.020827
P_{15}	0.00014883	0.00110905	.00521174	.016661
P_{16}	0.000074415	0.0006654	.0036482	.013329
P_{17}	0.000037207	0.0039926	.00255375	.010663
P_{18}	0.000018604	0.00023955	.00178763	.0085308
P_{19}	0.0000093019	0.00014373	.00125134	.0068246
P_{20}	0.0000046509	0.00008624	.00087593	.0054597

Second, it is clear that the mean number of units in the system and in the system queue are given by:

$$L = \sum_{n=1}^{N} nP_n \tag{58}$$

$$L_q = \sum_{n=1}^{N-M} nP_{M+n} \tag{59}$$

The relationship between L and L_q can be obtained by writing L as:

$$L = \left\{ \sum_{n=1}^{M} nP_n + M \sum_{n=1}^{N-M} P_{M+n} \right\} + \sum_{n=1}^{N-M} nP_{M+n}$$

$$L = L_s + L_q \tag{60}$$

where:

L_s = the average number in service.

Thus, the mean number in the system is equal to the sum of the mean number in service and in queue. When the queue is finite, L_s is equal to

$$L_s = \rho M(1-P_N) \qquad N \text{ finite} \tag{61a}$$

whereas for an infinite-queue length,

$$L_s = \rho M \qquad N = \infty \text{ only} \tag{61b}$$

The system utilization factor is, by Eq. (29)

$$\text{s.u.f} = \rho(1-P_N) \qquad N \text{ finite} \tag{62a}$$

$$\text{s.u.f} = \rho \qquad N = \infty \text{ only} \tag{62b}$$

where:

$$P_N = \frac{M^M}{M!}\rho^N P_o$$

Again, system utilization is equal to ρ for the infinite-queue case and approximately equal to ρ for the finite-queue system when $P_N \ll 1$. The mean queue length for a seven-channel system is plotted in Figure 22. Again, L_q becomes infinite as ρ approaches one for the infinite-queue case. Comparing a single- and a seven-channel system having identical mean queue lengths (i.e., Figures 19 and 22), it is apparent that a multiple-channel system can realize a higher system utilization for a given average queue length than a single-channel system.

The variance, for both the finite and infinite queue, is obtained from the fundamental relationship:

$$\text{Var}(L) = \sum_{n=1}^{N} n^2 P_n - (L)^2 \tag{63}$$

A number of the above formulas have been tabulated in the literature (Refs. 1 and 5). However, the desired results can always be obtained directly from a knowledge of the state probabilities and the above equations.

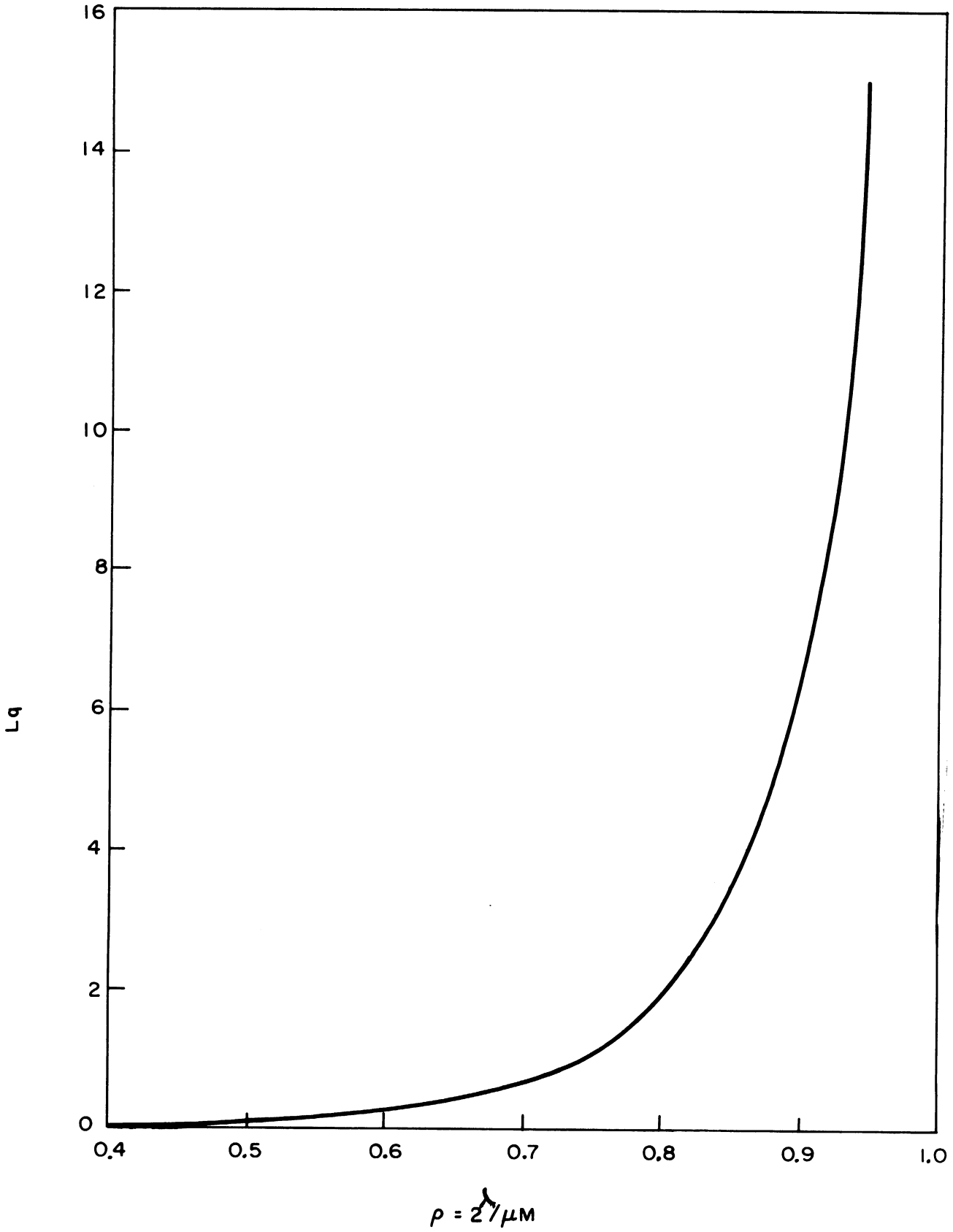

Figure 22. Queue length (L_q) for a seven-channel system (M=7, N=∞).

Waiting will occur in a multiple-channel system only when all the channels are occupied. Since the outputs from seven continuously occupied channels are still exponentially distributed (with a mean service rate of $M\mu$), the incremental probability $p_{M+n}(t_w)dt_w$, conditional upon the fact that the system is in state M+n when the unit of interest arrives, is given by

$$p_{M+n}(t_w)dt_w = \frac{(M\mu t_w)^{n-1}}{(n-1)!} e^{-M\mu t_w} M\mu dt_w \qquad (64)$$

which is the probability that there are initially n units in the queue, that n-1 units have completed service in time t_w, and that one additional unit will be completed in time interval dt_w. The waiting-time density function, considering states $n > M$, is

$$p(t_w) = \sum_{n=1}^{N-M} P_{M+n} p_{M+n}(t_w) + \delta(0) \sum_{n=0}^{N} P_n \qquad (65)$$

The second term in Eq. (65) accounts for the probability that the queue is empty just after the unit of interest arrives. Thus the probability that the waiting time will exceed a specified t is given by

$$P(t_w \geqslant t) = \int_t^\infty p(t_w)dt_w$$

The above equations can be reduced to the following equation for the infinite-queue case:

$$P(t_w \geqslant t) = Q_M e^{-\mu M(1-\rho)t} \qquad N = \infty \text{ only} \qquad (66)$$

where:

$t > 0$

$Q_M = \sum_{n=1}^{N-M} P_{M+n}$ the probability that all channels are busy.

Again, the throughput-time density function is equal to the convolution of the waiting-time density function and the service-time density for a single channel:

$$p(t_p) = p(t_w) * p(t_s)$$

and the probability distribution on throughput time is given by

$$P(t_p \geqslant t) = \int_t^\infty p(t_p)dt_p \tag{67}$$

4.3.2.3. Transient Solutions

The transient solutions for the single- and multiple-channel systems are similar in many respects. Therefore the following brief discussions are essentially extensions of the discussions in Section 4.3.1.3. The principal objective of this section is the presentation of typical transient solutions for a multiple-channel service facility. These transient solutions are applied, in Section 5 to the run-length-specification problem.

4.3.2.3.1. Finite Queue.—The fundamental difference between the single- and multiple-channel systems is in the [U] matrix. For example, the equations of detailed balance for a two-channel (M = 2, N = 3) system is given by

$$\dot{P}] = [U]\,P] \tag{68}$$

where:

$$[U] = \begin{bmatrix} -\rho & +1 & & \\ \rho & -(\rho+1) & 2 & \\ & \rho & -(\rho+2) & 2 \\ & & \rho & -2 \end{bmatrix}$$

Since the roots of the transition matrix determinant are all real and distinct, the solutions for the transient state probabilities are of the form indicated in Eq. (47) of Section 4.3.1.3. The root locus for N = 2, 3 and 4 are plotted in Figure 23. As for the single-channel system, there is always one root at the origin which gives rise to the steady-state probabilities P_n. Also, the minimal root moves toward the origin and becomes less dependent upon the parameter ρ as the maximum allowed state N is increased. One implication from Figure 23 might be that the roots are always real and distinct; however, this is not always the case. Lacking sufficient theory to draw general conclusions, each queueing problem must be investigated separately.

Typical transient solutions for a seven-channel (M = 7, N = 20) system are shown in Figures 24 and 25. It is evident that $P_{20}(0) = 1$ is a severe initial condition as the time required to reach steady-state is increased [consider $P_{10}(t)$] by a factor of two. Since the roots are identical for the

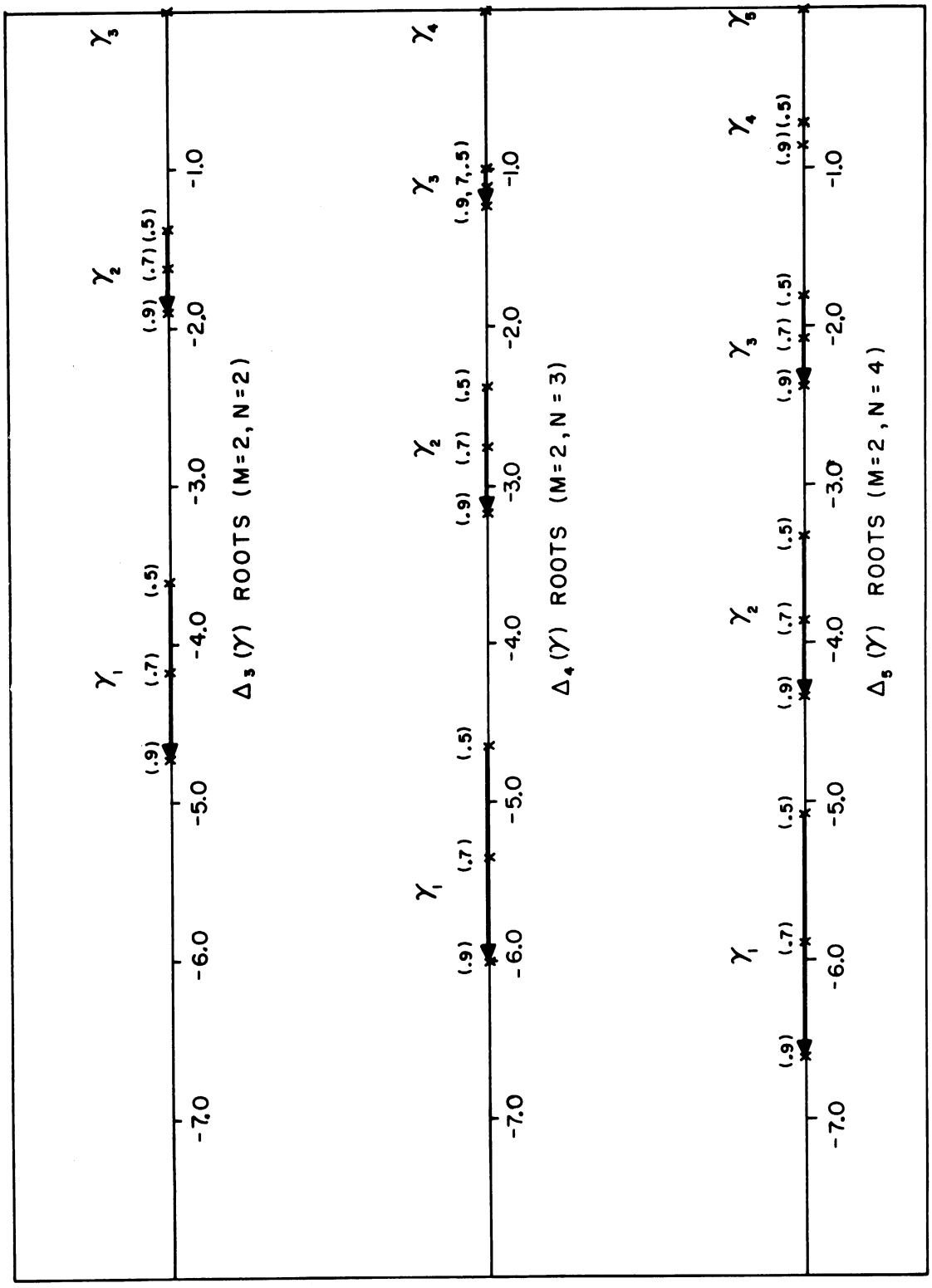

Figure 23. Root locus for a two-channel system.

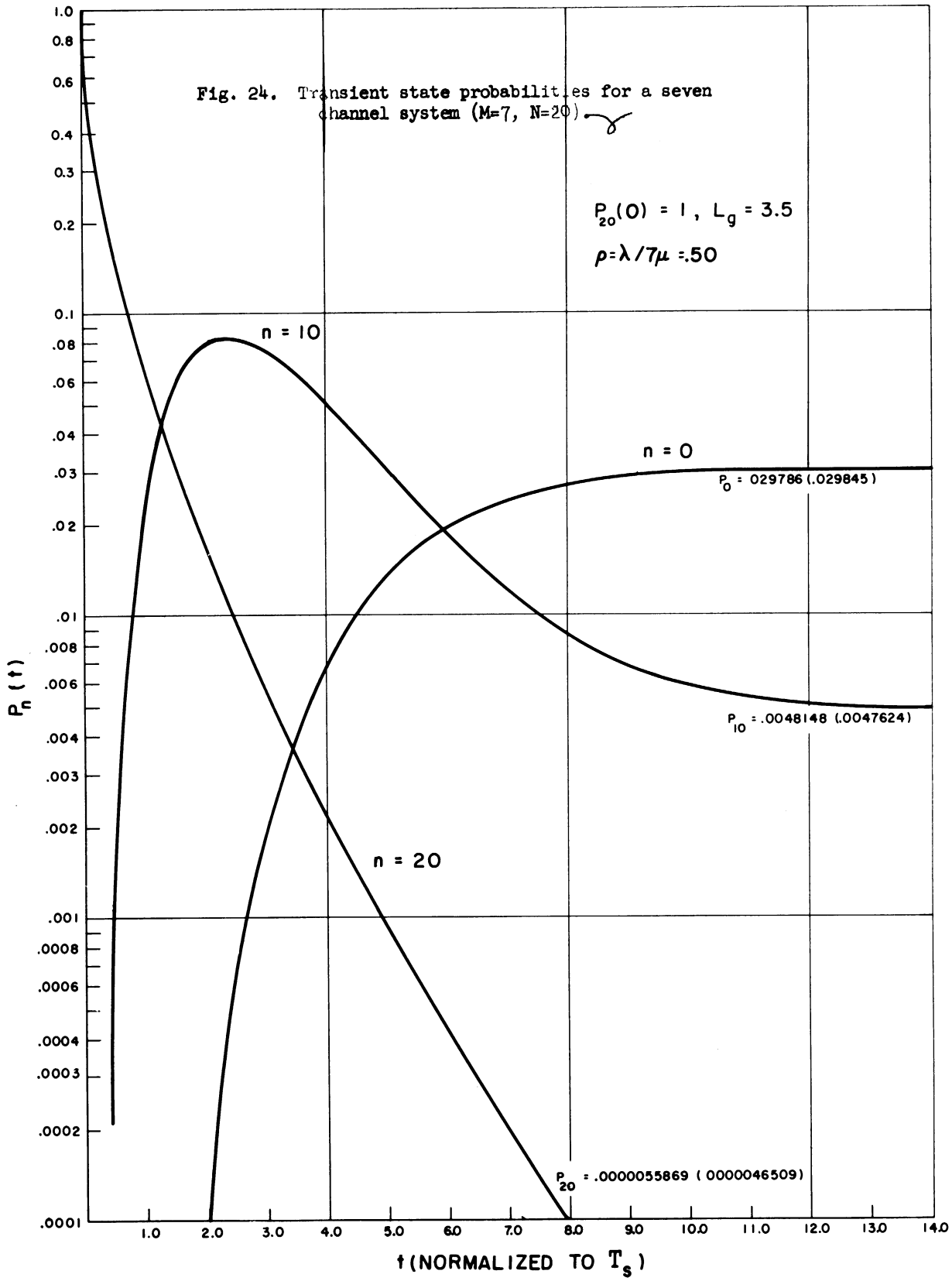

Figure 24. Transient state probabilities for a seven-channel system (M=7, N=20).

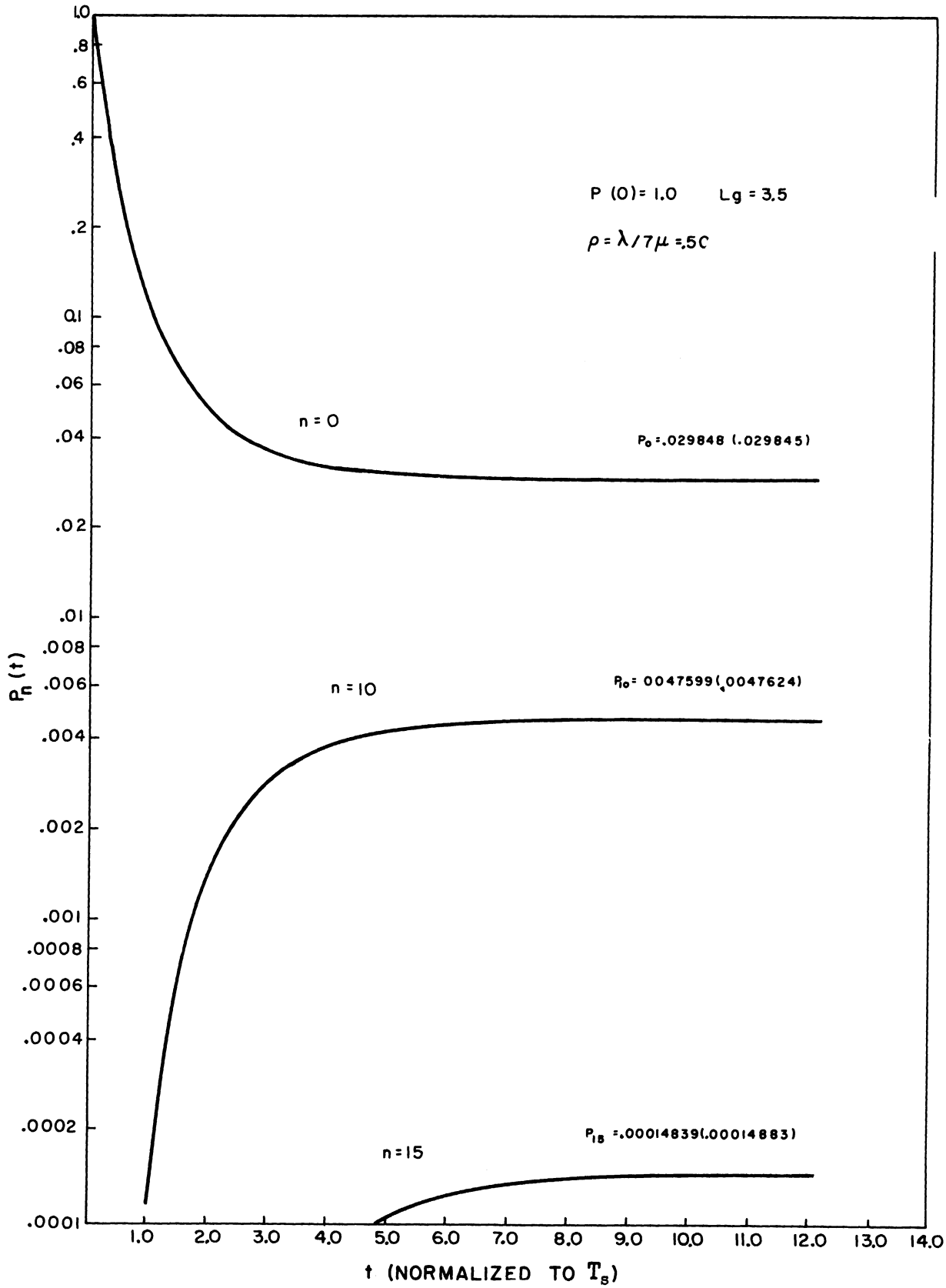

Figure 25. Transient state probabilities for a seven-channel system (M=7, N=20).

two figures, the relative magnitude coefficients of the exponential terms $e^{\gamma_i t}$ cannot be ignored [see Eq. (47)]. That is, the transient behavior cannot be approximated by the steady-state solution P_n and one minimal root.

If a one- and a two-channel system are subject to the same environment, i.e., equal arrival rates, the transient solutions for the two-channel system approach the steady-state values for P_n's much faster than for the single-channel case (Table VI). It is important to realize here that $P_0(\infty)|_{M=1} \neq P_0(\infty)|_{M=2}$ and, therefore, the approach to the steady-state probabilities P_n is noted by looking at the difference between successive values in each column. It is evident that the values of ρ required to obtain equal average arrival rates λ will also, by Eq. (61a) ($P_{10} \ll 1$), result in the same mean number of units in service for the one- and two-channel systems. That is to say, equal environments (λ's equal) imply equal L_s's.

TABLE VI

TRANSIENT SOLUTIONS FOR A ONE- AND A TWO-CHANNEL SYSTEM
(M = 1, N = 10; M = 2, N = 10)

t (Normalized to T_s)	$P_0(t)$		$P_5(t)$		$P_{10}(t)$	
	$\rho = .8$ M = 1	$\rho = .4$ M = 2	$\rho = .8$ M = 1	$\rho = .4$ M = 2	$\rho = .8$ M = 1	$\rho = .4$ M = 2
0	1.0	1.0	0.0	0.0	0.0	0.0
2.0	.47253	.49862	.00532	.00212	1.5×10^{-6}	3.7×10^{-7}
4.0	.37445	.44603	.02232	.00578	.00012	.00001
6.0	.33019	.43447	.03674	.00752	.00079	.00004
8.0	.30402	.43089	.04640	.00839	.00225	.00005
10.0	.28642	.42957	.05275	.00853	.00424	.00007
12.0	.27367	.42903	.05704	.00867	.00645	.00008
14.0	.26396	.42879	.06007	.00873	.00864	.00009
16.0	.25632	.42869	.06228	.00875	.01068	.00009
18.0	.25017	.42864	.06397	.00876	.01252	.00009
20.0	.24513	.42862	.06529	.00877	.01413	.00009

4.3.2.3.2. Infinite Queue.—When the queue is infinite and the number of channels is greater than one, the discussion in Section 4.3.2.3 must be expanded to include some additional considerations. The two-channel system will be used to illustrate these points. The equations of detailed balance are

$$\dot{P}_0(t) = -2\rho P_0 + P_1$$

$$\dot{P}_1(t) = 2\rho P_0 - (2\rho+1)P_1 + 2P_2$$

$$\dot{P}_n(t) = 2\rho P_{n-1}(t) - (2\rho+2)P_n(t) + 2P_{n+1} \qquad n > 1 \qquad (69)$$

$$\vdots$$

For the two-channel system, the equation analogous to Eq. (52a) is a function of $P_0(s)$ and $P_1(s)$.

$$2P_0(s) + \alpha_2(s)P_1(s) = \frac{\alpha_2(s)}{1 - \alpha_2(s)} \qquad (70)$$

One additional equation in terms of $P_0(s)$ and $P_1(s)$ is obtained by taking the Laplace transform of the first equation in set (69):

$$-(s+2\rho)P_0(s) + P_1(s) = -1 \qquad (71)$$

when:

$$P_0(0) = 1$$

Solving Eqs. (70) and (71) for $P_0(s)$, one obtains the desired equation for $P_0(s)$:

$$P_0(s) = \frac{\alpha_2(s)}{2\rho + 2}\left[\frac{1}{1 - \alpha_2(s)} + \frac{2\rho\alpha_2(s) + 2\rho - 2}{2\rho\alpha_2^2(2) - 2\alpha_2(s) + 4}\right] \qquad (72)$$

The technique is to expand the second term in partial fraction, i.e., $2\rho\alpha_2^2(s) - 2\alpha_2(s) + 4$ has a pair of complex roots $\alpha_2^1(s)$ and $\alpha_2^2(s)$, and take the inverse transform term by term. Again, the solutions involve series of modified Bessel functions similar to Eq. (53).

As the number of channels is increased, the problem becomes increasingly complex, since a set of M equations must be solved for $P_0(s)$ and the characteristic equation for the determinant of this set of equations must be solved for the roots $\alpha_2^1(s) \ldots \alpha_2^m(s)$. Each of the terms in the partial fraction expansion transforms into an infinite series of modified Bessel functions. Therefore the approach has been to approximate the infinite-queue case by a finite number of equations of detailed balance and then to obtain the transient solutions by a direct solution of the equations of detailed balance.

4.3.3. SUMMARY

The steady-state and transient solutions for the single- and multiple-exponential channel service facilities have been discussed in this section. We see that queueing theory models give results in two areas: (a) the number of units in the system queue and service channels, and (b) the time required to pass through the queue and the service channel. Each can be described by the moments of their respective distributions. Although the steady-state statistics have been emphasized in this section, many of these equations can be applied under transient conditions by replacing P_n with $P_n(t)$. This technique and other formulas developed in this section will be applied in the remainder of this report.

4.4. Application of the Exponential Model to the Polymorphic Computer

Several of the formulas developed for the exponential service facility in Section 4.3 will now be applied to the polymorphic computer as represented by this simple model. A number of measures of system performance can be used to describe a service facility; mean waiting time and system utilization are particularly well suited to a computer system. If one assumes equivalent environments, it is evident that increasing the number of job computers (channels) will reduce the waiting time per job request. The "price" which nature demands for this customer benefit is reduced system utilization. The purpose of this brief section is to point out the relationships between these conflicting measures of system performance.

Consider the mean waiting time T_W for the M = 1-, 2-, 4-, 6-, and 7-channel system operating in the same environment. It was pointed out in Section 4.3.2.3. (page 77), that systems having the same mean number of jobs in service L_S have equal arrival rates λ. The mean waiting time T_W, as a function of the environment L_S, has been plotted in Figure 26. T_W is normalized with respect to the mean interval between arrivals T_a. Since the queue is assumed to be finite, the system utilization factor (s.u.f) is equal to ρ. Therefore, the mean waiting time and the utilization factor can be tabulated (Table VII) for various values of L_S.

It is clear that the mean waiting time can be reduced by an order of magnitude by reducing the system utilization by a factor of two. Given a specified environment, $L_S = 3.5$, the mean waiting time is reduced from 5.0 to .25 by dropping the utilization factor from .875 to .583.

The waiting-time distribution for a multiple-channel, infinite-queue system is given in Section 4.3.2.2, Eq. (66). If t is normalized with respect to the mean service time T_S for a single channel, the waiting-time distribu-

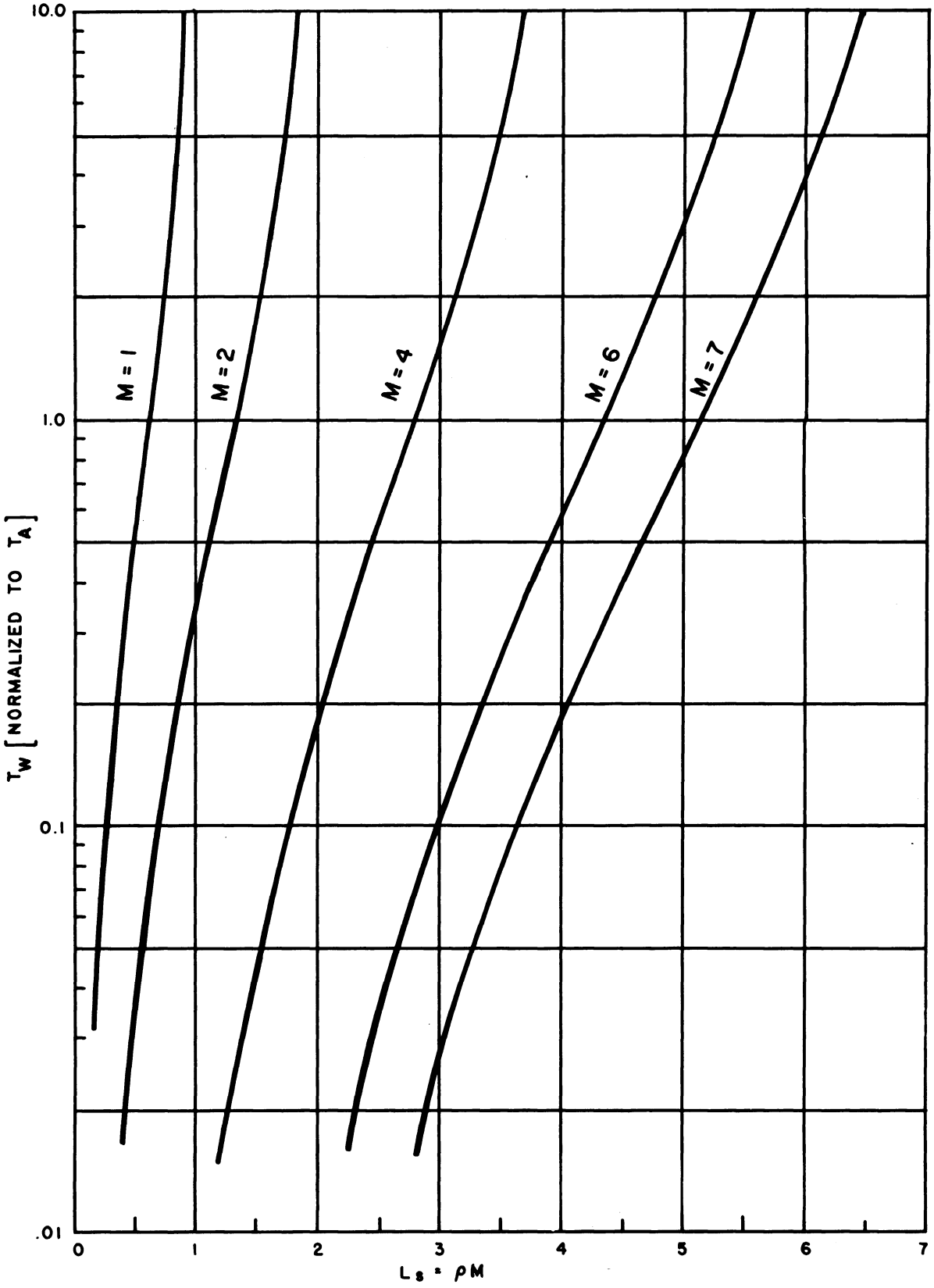

Figure 26. Mean waiting time vs. mean number in service (N=∞).

TABLE VII

SYSTEM PERFORMANCE CHART

L_s	M	T_w (Normalized to T_a)	System Utilization Factor
1.6	2	.67	.8
1.6	4	.0158	.4
3.5	4	5.0	.875
3.5	6	.25	.583
3.5	7	.08	.500
5.0	6	3.0	.834
5.0	7	.78	.715

tion can be written as

$$P(t_w \geqslant t) = Q_M e^{-M(1-\rho)t} + [1-Q_M] \qquad t \geqslant 0 \qquad (73)$$

where:

$$Q_M = \sum_{n=1}^{\infty} P_{M+n};$$

$$Q_M = \sum_{M=1}^{\infty} P_M \text{ the probability that all channels are busy;}$$

$$1-Q_M = \sum_{n=0}^{M} P_n \text{ the probability that the queue is empty; and}$$

$$\rho = \lambda/M\mu.$$

Assuming equal environments L_s, it is instructive to plot the waiting-time distributions for some multiple-channel systems (Figures 27a-d). The probability of immediate service, $1-Q_M$, has been omitted from these figures. This contribution to the waiting-time distribution, arising from the $\delta(o)[1-Q_M]$ term in the density function, is included in Table VIII.

Each curve in Figures 27a-d is identified by M and ρ. Since ρ is equal to the system utilization factor, the curves again show the trade-off between waiting time and system utilization. For example, assume that the environment places a requirement upon the system that the mean number of jobs in service is equal to 1.6 under steady-state conditions (Figure 27b). We want to determine the value of t for which the probability of waiting t mean service times or more is .01 (i.e., $P(t_w \geqslant t) = .01$) and the system utilization

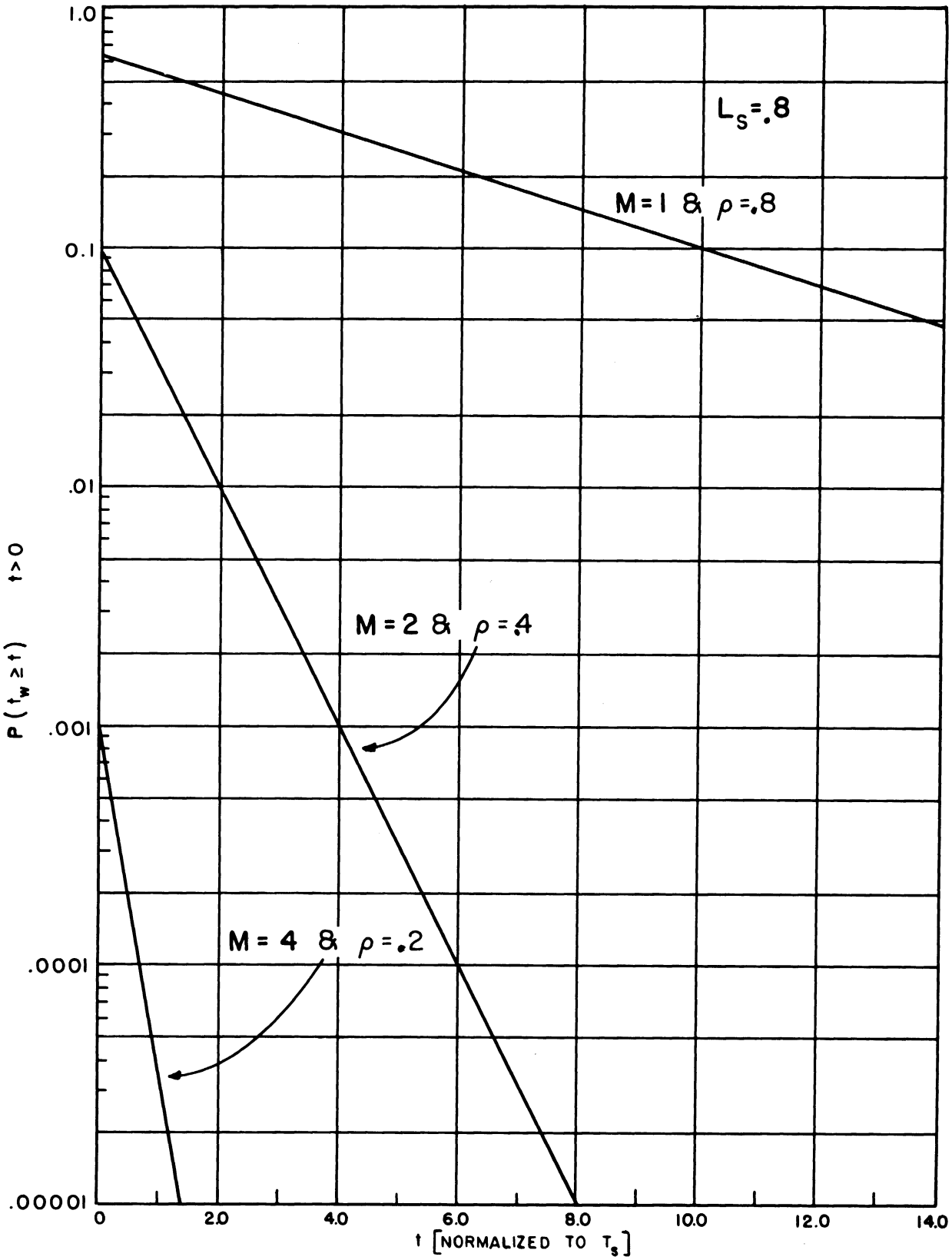

Figure 27a. Waiting-time distributions for multiple-channel systems.

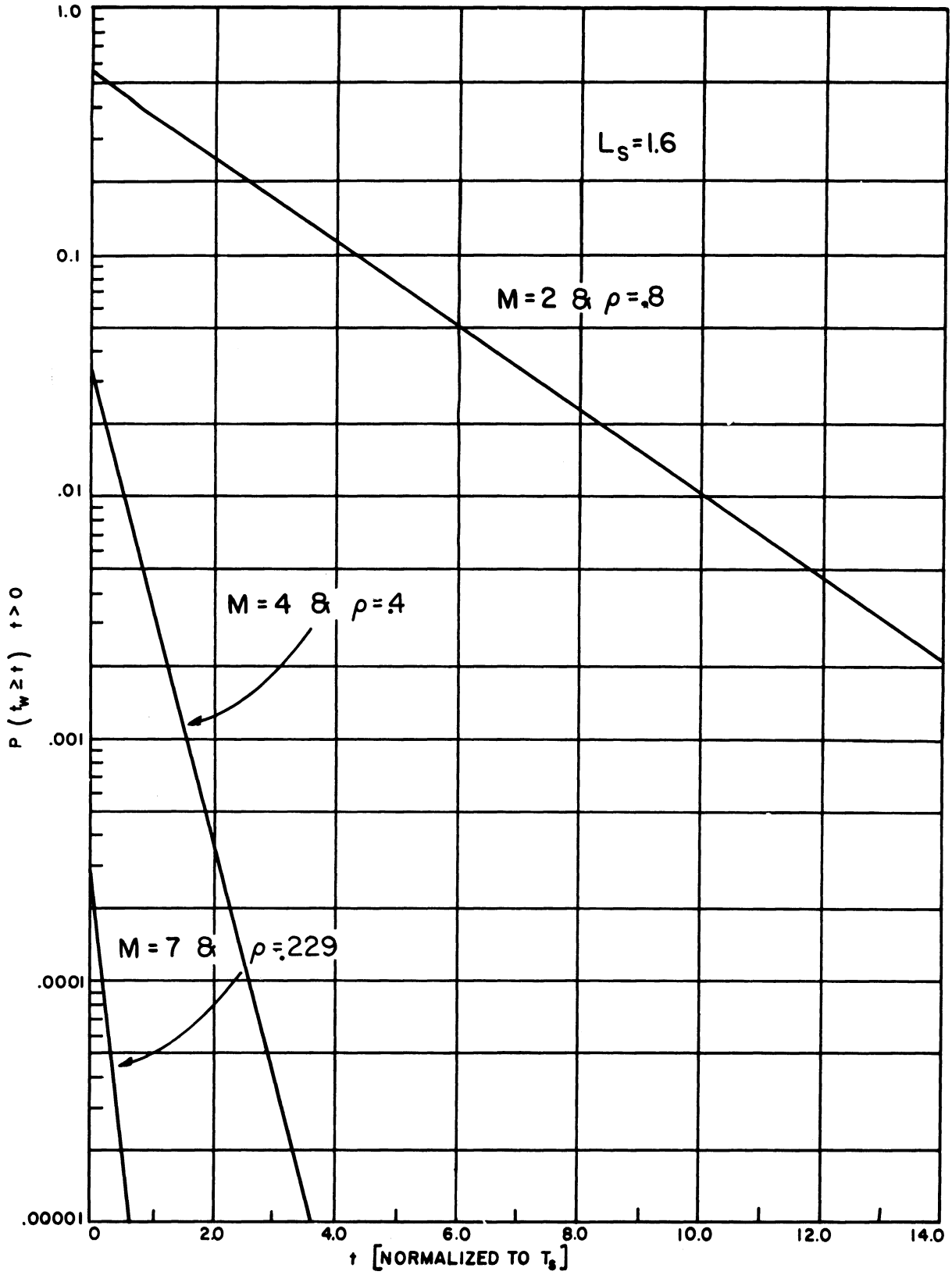

Figure 27b. Waiting-time distributions for multiple-channel systems.

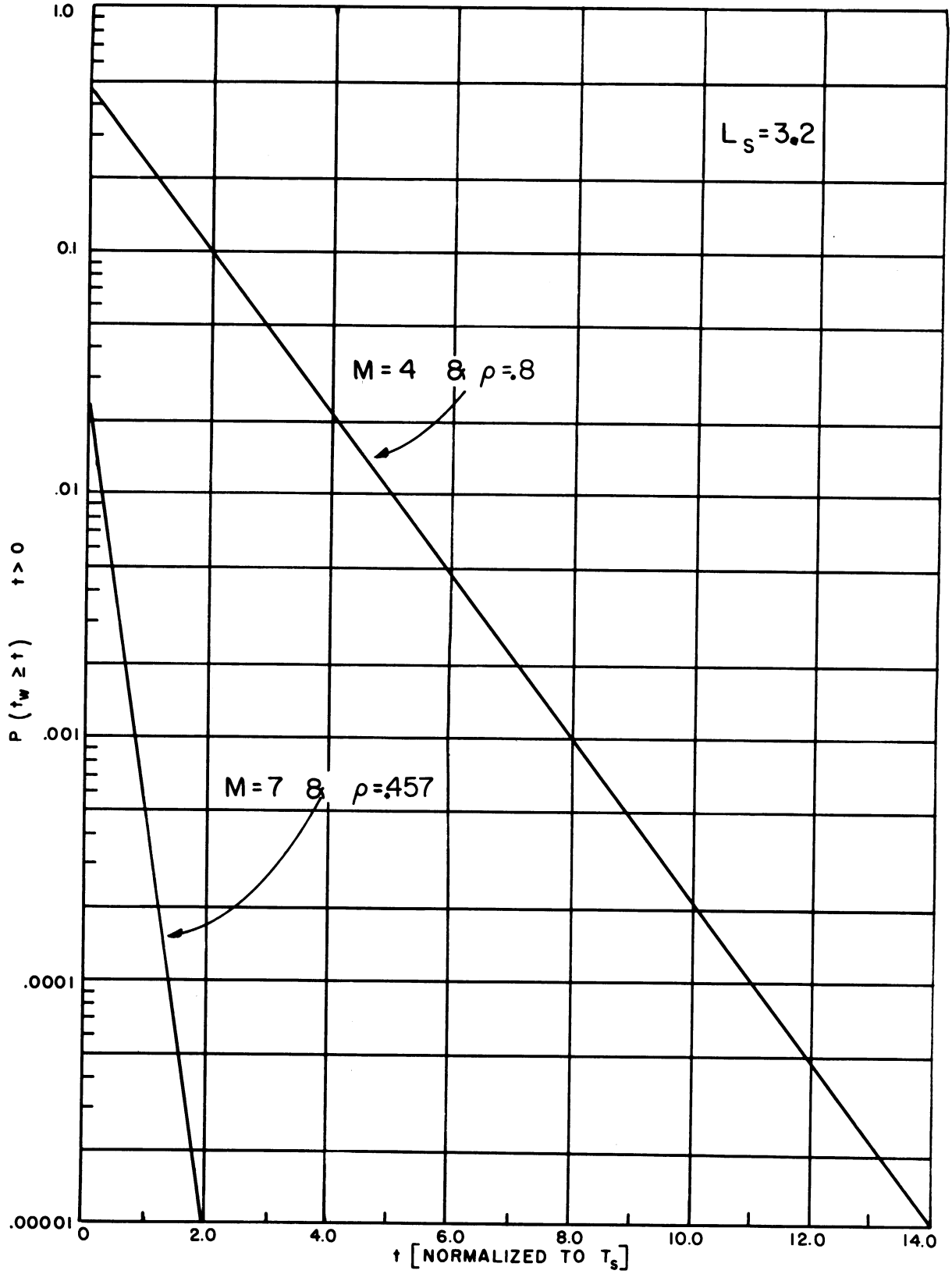

Figure 27c. Waiting-time distributions for multiple-channel systems.

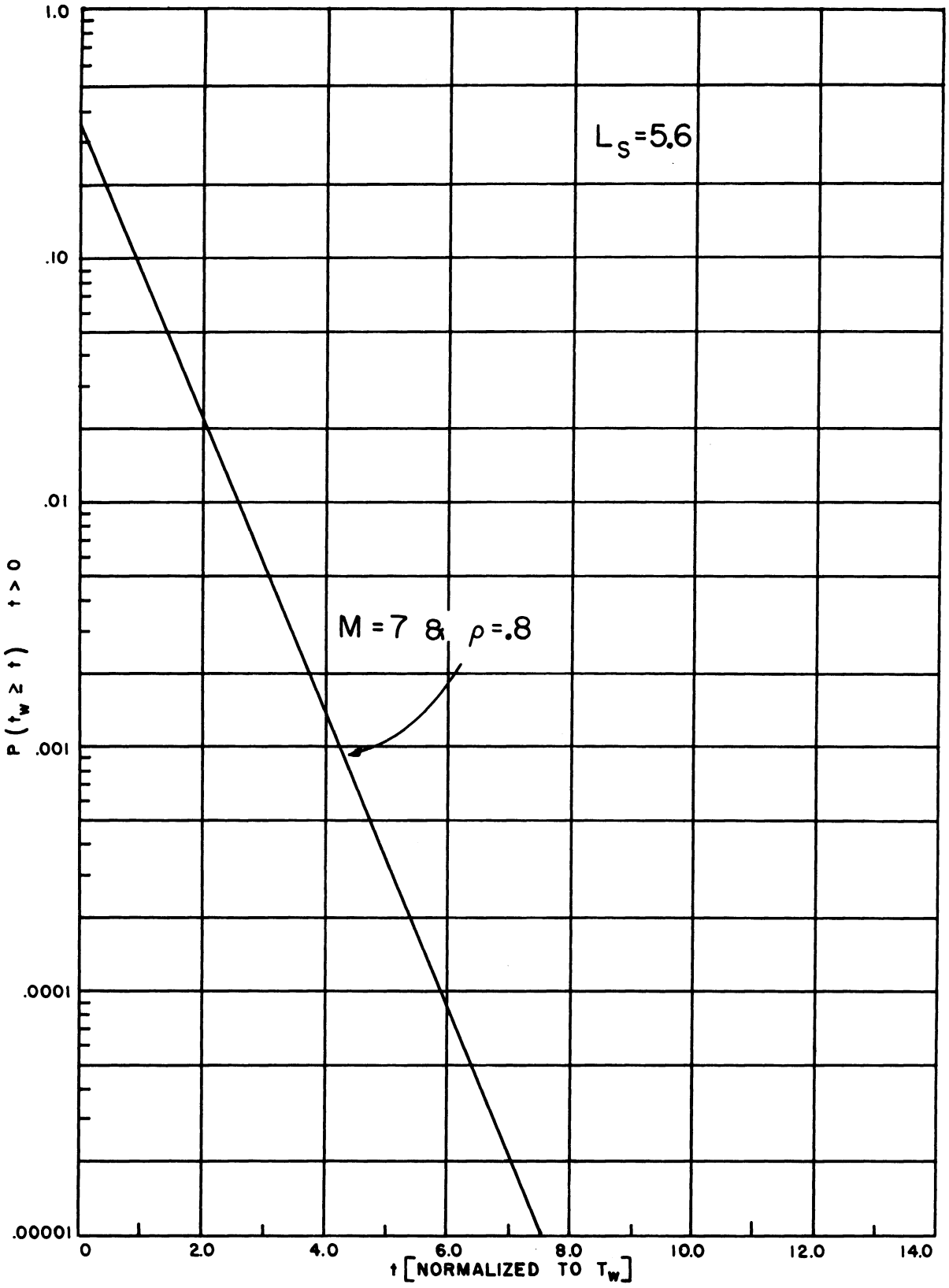

Figure 27d. Waiting-time distributions for multiple-channel systems.

factor for each system is considered. Table IX is the system performance chart. Thus, the addition of two channels reduces the critical waiting time

TABLE VIII

THE PROBABILITY OF IMMEDIATE SERVICE

M	L_S	$P[t_w = 0]$
1	.8	.3600
2	.8	.9090
4	.8	.9981
2	1.6	.4320
4	1.6	.9638
2	1.6	.9997
4	3.2	.524
7	3.2	.977
7	5.6	.615

TABLE IX

SYSTEM PERFORMANCE CHART. $P(t_w \geqslant t) = .01$

L_S	M	t	System Utilization Factor
1.6	2	10.1	.8
1.6	4	.5	.4
1.6	7	0	.229

t by a factor of twenty, but the utilization, on all four-job computers, drops to .4. The trade-off between delay and utilization is essentially the same regardless of the measure of system performance which might be employed.

The final selection of the number of job computers M is based upon a relative weighting between the benefit to the customer, reduced delay, and penalty for a reduced utilization factor on the job computers. (System and job-computer utilization are equivalent terms because the queued job requests are divided equally among the job computers.) It is evident that M must be great enough to handle the environment L_S. However, the number of additional channels depends on the relative weighting between utilization and delay which is, in turn, dependent upon the system application. For

example, a university computing center, which is maintained on a limited budget, would stress system utilization over the delay in processing student problems.

In the next two Sections, 4.5 and 4.6, two additional queueing problems are considered. The first deals with storage saturation and the second with competition for a shared module.

4.5. First Passage on the n^{th} State

The measures of performance which have been described so far for the multiple-channel system are those most often discussed in queueing theory literature. However, if the penalty for storage saturation is very high (e.g., if the saturating request is destroyed), an equally important measure is the probability that the system will reach a given state n for the first time, assuming that the system was in state zero at t = 0. It is evident that the probability of nonsaturation is equal to one minus the first-passage probability, on the n^{th} state for an N-state limited system. As in the previous problems, the service and arrival-time distributions are assumed to be exponential and the queue discipline is first come, first serve. A typical "system-state waveform" is shown in Figure 28. The time $t_{o,n}$ required to reach the n^{th} state is indicated on the figure. The time to reach saturation, $t_{o,N}$ is just a special case of the above.

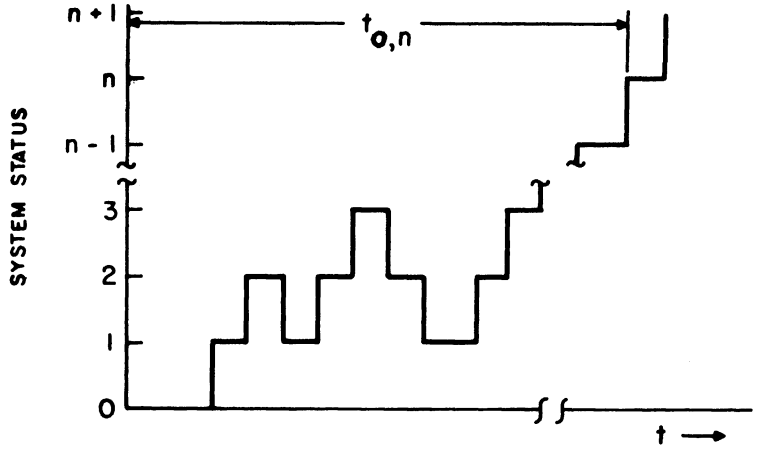

Figure 28. First-passage to the n^{th} state.

The equations of detailed balance and their solutions are summarized in Section 4.5.1. Some additional queueing problems of the same general type are discussed in Section 4.5.2.

4.5.1. SOLUTION FOR FIRST-PASSAGE PROBABILITY DISTRIBUTION

The equations of detailed balance for a seven-channel, infinite-queue system are:

$$\begin{aligned}
\dot{P}_0(t) &= -\lambda P_0(t) + \mu P_1(t) \\
\dot{P}_n(t) &= \lambda P_{n-1}(t) - (\lambda+n\mu)P_n(t) + (n+1)\mu P_{n+1}(t) && n < 7 \\
\dot{P}_n(t) &= \lambda P_{n-1}(t) - (\lambda+7\mu)P_n(t) + 7\mu P_{n+1}(t) && n \geq 7
\end{aligned} \tag{74}$$

where:

λ = average arrival rate;

μ = average service rate for a single service channel; and

$\rho = \lambda/7\mu$.

The above equations can be written in normalized form as

$$\begin{aligned}
\dot{P}_0(t) &= -7\rho P_0(t) + P_1(t) \\
\dot{P}_n(t) &= 7\rho P_{n-1}(t) - (7\rho+n)P_n(t) + (n+1)P_{n+1}(t) && n < 7 \\
\dot{P}_n(t) &= 7\rho P_{n-1}(t) - (7\rho+7)P_n(t) + 7P_{n+1}(t) && n \geq 7
\end{aligned} \tag{75}$$

To determine the first-passage distribution, an absorbing barrier is placed at the n^{th} state. That is, transitions <u>from</u> the n^{th} or higher states are not allowed and, as a result, transitions among the states 0, 1,..., n-1 will terminate when the n^{th} state is reached for the first time. Notice, however, that transitions <u>to</u> the n^{th} state are allowed. For example, the rate of change of the probability of being in the 10^{th} state is given by

$$\dot{P}_{10}(t) = 7\rho P_9(t) \tag{76}$$

Also, $\dot{P}_9(t)$ is modified by the restriction that no transitions from the 10^{th} state, resulting from service completions, can occur.

$$\dot{P}_9(t) = 7\rho P_8(t) - (7\rho+7)P_9(t) \tag{77}$$

The remaining equations remain unchanged. Thus the infinite set of equations required for the infinite-queue case become a finite set of n+1 equations when a barrier is place at the n^{th} state.

In general, for $n \geq 8$ the equations are of the form

$$\dot{P}(t)\,] = [U] \cdot P(t)\,] \tag{78}$$

where:

$$\dot{P}(t)\,] = \begin{bmatrix} \dot{P}_0(t) \\ \dot{P}_1(t) \\ \vdots \\ \dot{P}_{n-1}(t) \\ \dot{P}_n(t) \end{bmatrix} \quad P(t)\,] = \begin{bmatrix} P_0(t) \\ P_1(t) \\ \vdots \\ P_{n-1}(t) \\ P_n(t) \end{bmatrix} \quad P(0)\,] = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$[U] = \begin{bmatrix} -7\rho & 1 & & & & \\ 7\rho & -(7\rho+1) & 2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & 7\rho & -(7\rho+7) & \\ & & & & 7\rho & \end{bmatrix}$$

Notice that in Eq. (78) the initial conditions are specified by column vector P(0), and in the case considered here the system is always initially in state zero, $P_0(0) = 1.0$.

The time-dependent or "transient" solution of Eq. (78) provides

$P_n(t)$ = $P(t_{o,n} \leqslant t)$, the probability-distribution function on the first-passage time; and

$\dot{P}_n(t)$ = $p(t_{o,n})$, the probability-density function on the first-passage time, $t_{o,n}$.

Numerical solutions of these equations have been obtained for $P(t_{o,n} \leqslant t)$ with $n = 7, 10$ and $\rho = 0.5, 0.7, 0.9$, and also for $n = 20$ and $\rho = 0.7, 0.9$. These are shown in Figures 29-32.

For small $n(n \leqslant 7)$ it is evident from Figure 29 that the system is certain to reach capacity (state 7 in a 7-channel system) in a few service times regardless of the value of ρ. As one considers first-passage to higher and higher states (Figures 30 and 31), the parameter ρ becomes increasingly important. The result is anticipated when the dependence of the steady-state state probability on ρ is considered for a given value of n (see Table X). The P_n's are nearly equal for small values of n (say $n = 7$) while they differ by orders of magnitude for increased n.

TABLE X

STEADY-STATE STATE PROBABILITIES FOR AN INFINITE-QUEUE, SEVEN-CHANNEL SYSTEM

States	$\rho = .5$	$\rho = .7$	$\rho = .9$
$n = 7$	.0380993	.090221	.0671113
$n = 10$	.0047624	.0309457	.0489256
$n = 20$	.000046509	.000874145	.0170593
$n = 40$	----	----	.002074010

4.5.2. ADDITIONAL PROBLEMS

First-passage time distributions from initial states other than zero can be obtained by setting $P_i(0) = 1$, $i < n$, in Eq. (78). That is, first-passage to the state n, assuming that the system is initially in state i, is obtained by a simple change of the initial conditions.

A second possibility is to put the absorbing barrier at some low level. Saaty (Ref. 3) determines the distribution for the system busy period $P(t_{1,o})$, infinite-queue case, by placing the barrier at $n = 0$ and setting $P_1(0) = 1$. The busy period is the interval between an arrival to an empty

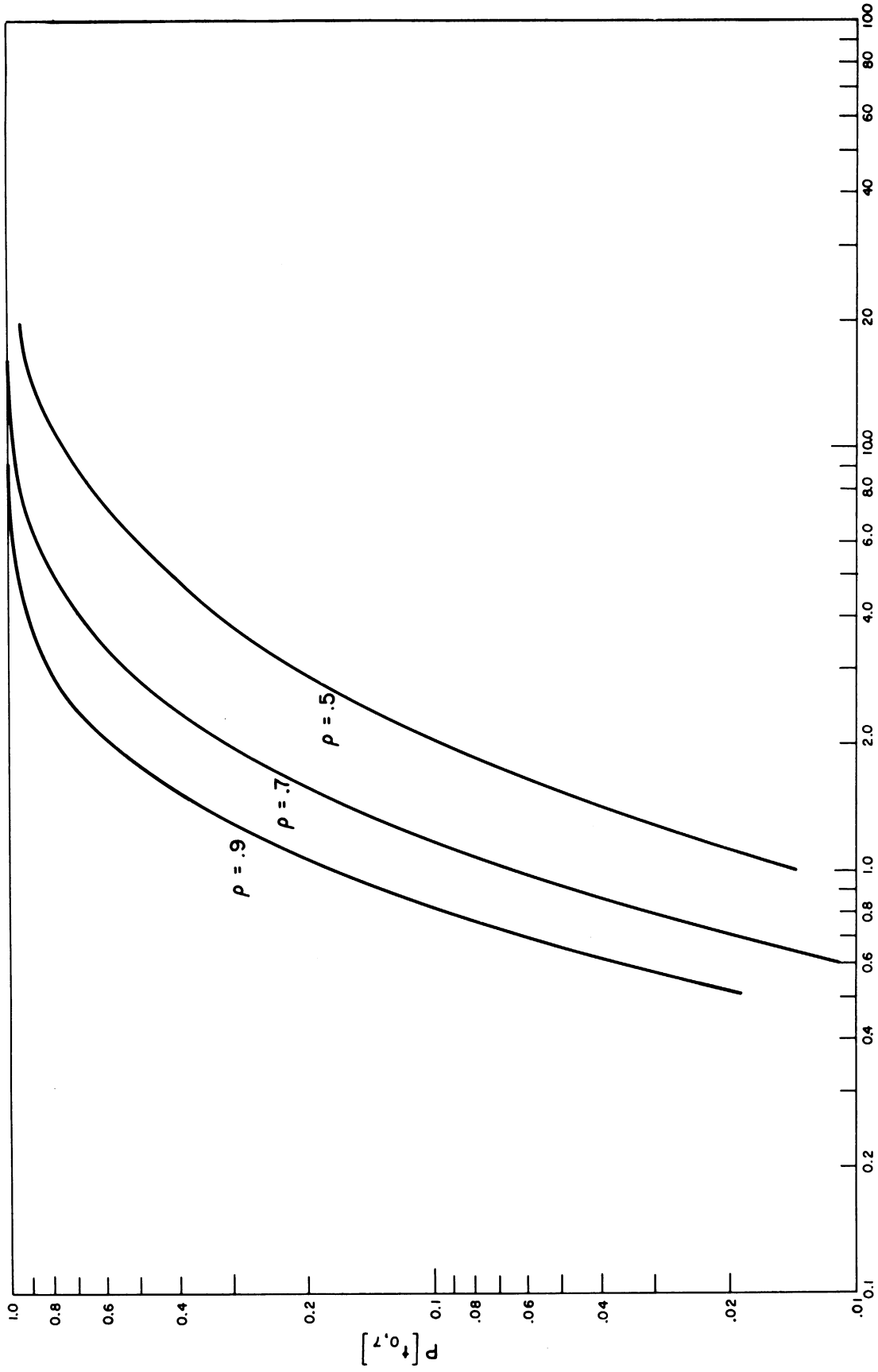

Figure 29. First-passage probability to state n = 7 for ρ = .5, .7, .9.

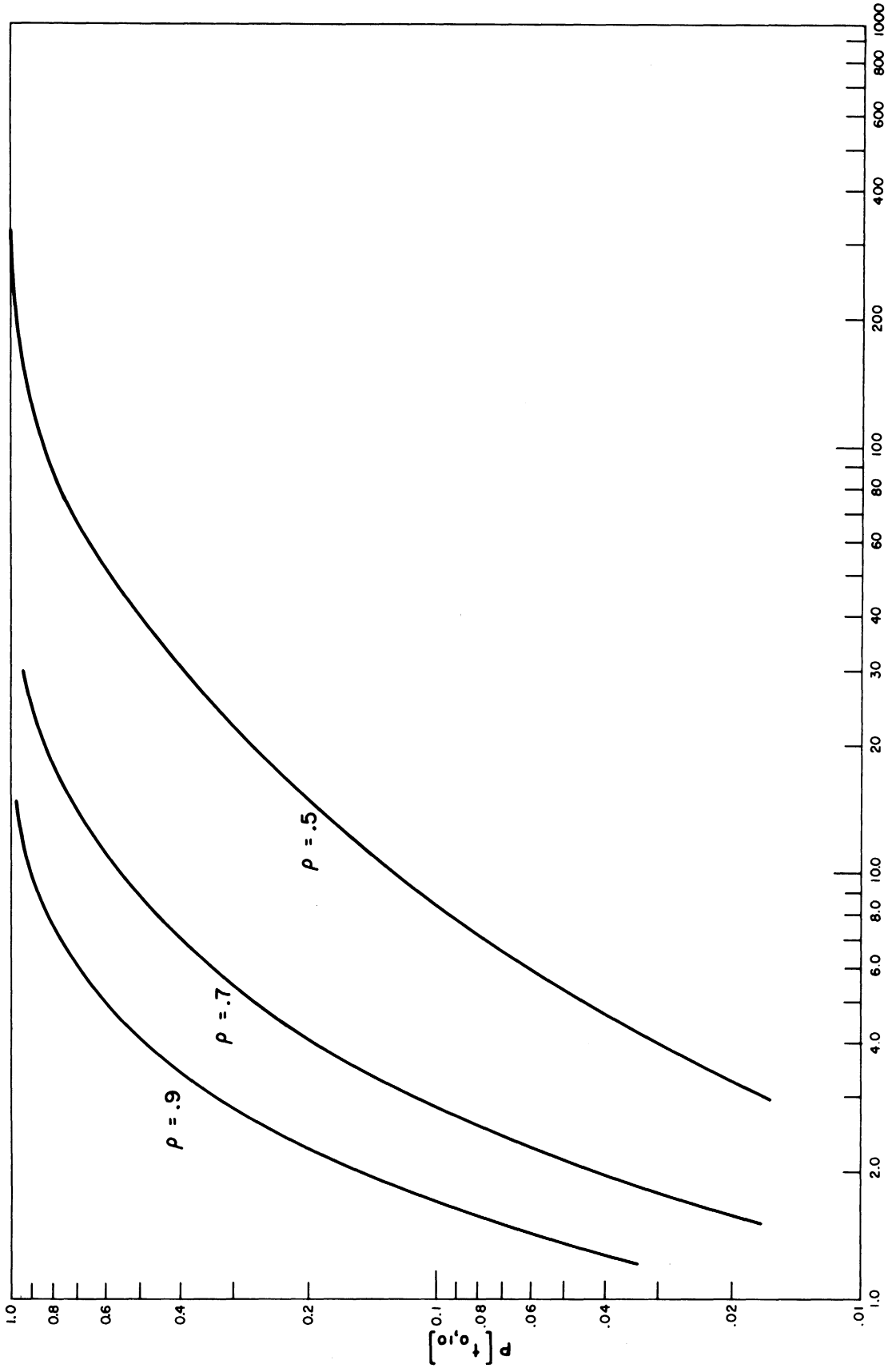

Figure 30. First-passage probability to state n = 10 for ρ = .5, .7, .9.

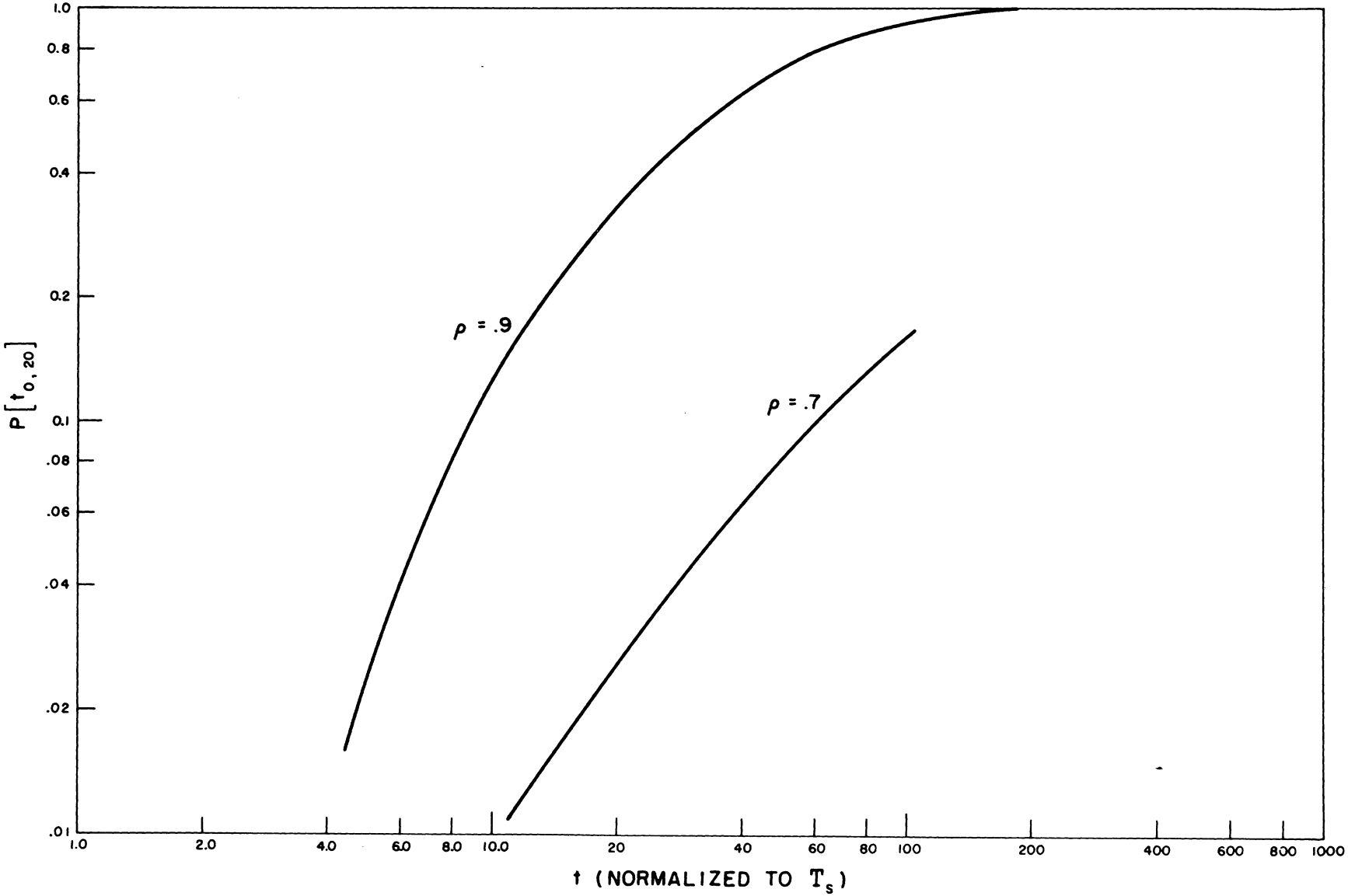

Figure 31. First-passage probability to state n = 20 for ρ = .9, .7.

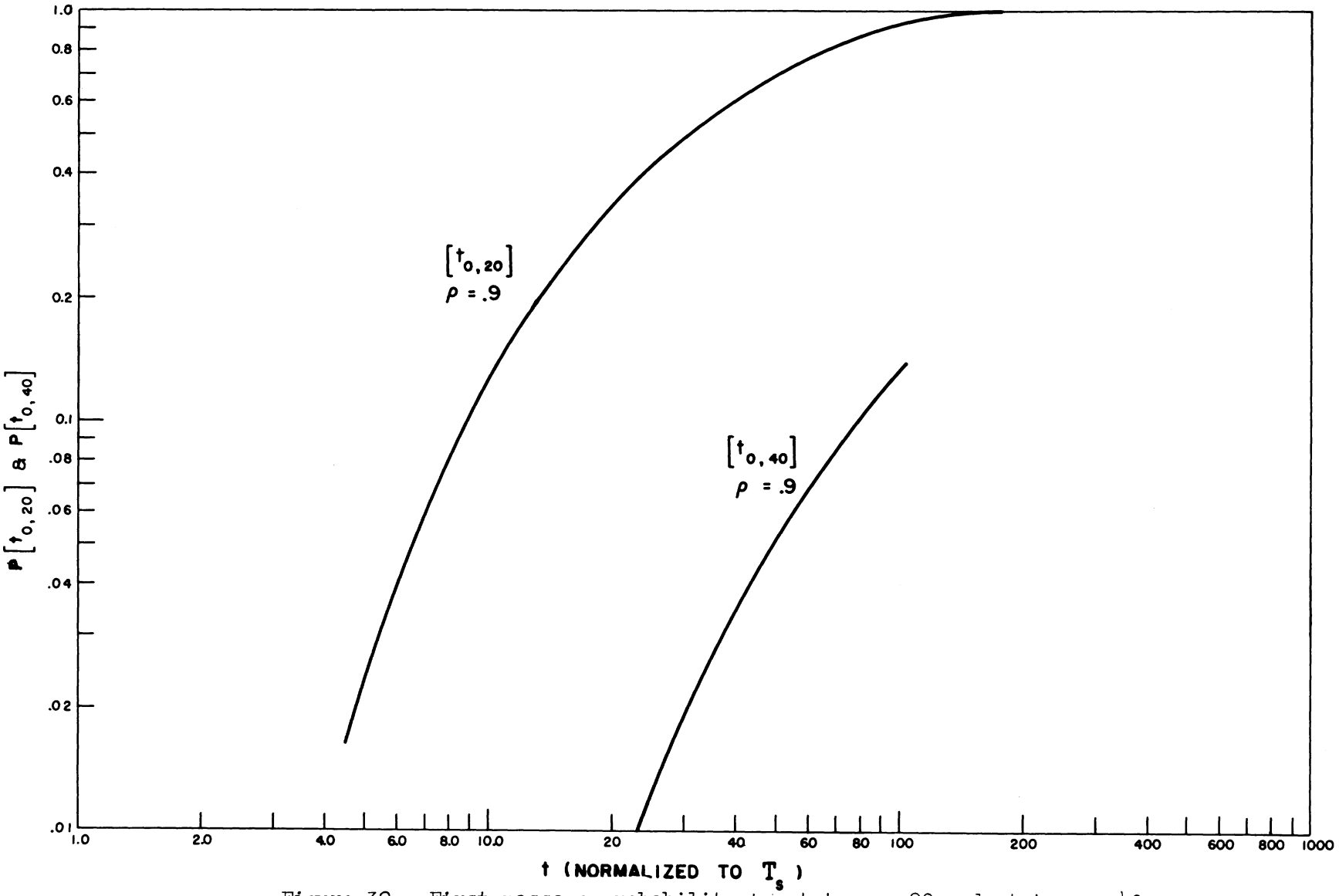

Figure 32. First-passage probability to state n = 20 and state n = 40 for ρ = .9.

system and the next time the system is empty. For the infinite-queue case the number of equations remains infinite. However, if the maximum number of states is limited (finite queue), the only modification on Eq. (78) is in the [U] matrix. Leading rather than trailing terms are modified.

It is apparent, therefore, that by controlling the initial conditions and the location of the absorbing barrier, a large variety of first-passage probability distributions may be obtained, depending upon the initial conditions and the barrier location.

4.5.3. SUMMARY

The equations and some typical solutions have been presented for a seven-channel system (assuming exponential arrivals and services). The probability of saturating storage, first-passage on the N^{th} state, decreases for large N and reduced system utilization ρ. The emphasis placed upon this measure of system performance is a function of the penalty associated with storage saturation, which is, again, dependent upon the system application. Although the major application here is to the storage saturation problem, one can obtain the first-passage time distribution to any desired state n of interest.

4.6. Job Computer Queueing for Use of Shared Modules

Recall, from the description of the AN/FSQ-27 system in Sections 1 and 2, that a job computer and other subordinate modules are assigned to process each job request. However, some modules are not assigned exclusively to one job, but are shared by all "in progress" job requests. Consequently, there is competition or queueing of job computers for the shared modules (e.g., the master drum). This queueing of job computers for use of a shared subordinate module in a polymorphic system corresponds to the channel machine maintenance problem in queueing theory. The finite number of job computers, N, which request use of the shared module, are analogous to the machines that breakdown. The shared module is analogous to the repair crew that must service the failed machines.

In the representation of Figure 33, requests for service are units that circulate in the system, and job computers are the arrival generating channels. Each request unit, upon completing service, returns to the arrival channel (job computer) from where it originated, and experiences a delay of random duration before entering the queue again for another service.

The solution of this queueing problem is discussed in this section. The queue discipline is first come, first served, and the arrival and service-time distributions are either both exponential or both constant.

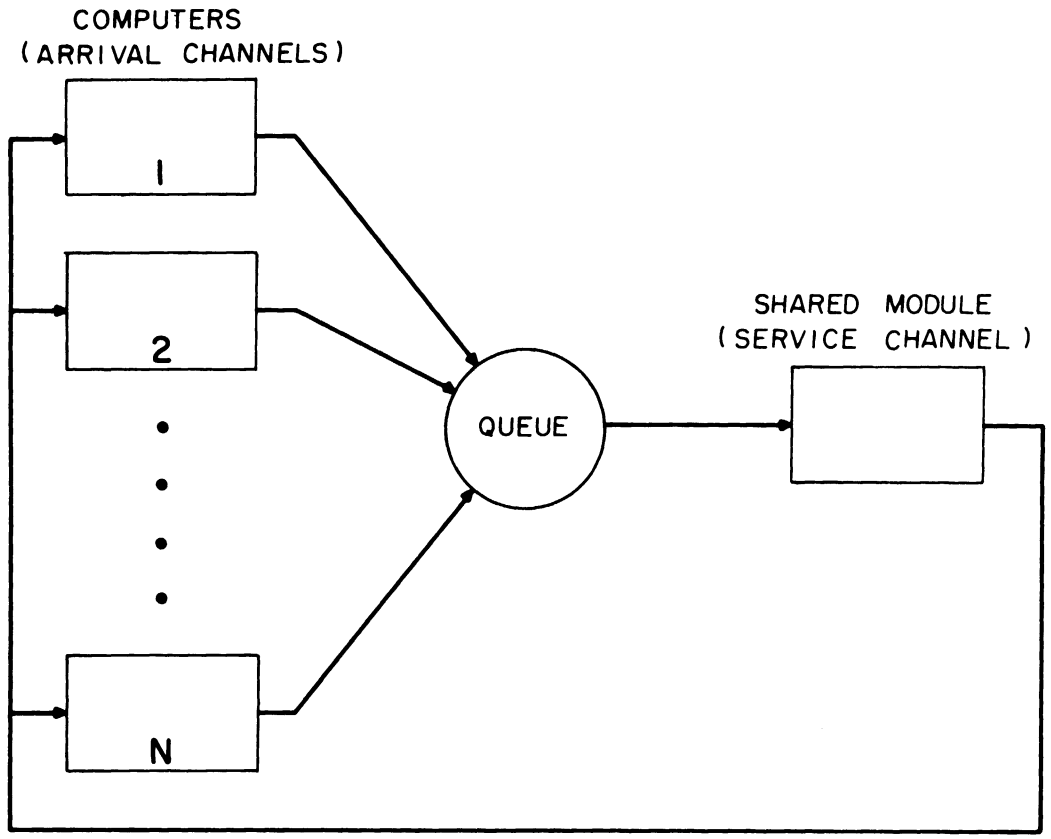

Figure 33. Representation of shared module queueing.

4.6.1. SOLUTION FOR CONSTANT ARRIVAL AND SERVICE TIMES

The case of constant arrival and service times with first come, first served queue discipline can be readily solved without resorting to differential equations. Let

T_s = service time for a computer by the shared module (same for all computers);

T_a = time between completion of a service by the shared module and the next request for service from the same job computer; and

$\rho = T_s/T_a$.

and let the job computers be numbered 1 through N, as shown in Figure 33. Assume that the system begins operating with each computer making a request for use of the module, in turn according to its number, with only a small time increment between each request. The first computer to have use of the module will be number 1, and after an interval T_s job computer 2 will obtain

use of the module, etc. Now job computer 1 will request use of the module again in time T_a after completion of its first service by the shared module. It is clear that if $T_a \geqslant (N-1)T_s$, job computer 1 will experience no waiting time on its second request, nor on succeeding requests, and furthermore none of the computers will experience additional waiting time. This is because after the first round, the request times of the computers will be in a phase relationship which does not change thereafter, and for which there is no waiting time if $T_a \geqslant (N-1)T_s$.

On the other hand, if $T_a < (N-1)T_s$, the waiting time per request for each computer will be the difference of $(N-1)T_s$, the time to service the other computers, and T_a, the interval since its last request. Hence

$$W_q = (N-1)T_s - T_a = T_s[(n-1) - \frac{1}{\rho}] \qquad \rho > \frac{1}{N-1} \qquad (79)$$

$$W_q = 0 \qquad \rho \leqslant \frac{1}{N-1} \qquad (80)$$

Figure 34 gives a plot of $L_q = W_q/T_s$ versus ρ for the case of N = 7 computers with constant arrival and service times.

4.6.2. EXPONENTIAL ARRIVAL-AND SERVICE-TIME DISTRIBUTION

The solution of this queueing problem for exponential arrival and service-time distributions is given in a number of references (Refs. 1 and 6). The time-dependent equations for the state probabilities (the probabilities of having a given number of computers queued for the shared module) are as follows for N = 7:

$$\begin{bmatrix} \dot{P}_0 \\ \dot{P}_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \dot{P}_7 \end{bmatrix} = \begin{bmatrix} -7\lambda & & & & & & \\ 7\lambda & -[\mu+6\lambda] & \mu & & & & \\ & 6\lambda & -[\mu+5\lambda] & \mu & & & \\ & & 5\lambda & -[\mu+4\lambda] & \mu & & \\ & & & 4\lambda & -[\mu+3\lambda] & \mu & \\ & & & & 3\lambda & -[\mu+2\lambda] & \\ & & & & & 2\lambda & -[\mu+\lambda] \quad \mu \\ & & & & & & \lambda \quad -\mu \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ P_7 \end{bmatrix}$$

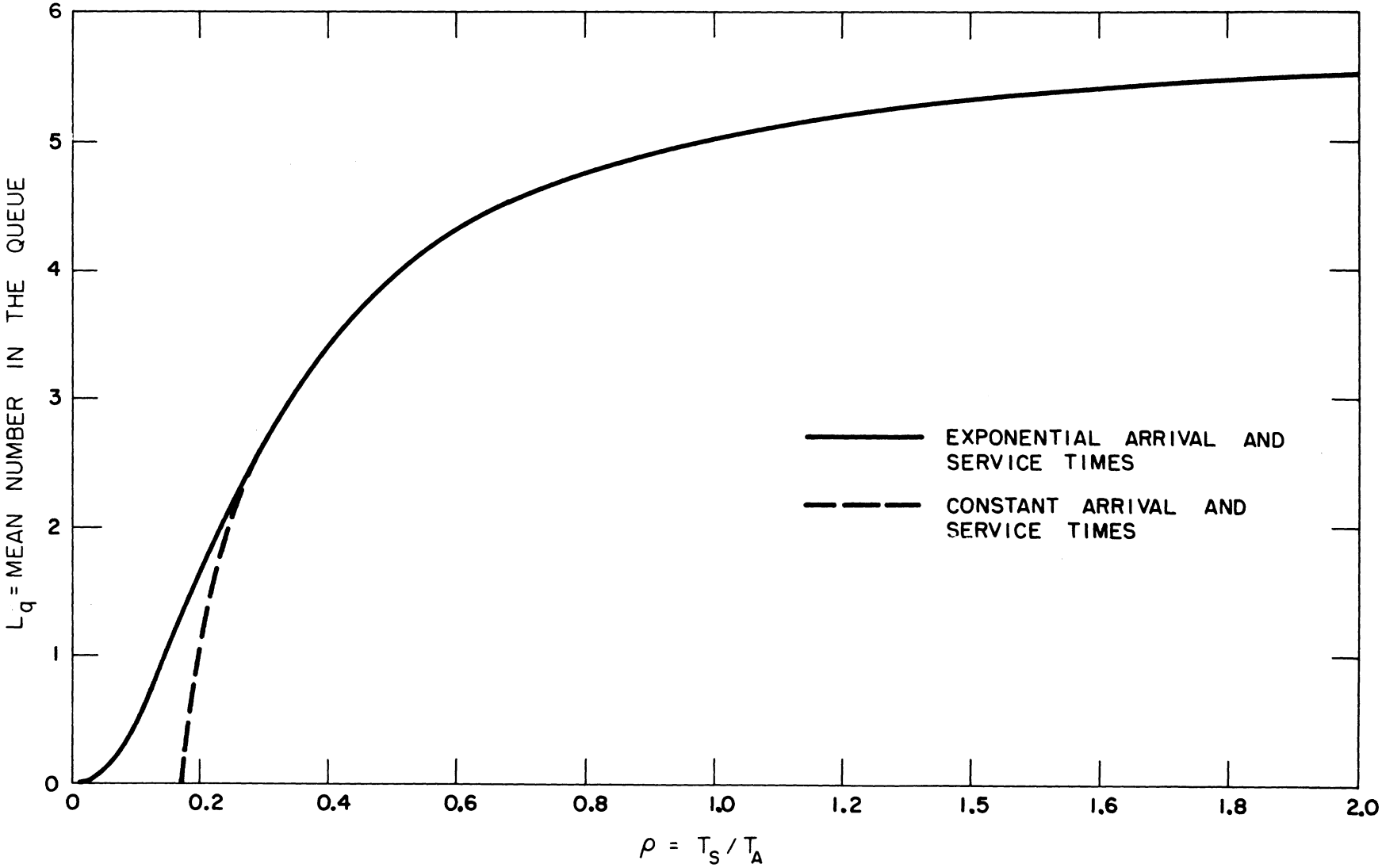

Figure 34. Mean number in queue vs. ρ (M = 7).

The steady-state solution of these equations is obtained by setting the derivatives $\dot{P}_0, \dot{P}_1, \ldots, \dot{P}_7 = 0$. Then one finds:

$$P_n = \frac{\rho^n N!}{(N-n)!} P_0 \qquad 0 \leqslant n \leqslant N \tag{81}$$

where:

λ = mean arrival rate for a single computer;

μ = mean service rate by the shared module; and

$\rho = \lambda/\mu = T_s/T_a$.

The mean arrival rate λ for each computer is equal to $1/T_a$, where T_a is the mean time between completion of a service by the shared module and the arrival of the next request by the computer that was serviced. T_s, the mean service time by the shared module, is equal to λ/μ.

P_0 is found from the constraint,

$$\sum_{n=0}^{N} P_n = 1.0 \tag{82}$$

The mean number in the system, i.e., the mean number of job computers waiting for the shared module including the one which has it, is

$$L = \sum_{n=0}^{N} nP_n \tag{83}$$

and the number in the queue is

$$L_q = \sum_{n=1}^{N} (n-1)P_n = L - 1 + P_0 \tag{84}$$

The mean waiting time in the queue is

$$W_q = L_q T_s \tag{85}$$

References 1 and 5 provide tables of functions from which L_q can be calculated for a range of values of N, ρ, and M.

Figure 34 presents a curve of $L_q = \mu W_q$ versus the ratio of mean service time and mean arrival time per job computer, ρ. Figure 35 presents the same quantity plotted against the mean shared module utilization factor (s.m.u.f).

$$s.m.u.f = 1 - P_o \tag{86}$$

That is, s.m.u.f is the mean fraction of total time that the shared module is used. Also plotted in Figure 35 is the normalized average throughput time, T_p,

$$\frac{T_p}{T_a} = \frac{\lambda}{\mu}(L_q+1) = \rho(L_q+1) \tag{87}$$

which is the mean total delay experienced in using the shared module (waiting time + service time) in terms of the mean arrival interval, T_a.

Because job-computer waiting time for the shared module is not useful in furthering the completion of a job request, queueing for a shared module results in a decrease in the job-computer utilization factor, the fraction of the available time the job computer is available for computation (i.e., not waiting for the shared module). Figure 36 presents the mean utilization factor of a single job computer against the utilization factor of the shared module. The job-computer utilization factor (j.c.u.f) is given by

$$j.c.u.f = \frac{T_a + T_s}{T_a + T_s + W_q} = \frac{1 + \rho}{1 + \rho(L_q+1)} \tag{88}$$

It is evident from Figure 36 that an attempt to achieve very high utilization of a shared module may result in decreased system efficiency because of the resulting degradation in utilization factor of the job computers. Any procedure for sharing a subordinate module must therefore take into account this trade-off in utilization, and the relative weight to be given to job-computer utilization as opposed to shared-module utilization.

4.7. The Effect of Arrival- and Service-Time Distributions on System Behavior

The performance of a service facility as determined by the various measures which may be applied (e.g., mean number in the system or mean waiting time), depends to a great extend upon the nature of the service and arrival-time distributions. Most of the queueing theory literature deals with models

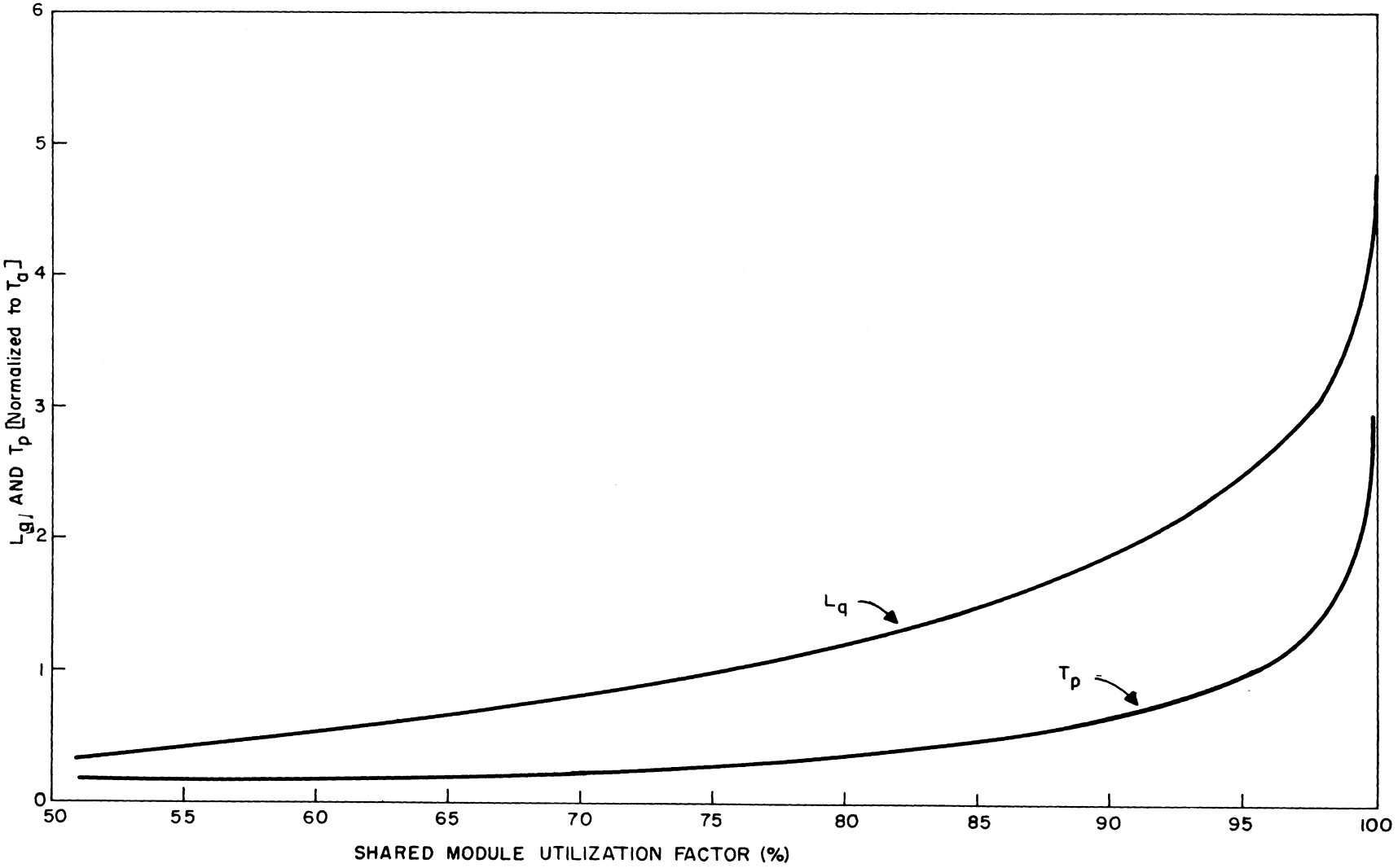

Figure 35. Queue length (L_q) and throughput time T_s (normalized to T_a) vs. the shared module use factor.

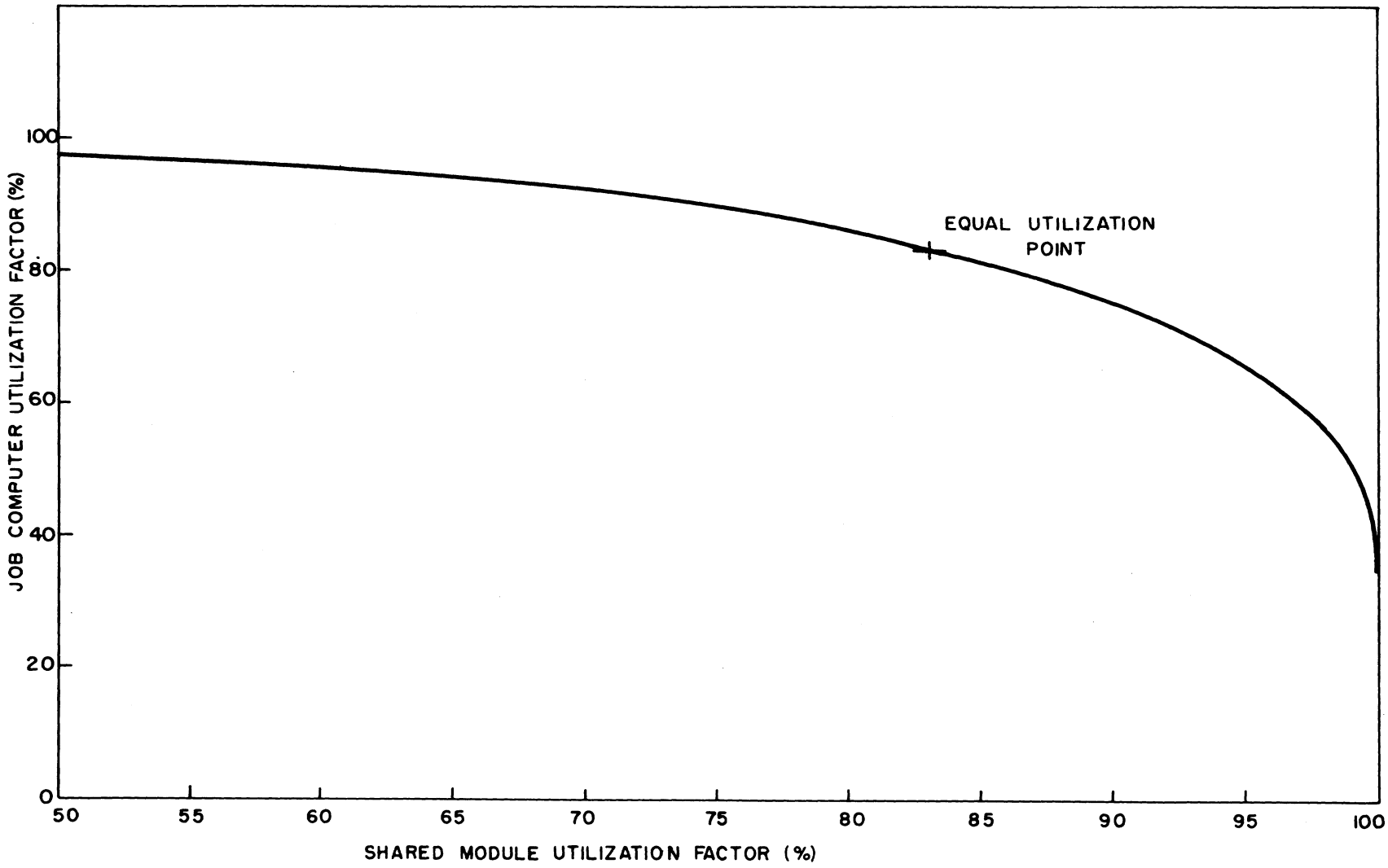

Figure 36. Job-computer vs. shared-module utilization.

in which these distributions are exponential, i.e., the probability density function of the time duration has the form

$$f(t)dt = \lambda\epsilon^{-\lambda t}dt \tag{89}$$

The purpose of this section is to discuss the relationship of system performance with exponential distributions to the performance with nonexponential distributions. Specifically the nonexponential distributions to be considered are the Erlang distribution of order k, and the Hyper-Exponential distribution. These distributions exhibit smaller and larger variance, respectively, than the exponential distribution. The questions to be considered are the effect of these distributions on the mean number in the system and on the transient solutions for the state probabilities. The entire discussion will deal only with the single-channel system.

4.7.1. AVERAGE NUMBER IN THE SINGLE-CHANNEL QUEUE SYSTEM, ARBITRARY ARRIVAL- AND SERVICE-TIME DISTRIBUTIONS

Under the assumption that a queueing process is stationary (i.e., that the expected number in the system does not vary with time), it is possible to derive a relationship for the mean number in the single-channel system with arbitrary arrival- and service-time distributions (Ref. 6, p. 336). This result is

$$E(n) = \rho + \frac{E(r^2)-\rho}{2(1-\rho)} \tag{90}$$

As in the previous sections:[6]

n = number in the system;

λ = mean arrival rate $(1/T_a)$;

μ = mean service rate $(1/T_a)$;

ρ = λ/μ; and

r = number of arrivals during the service for a single unit.

[6]Mean rates λ and μ describe arbitrary arrival and service distributions.

Equation (90) holds provided only that the arrival- and service-time distributions are stationary and independent of the number of units in the system n. The implication is that the system has an infinite queue. It is not possible to derive a similar equation for the multiple-channel system since the interval between service completions depends upon the number in service at any instant, which is unknown.

Let us now attempt to determine the relationship of $E(r^2)$ to the arrival- and service-time distributions. If we define the following,

$a(r|t)$ = probability of r arrivals in given time t (derived from the arrival-time distribution);

$s(t)dt$ = probability of a service time of length t (the incremental service-time distribution);

then

$a(r|t)s(t)dt$ = the joint probability of r arrival in time t and a service time t;

$\int_0^\infty r(r|t)s(t)dt$ = the probability of r arrivals during a service time;

and $E(r)$ and $E(r^2)$ are as follows:

$$E(r) = \sum_{r=0}^{\infty} r \int_0^\infty a(r|t)s(t)dt \tag{91}$$

$$E(r^2) = \sum_{r=0}^{\infty} r^2 \int_0^\infty a(r|t)s(t)dt \tag{92}$$

the mean number of arrivals in time t and the mean service time are

$$\sum_{r=0}^{\infty} ra(r|t) = \lambda t \tag{93}$$

$$\int_0^\infty ts(t)st = \frac{1}{\mu} = T_s \tag{94}$$

for all distributions $a(r|t)$ and $s(t)$.

Hence we can obtain, from Eq. (91),

$$E(r) = \int_0^\infty \left[\sum_{r=0}^{\infty} ra(r|t) \right] s(t)dt$$

$$E(r) = \int_0^\infty \lambda t s(t)dt = \lambda T_s = \rho \tag{95}$$

This result states that E(r) is the mean number of arrivals in a mean service time, which is intuitively obvious.

We also know that the following are true:

$$\sum_{r=0}^{\infty} r^2 a(r|t) = \mathrm{Var}(r|t) + (\lambda t)^2 \tag{96}$$

$$\int_0^\infty t^2 s(t)dt = \mathrm{Var}(t_s) + \left(\frac{1}{\mu}\right)^2 \tag{97}$$

where $\mathrm{Var}(r|t)$ is the variance of the number of arrivals in a specified interval t, and $\mathrm{Var}(t_s)$ is the variance of the service-time distribution. Then, using Eqs. (96) and (97), we obtain from Eq. (92),

$$E(r^2) = \int_0^\infty [\mathrm{Var}(r|t) + (\lambda t)^2] s(t)dt$$

$$E(r^2) = \frac{\lambda^2}{\mu^2} + \lambda^2 \mathrm{Var}(t_s) + \int_0^\infty \mathrm{Var}(r|t) s(t)dt \tag{98}$$

It is now clear that $E(r^2)$ is minimum when $\mathrm{Var}(t_s)$ and $\mathrm{Var}(r|t)$ are both zero, corresponding to constant arrival and service times. It is therefore the combination of constant arrival and service time which yields minimum expected number in the system. Unfortunately Eq. (90) cannot be used to determine E(n) for constant arrival and service times, because these distributions do not result in stationary solutions, which is the important assumption in the derivation of Eq. (90). But the above conclusion is not altered, since one can obtain a constant service-time distribution and an arrival distribution for which $\mathrm{Var}(r|t)$ is very close to zero. A stationary solution would then exist, and one could calculate E(n) from Eq. (90). Then, as one made $\mathrm{Var}(r|t)$ still closer to zero, one would arrive at the above conclusion.

It is also clear from Eq. (98) that one may reduce E(n) either by reducing the variance of the service-time distribution or by reducing the variance of the number of arrivals in a specified time. Hence, the general

conclusion may be drawn that arrival- and service-time distributions which are more random, in the sense of having larger variance, will result in longer queue lengths (in first come, first served queue discipline).

4.7.1.1. Exponential Arrival-Time Distribution

In this section we will compare the expected number in the single-channel system with an exponential arrival-time distribution and exponential, Erlang, or Hyper-Exponential service-time distributions. It would be desirable to also compare systems in which the arrival-time distributions were either Erlang or Hyper-Exponential. Unfortunately, the determination of $Var(r|t)$, which involves finding the r^{th} convolution of the arrival-time distribution, is difficult to carry out for these cases.

For exponential arrivals

$$a(r|t) = \frac{(\lambda t)^r}{r!} e^{-\lambda t} \tag{99}$$

and

$$Var(r|t) = \sum_{r=0}^{\infty} r^2 \frac{(\lambda t)^r}{r!} e^{-\lambda t} - (\lambda t)^2 = \lambda t \tag{100}$$

Hence, for exponential arrivals

$$E(r^2) = \rho^2 + \lambda^2 Var(t_s) + \rho \tag{101}$$

and therefore

$$E(n) = \rho + \frac{\rho^2 + \lambda^2 Var(t_s)}{2(1-\rho)} \tag{102}$$

The Erlang k service-time distribution has a density function

$$s(t) = \frac{(k\mu)^k t^{k-1}}{(k-1)!} e^{-k\mu t} \tag{103}$$

The expected value and variance are

$$E(t) = \frac{1}{\mu} \tag{104}$$

$$\text{Var}(t) = 1/\mu^2 k \tag{105}$$

For the Hyper-Exponential distribution

$$s(t) = 2\sigma^2\mu e^{-2\sigma\mu t} + 2(1-\sigma)^2\mu e^{-2(1-\sigma)\mu t} \tag{106}$$

$$E(t) = \frac{1}{\mu} \tag{107}$$

$$\text{Var}(t) = \frac{1}{\mu^2}\left[1 + \frac{(1-2\sigma)^2}{2\sigma(1-\sigma)}\right] \tag{108}$$

where:

$$0 < \sigma \leqslant 1/2$$

The variance of the Erlang distribution is less than that of the exponential by the factor k. The variance of the Hyper-Exponential is greater than that of the exponential. Table XI presents values of the ratio of the variance of the Hyper-Exponential distribution to that of the exponential for several values of σ.

TABLE XI

RATIO OF HYPER-EXPONENTIAL VARIANCE AND EXPONENTIAL VARIANCE

σ	Ratio
0.4	1.084
0.2	2.124
0.1	4.560
0.01	49.50
0.001	500.0

Table XII presents values of the expected number in the system for several values of ρ with exponential arrivals and the different service-time distributions. It is clear that the Erlang and Hyper-Exponential distributions can be used to obtain a variance of almost any magnitude relative to that of the exponential distribution. [Note in particular that the Erlang k = 1 distribution is the exponential, while $k = \infty$ gives a constant service time as $\text{Var}(t_s) = 0$.] Table XII indicates the general trend in the improve-

TABLE XII

EXPECTED NUMBER IN THE SINGLE-CHANNEL SYSTEM WITH EXPONENTIAL ARRIVAL TIME

(a) Exponential service time

ρ	E(n)
0.1	0.1111
0.25	0.3333
0.50	1.0000
0.90	9.0000

(b) Erlang service time

ρ	k = 2		k = 10	
	E(n)	Ratio to Exponential	E(n)	Ratio to Exponential
0.1	0.1084	0.975	0.1061	0.955
0.25	0.3125	0.940	0.2959	0.888
0.5	0.8750	0.875	0.7750	0.775
0.9	6.9700	0.775	5.3500	0.595

(c) Hyper-Exponential service time

ρ	σ = 0.1		σ = 0.01	
	E(n)	Ratio to Exponential	E(n)	Ratio to Exponential
0.1	0.1309	1.175	0.3805	3.420
0.25	0.4820	1.450	2.3550	7.080
0.5	1.8900	1.890	13.1250	13.125
0.9	23.4500	2.605	205.4000	22.80

ment of degradation of the mean number in the system for various service-time distributions. Also, it is clear from Eqs. (102) and (108) that the worst possible distribution is the Hyper-Exponential with $\sigma \to 0$, having infinite variance. It seems quite unlikely that such a distribution would ever be encountered in practice. Indeed, a variance of only several times the square of the mean seems to be the greatest that would be encountered. Since the variance of the exponential distribution is equal to the square of its mean value, this distribution could be considered as a moderately pessimistic

case. Considering also its mathematical convenience, this justifies the emphasis on the exponential model.

4.7.2. TRANSIENT SOLUTION FOR THE STATE PROBABILITIES

The effect of the service-time distribution on the mean number in the system is indicative of its effect on other measures of steady-state performance, average number in the queue, mean throughput time, etc. In this section the transient solutions are considered for a single-channel system with exponential arrivals and exponential, Erlang k = 2, and Hyper-Exponential service-time distributions.

The set of differential equations has already been discussed in Section 4.2 for the exponential arrival-time distribution with the exponential, Erlang Type Two and Hyper-Exponential service-time distributions. Solutions for these equations have been obtained for values of $\rho = \lambda/\mu$ of 0.25 and 0.5. The maximum allowed state for the system was chosen sufficiently high to give a good approximation to the infinite-queue case. The steady-state values for the state probabilities for the Erlang and Hyper-Exponential distributions have not been calculated. However, P_0 is known since P_0 is equal to $1-\rho$ regardless of the service-time distribution for the single-channel system. $P_0(t)$ is therefore used here as the measure of how close the system is to steady-state behavior. The only initial condition used was zero requests in the system.

Figures 37 and 38 compare $P_0(t)$ for the exponential, Erlang Type Two, and Hyper-Exponential distribution with $\sigma = 0.1$. It is clear that the system with Erlang service-time distribution approaches steady-state behavior faster than the exponential system. On the other hand, the Hyper-Exponential system takes longer to reach the steady state. The conclusion can therefore be drawn that systems having smaller variance in the service time than that of the exponential distribution will reach steady-state conditions faster, while those with larger variance will approach the steady state more slowly.

4.7.3. SUMMARY

It has been shown that the expected number of units waiting or in service in the single-channel system, with first come, first served queue discipline, is a function of the variance of the service-time and arrival-time distributions. The expected number can be reduced by reducing either or both of these variances.

Consideration of the transient solutions for the state probabilities for exponentially distributed arrival intervals and exponential, Erlang, and Hyper-Exponential service-time distributions has shown that the time required

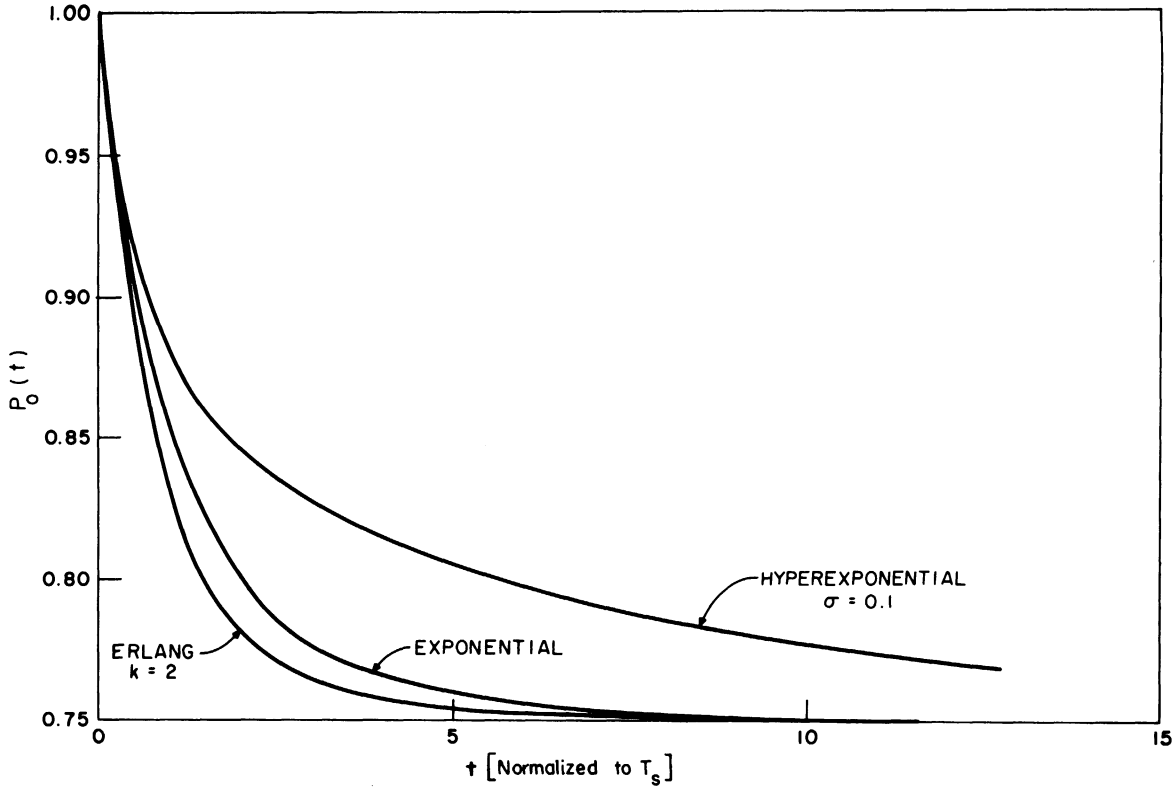

Figure 37. $P_0(t)$ for the single-channel system, $\rho = 0.25$.

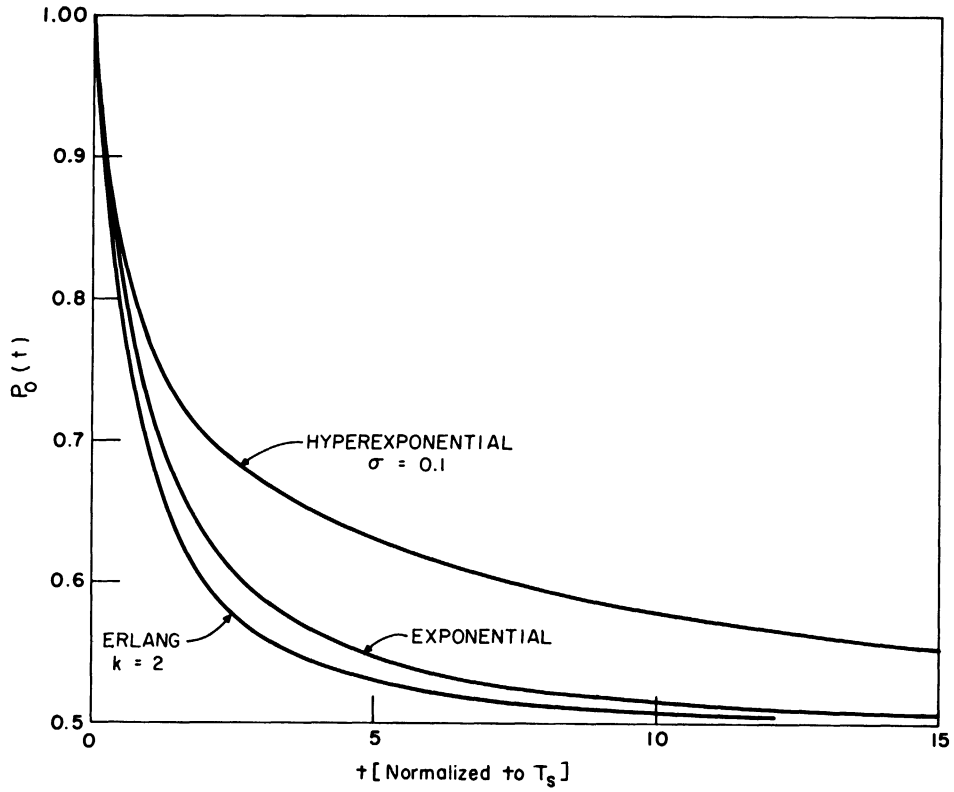

Figure 38. $P_0(t)$ for the single-channel system, $\rho = 0.5$.

to reach steady-state behavior from a zero initial condition can also be reduced by reducing the variance of the service-time distribution.

It seems quite likely that the optimum distribution for arrival and service times in regard to all aspects of performance of the single-channel system is to have them as constants, although this investigation has not been so extensive as to warrant this as a rigorous conclusion. On the other hand, the exponential distribution, because its variance is near the limit of what one might expect to encounter in a realistic case, and because of the comparison of the performance of exponential systems with those having other service-time distributions, would seem to provide a model with intermediate performance. Coupled with the mathematical convenience of using the exponential distribution, this suggests the exponential arrival- and service-time model as a good choice for exploratory studies.

In this section, queueing theory has been applied to a number of problems which arise in a polymorphic computer. These problems and the resulting conclusions are summarized in Section 6.3. The reader who is particularly interested in the application of this service facility model to the AN/FSQ-27 should bypass Section 5, which deals with the statistical accuracy of the data obtained from the simulation.

5. STATISTICAL ACCURACY OF MEASUREMENTS FROM SIMULATION DATA

5.1. Introduction

Simulation is in essence an experimental method of obtaining results, except that experiments are not conducted on the actual system, but on a model representative of the system. Hence one has the problem of determining the accuracy of the measurements obtained, not withstanding the accuracy of the representation of the system.

The simulation which has been developed for the AN/FSQ-27 computer system is in many respects a statistical mechanism. The behavior of the system, as simulated, is therefore best described by the respective distributions and distribution moments for the random variables of interest, as such descriptions constitute the measures of system performance. The measurements to be made are consequently measurements of moments and distributions, and must be based upon a large number of observed values of the variable whose distribution is sought. To determine how accurately a measurement represents the true value of the performance measure requires some knowledge of the true distribution of the observed values, and the number of sample values used.

It is obvious that little is known about the true distributions of variables in the complex system representation realized in the AN/FSQ-27 simulation. However, the gross behavior of the system is that of a multiple-channel service facility. The true values of the several performance measures for this analytical model can of course be calculated as indicated in the preceding section. The accuracy of measurements in the AN/FSQ-27 simulation (Section 3), can therefore be investigated on an approximate basis through a study of the statistical accuracy of measurements which one would obtain from a simulation based upon the multiple-channel model for the AN/FSQ-27.

This section describes methods for determining statistical errors as a function of the sample size. Specific calculations are made of the required sample sizes for a given accuracy when the measured quantities are:

(1) mean arrival or service time;

(2) mean throughput time (mean total delay from input to output);

(3) mean number of requests in the system;

(4) state probabilities; and

(5) mean time to the first passage to a given number in the system.

The calculations are based upon exponential arrival- and service-time distributions in the multiple-channel model.

Finally, the calculated sample sizes are related to length of simulation runs using two observed values for the speed of the AN/FSQ-27 simulation program.

5.2. General Discussion of Statistical Errors

The measurements which are to be considered here fall into two categories: measurement of a mean value, and measurement of a probability.

A measurement of a mean value can be obtained by averaging a large number of independent observations of the random variable (e.g., arrival interval or throughput time) whose mean value is desired. Because of the random fluctuations of the individual observations, the measured mean value has a random deviation about the true mean. However, as a result of the central limit theorem, the distribution of the measured mean is Gaussian regardless of the distribution of the observed values, provided only that the number used (the sample size) is large enough. The expected value of the measurement is the true mean value, and the variance[7] is given by:

$$\text{variance of sample mean} = \frac{\text{population variance}}{\text{sample size}} \qquad (109)$$

The situation in regard to determining the statistical error of a measured mean from the true mean is, then, as shown in Figure 39. Given a maximum allowed error (ϵ), the true mean (m), the population variance, the sample size (N), and the standard deviation (σ) of the measurement, one can compute from a table of the normal probability density function, knowing sample variance from Eq. (109) the probability that the measured value will fall within the error interval. This is the cross-hatched area in Figure 39. Conversely, given the maximum error and the probability that the measurement falls within it, one can compute the required sample variance and consequently the required sample size.

A measurement of the probability of an event can be obtained by counting the number of observations on which the event occurred, and dividing by the

[7] Cf. Goode and Machol (Ref. 6), p. 101.

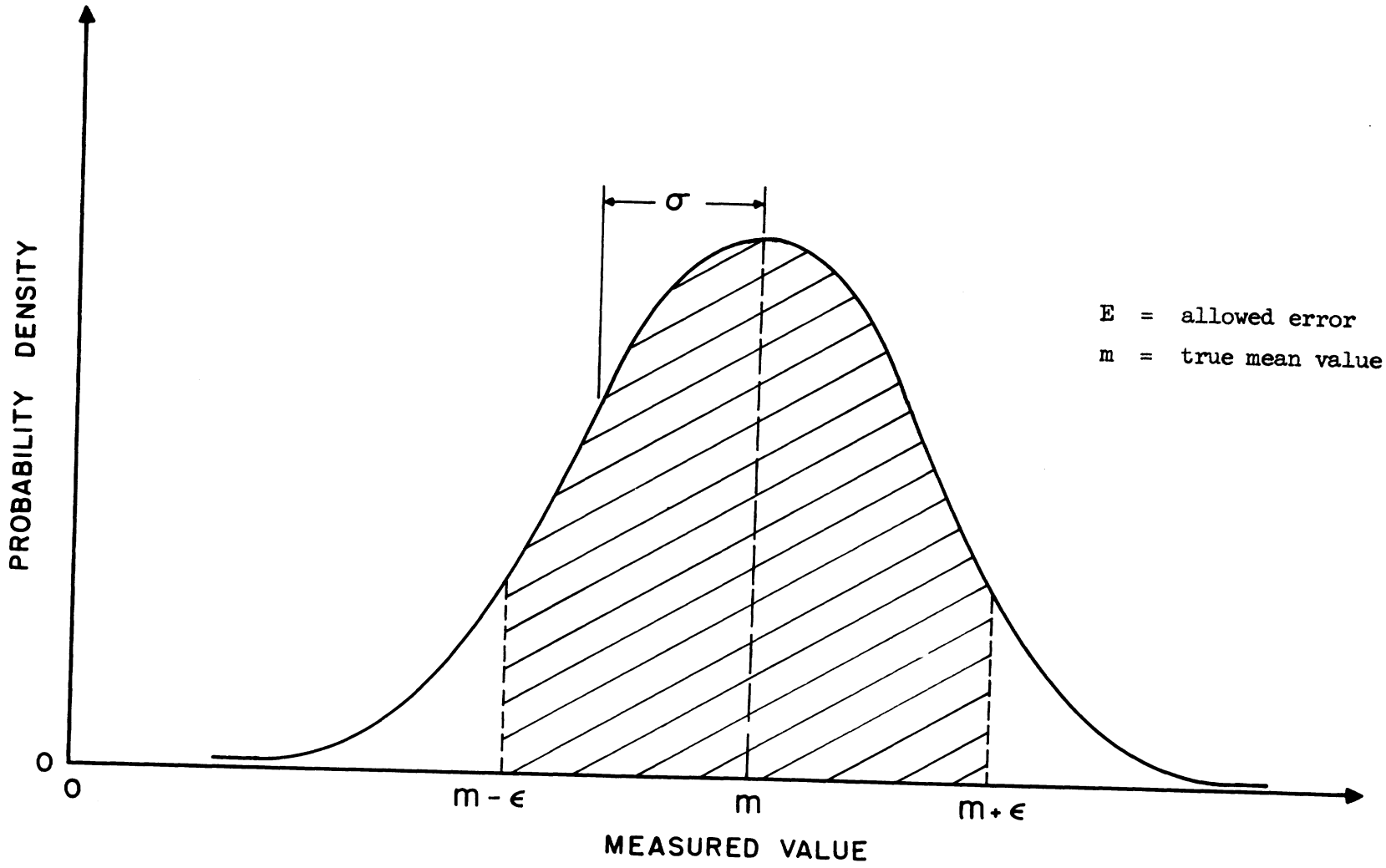

Figure 39. Distribution of a measured mean value.

total number of observations. The expected value of this estimate is equal to the true probability. The variance of the number of occurrences of the event is determined from the binomial distribution:[8]

$$\text{variance of number of occurrences} = Np(1-p) \tag{110}$$

where p is the true probability of the event, and N is the total number of independent observations or trials. Since the probability estimate is obtained by dividing the number of occurrences by N, it is clear that the variance of the probability estimate is found by dividing Eq. (110) by N^2.

Hence,

$$\text{variance of probability estimate} = \frac{p(1-p)}{N} \tag{111}$$

The probability estimate is again normally distributed, and hence one can calculate the probability that the estimate falls within a certain percentage of the true value just as in the case of a mean value.

Formulas (109) and (111) are used in the following sections to determine sample sizes (N) needed to obtain estimates falling within a given error percentage of the true values.

5.3. Calculation of Sample Sizes

5.3.1. MEAN ARRIVAL AND SERVICE TIME

For the exponential distribution on time interval between arrivals or time duration for service,

$$f(t)dt = \lambda\epsilon^{-\lambda t}\, dt, \tag{112}$$

the population variance is $1/\lambda^2$, and the mean value is $1/\lambda$. Therefore, the variance of the measured mean time interval is

[8] Feller (Ref. 7), p. 214.

$$\sigma^2 = \frac{1}{\lambda^2 N} \tag{113}$$

and since the expected value of the measured mean is equal to $1/\lambda$,

$$\frac{\sigma}{m} = (N)^{-\frac{1}{2}} \tag{114}$$

In this case the sample size N is the number of arrivals or service completions observed. Values of N required to make the maximum error in the measured mean ±10, ±20, and ±40% with 0.95 probability have been calculated. The results are presented in Table XIII.

TABLE XIII

SAMPLE SIZES FOR MEASUREMENTS OF EXPONENTIALLY DISTRIBUTED ARRIVAL OR SERVICE TIME

95% Probable Maximum Error	Expected Value of Error	σ/m Ratio of Measurement	Required N
± 10%	± 4.85%	.061	270
± 20%	± 9.7%	.122	68
± 40%	±19.4%	.244	17

Also indicated in Table XIII are the expected values of the error for a given sample size N.

5.3.2. MEAN THROUGHPUT TIME

Any observed value of arrival or service time is independent of the state of the system. Such is not the case for the remaining measurements to be discussed here. For example, throughput time is by definition the total time taken for a request to go through the system, from the moment of its arrival to the moment of its departure. It thus includes service time and waiting time, with the latter dependent upon the state of the system at the time of arrival. The state of the system at any instant is in turn dependent to some extent upon its state at any previous instant (the "transient" effect discussed in Section 4.3). Hence, in contrast to observations of service time, consecutive observations of throughput time of completed jobs are not inde-

pendent. For independent observations to be obtained, an amount of time must elapse between observations so that the degree to which the system state at one observation point determines the system state at the next point has become negligible. This interval, which can be determined from the transient solutions for the state probabilities, will be called the "sampling interval."

The discussion of transient solutions has pointed out that the sampling interval is dependent upon the system state (initial condition) at the preceding observation point. It has been found that for values of $\rho \leqslant 0.8$, an initial state of 20 requests in the system results in the longest sampling interval (for a system limited to N = 20). Rather than determine a sampling interval for each of the possible initial states, the sampling interval for an initial state of 20 will be used. This will yield somewhat pessimistic results for the total time duration of a simulation run.

The sampling interval also depends upon the measurement being made. For example, the distribution of throughput time,[9] t_p, given by

$$P(t_p \geqslant t) = \frac{(M-1-\rho M+Q_M)e^{-\mu t} - Q_M}{M - 1 - \rho M} e^{-\mu M(1-\rho)t} \qquad (115)$$

depends upon the transient phenomena only to the extent that

$$Q_M = \sum_{n=M}^{N} P_n(t)$$

does. The time interval required for Q_M to reach the steady-state value (within a given small error) is clearly smaller than the time for $P_{20}(t)$ to reach steady-state value. Hence, measurements of mean throughput time can be made with a smaller sampling interval than measurements of P_{20}. The criterion for determining the sampling interval is, therefore, that the error due to the transient (nonindependent) phenomenon, in conjunction with the statistical error due to a finite number of sample values, shall be less than the desired error in the measurements of steady-state behavior.

From Eq. (115) the expected value and variance of throughput time can be obtained as follows:

$$E(t_p) = \frac{1}{\mu} + \frac{Q_M}{\mu M(1-\rho)} \qquad (116)$$

---

[9]See Morse (Ref. 1), p. 107.

$$\mathrm{Var}(t_p) = 1 + \frac{Q_M(2-Q_M)}{M^2(1-\rho)^2} \qquad (117)$$

The transient solutions for the state probabilities yield transient values of $E(t_p)$ and $\mathrm{Var}(t_p)$, as shown in Table XIV. (As previously indicated, the initial state is 20.)

TABLE XIV

TRANSIENT VALUES FOR Q_M, $E(t_p)$, AND $\mathrm{Var}(t_p)$

ρ	T (Normalized to T_s)	$Q_M = \sum_{M}^{N} P_n(t)$	$\mu E(t_p)$	$\mu^2 \mathrm{Var}(t_p)$
0.5	10.5	.08131	1.02323	1.01273
	11.5	.07873	1.02249	1.01235
	12.5	.07744	1.02212	1.01215
	∞	.07619	1.02177	1.01196
0.8	10.5	.53993	1.38567	1.40221
	12.5	.51200	1.36571	1.38870
	15.0	.49321	1.35229	1.37916
	∞	.47472	1.33908	1.36943

The value T is the sampling interval normalized to the mean service time, $T_s = 1/\mu$. Steady-state values are given as the entries $T \to \infty$.

Table XV presents the sample sizes calculated at the given sampling intervals to achieve 10% maximum error from the steady-state mean throughput time. It is clear that, as T increases, the error due to the transient solution decreases, and the total error is principally composed of the statistical error due to the finite sample. Hence, the number of sample values required decreases, but as a result of the increasing sampling interval, the total simulated time required (NT) increases. The desired sampling interval is the one which results in minimum NT. For $\rho = 0.5$. T is therefore 7 mean service times.

5.3.3. MEASUREMENT OF MEAN NUMBER IN THE SYSTEM

The mean number of job requests in the system, either in service or awaiting service, is

$$E(n) = L = \sum_{n=0}^{N} nP_n \qquad (118)$$

TABLE XV

CALCULATED VALUES OF SAMPLE SIZE FOR DIFFERENT SAMPLING INTERVALS IN MEASUREMENT OF MEAN THROUGHPUT TIME

ρ	T (Normalized $t_o T_s$)	95% Probable Maximum Error	σ/m Ratio of Measurement	Sample Size N	NT
0.5	5.0	± 10%	.0253	1625	8125
	7.0	± 10%	.0495	416	2912
	10.5	± 10%	.0521	375	3938
0.8	10.5	± 10%	.0529	500	5250
	12.5	± 10%	.0627	356	4450
	15.0	± 10%	.0670	306	4590

where P_n are the state probabilities. The steady-state value of L is obtained using the steady-state values for the probabilities

$$P_n = \rho^n \frac{M^n}{n!} P_o \qquad 0 \leqslant n \leqslant M$$

$$P_n = \rho^n \frac{M^M}{M!} P_o \qquad M \leqslant n \leqslant N$$

where P_o is determined by

$$\sum_{n=0}^{N} P_n = 1.0$$

Use of the transient values of the state probabilities gives a transient value, L(t), just as in the case of mean throughput time discussed in the preceding paragraphs.

The calculation of the sampling interval and sample sizes is therefore quite similar to the discussion in Section 5.3.2. Table XVI presents the values of N and T which give the desired maximum error with minimum length of run, NT.

5.3.4. MEASUREMENT OF STATE PROBABILITIES

The number of samples and sampling intervals have been determined for measurements of state probabilities. The determination involves use of Eq.

TABLE XVI

SAMPLE SIZES AND SAMPLING INTERVALS FOR MEASUREMENTS OF MEAN NUMBER IN THE SYSTEM

ρ	95% Probable Maximum Error	Steady-State L	Transient L	T	N
0.5	± 10%	3.576	3.593	12.0	124
	± 20%	3.576	3.593	12.0	31
0.8	± 10%	7.163	7.217	20.0	119
	± 20%	7.163	7.217	20.0	30

(110), using for p the calculated transient state probability for an initial state of 20. Values of N and T are then found so that the deviation of the transient probability from the steady-state value, together with the statistical error, is less than the desired maximum error in measurement of the steady-state value (with 95% probability). The values of N and T also give minimum NT.

Results are presented in Table XVII for several probabilities. The results indicate that a large number of samples is required, primarily because the state probabilities are rather low. Sample sizes could be reduced considerably if, instead of measuring probabilities of individual states, one were content to measure, for example, the probability of all channels being busy.

TABLE XVII

SAMPLE SIZES FOR MEASUREMENT OF STATE PROBABILITIES

Steady-State Probability	ρ	T	Transient Probability	95% Probable Maximum Error	N
P_7 = .038099	0.5	12.0	.038486	± 10%	10,230
				± 20%	2,465
P_7 = .090407	0.7	12.1	.09006	± 10%	3,870
				± 20%	955
P_{12} = .001191	0.5	14.3	.001214	± 10%	390,000
				± 20%	84,600

5.3.5. MEASUREMENT OF THE MEAN FIRST-PASSAGE TIME TO THE n^{th} STATE

The mean first-passage time ($t_{o,n}$) to the n^{th} state, assuming that the system is initially in state zero at time t = 0, is given by

$$E(t_{o,n}) = \int_o^\infty tp(t_{o,n})dt \qquad (119)$$

where:

$p(t_{o,n})$, the probability density of the first passage to the n^{th} state.

If, however, $E(t_{o,n})$ is obtained from reduction of simulation data, the parameter of interest is the number of observations required to reduce the variance of the sample mean to the desired level [Eq. (109)]. The number of samples required to obtain maximum error of ±10% or ±20% (with 95% probability of the measurement falling in the specified error interval about the true mean) for some representative measurements is shown in Table XVIII. The number of observations is greatest for small ρ and/or large N.

Each observation corresponds to one simulation run. The system initially is in state zero, and the observation is terminated when the n^{th} state is reached. When the distributions $P(t_{o,n})$ are known, everything about the length of the simulation run is known. The normalized sampling interval T is therefore the expected value $E(t_{o,n}) = T$, indicated in Table XVIII.

The total amount of simulated time (in terms of $T_S = 1/\mu$) required is equal to the TN product. For example, this computation for $P(t_{o,10})$ at $\rho = .5$ is given by

$$TN = 15{,}450 \text{ units of } T_S$$

5.4. Length of Simulation Run

The results for sample sizes presented in the previous sections can be expressed in terms of average IBM-704 time required to achieve that number of samples in the AN/FSQ-27 simulation program. The two observed speeds of this program, normalized to the arrival rate, are 0.86 λ and 0.044 λ. (The units of these numbers are minutes of computer time per unit of simulated time.) To convert data on sample sizes to minutes of simulation run, then, requires an expression for the mean interval between samples in terms of the average arrival interval, $T_a = 1/\lambda$. This interval is tabulated in Table XIX, column two, for the various measurements to be made. The expression for the

interval in terms of λ makes use of the relation $\rho = \lambda/7\mu$.

TABLE XVIII

SAMPLE SIZES FOR THE MEASUREMENT OF THE MEAN FIRST-PASSAGE TO THE m^{th} STATE

$P(t_{o,n})$	ρ	95% Probable Maximum Error	T	$Var(t_{o,n})$	N
$P(t_{o,7})$	.5	± 10%	8.218	48.579/μ	194
		± 20%			69
	.7	± 10%	3.510	6.325/μ	111
		± 20%			49
$P(t_{o,10})$	.5	± 10%	60.056	3,462.888/μ	258
		± 20%			92
	.7	± 10%	12.134	109.430/μ	200
		± 20%			71
	.9	± 10%	5.157	15.113/μ	153
		± 20%			54
$P(t_{o,20})$	.9	± 10%	41.732	1,343.484/μ	208
		± 20%			74

TABLE XIX

AVERAGE INTERVAL BETWEEN SAMPLES

Measurement	Mean Interval in Units of Simulated Time	Interval in Terms of λ
Mean arrival time	$1/\lambda$	$1/\lambda$
Mean service time	$1/\lambda$	$1/\lambda$
Mean throughput time	T/μ	$7\rho T/\lambda$
Mean number in system	T/μ	$7\rho T/\lambda$
State probability	T/μ	$7\rho T/\lambda$
Mean first-passage time	T/μ	$7\rho T/\lambda$

To convert the data on sample sizes to simulation run length, the number of samples N is multiplied by the appropriate average interval between samples given in Table XIX and the speed of the program (.86 λ or .044 λ). Table XX presents the values of 704 time obtained in this manner for various measurements. The steady-state value of the quantity to be measured, as computed from the assumed analytical model, has been included.

The principal factor determining the speed of the simulation program is the number of time log entries per unit of simulated time. The number of entries can vary over a wide range, depending upon the complexity of the job program representation (i.e., the number of major joints required per job program). The figures in the last column of Table XX indicate the 704 time required when the job program is represented by only one major joint. As a result these figures are representative of the maximum performance for the present stimulation. On the other hand, the next to last column indicates the time requirements when the job program representation contains a large number ($\approx$ 1000) major joints per job. In the first case, we have essentially a simple queue simulator, while in the second, queue interaction occurs because there is competition for the shared modules.

5.5. Summary

It is apparent from a consideration of Table XX that even with an efficient simulation (i.e., high speed), experimentally determined state probabilities and mean first-passage times for small ρ and large n are beyond consideration at a charge rate of $5.00 per minute (the cost of 704 computer time). On the other hand, mean arrival, service, and throughput time, as well as measurements on the mean number of jobs in the system, are certainly within acceptable limits. The principal advantage of a Monte Carlo technique is that the system can be modeled to a detail which cannot be obtained by analytical methods. The price to be paid for this desirable feature is, in general, expensive computer runs. It is clear, then, that the advantage of the simulation approach can be lost if the cost of simulation runs approaches the cost of direct experimentation with an actual system.

Numerical solutions for an analytical model, as in the case of transient solutions for the state probabilities, represent a second approach to the problem of system evaluation. Once the equations of detailed balance are obtained and programmed on a computer, the computer costs are considerably less than for a Monte Carlo simulation. For example, numerical calculation of the first-passage time distribution to the 10^{th} state for $\rho = .5$ was obtained in this study for $120.00; the same result (see Table XIX) would have cost $11,900 using the simulation. Therefore it is desirable to program the solutions to an analytical model if the model studied is satisfactory for the application.

TABLE XX

AVERAGE DURATION OF SIMULATION RUN FOR MEASUREMENTS

Measurement	ρ	Steady-State Value to be Measured	95% Probable Maximum Error	Number of Samples Required	Average 704 Time Required at Speed of 0.86 λ	Average 704 Time Required at Speed of 0.044 λ
Mean arrival or service time	-	$1/\lambda$ or $1/\mu$	± 10%	270	232 min	12 min
			± 20%	68	58	3
Mean throughput time	0.5	$1.022/\mu$	± 10%	416	8,750	448
			± 20%	97	2,040	105
	0.8	$1.339/\mu$	± 10%	356	21,400	1,100
			± 20%	78	4,700	240
Mean number in the system	0.5	3.576	± 10%	124	4,480	230
			± 20%	31	1,120	58
	0.8	7.163	± 10%	119	11,500	586
			± 20%	30	2,900	148
State probability P_7	0.5	0.038099	± 10%	10,230	370,000	18,900
			± 20%	2,465	89,000	4,550
	0.7	0.090407	± 10%	3,870	197,500	10,100
			± 20%	955	48,500	2,490
P_{12}	0.5	0.001191	± 10%	390,000	16,800,000	860,000
			± 20%	84,600	3,640,000	186,000
Mean first-passage time to state n, n = 7	0.5	$8.218/\mu$	± 10%	194	4,470	245
			± 20%	69	1,695	87
	0.7	$3.510/\mu$	± 10%	111	1,640	838
			± 20%	49	725	37
n = 10	0.5	$60.1/\mu$	± 10%	258	46,500	2,380
			± 20%	92	16,550	850
	0.7	$12.1/\mu$	± 10%	200	10,200	518
			± 20%	71	3,530	184
n = 20	0.9	$41.7/\mu$	± 10%	208	47,000	240
			± 20%	74	16,700	85

6. SUMMARY AND CONCLUSIONS

6.1. Capability of the Simulation

The computer program described in Section 3 incorporates a considerable degree of flexibility. The result of this flexibility is a rather broad capability for study of different aspects of the system design.

The input generating mechanism of the simulation program makes it possible to consider a variety of problem mixes. It provides for arbitrary specification of the distribution of arrival intervals for each problem class, and also for the simulation of interdependence between arrivals. The latter feature is especially useful in studies of job-program segmentation, where arrivals would be interpreted as requests for execution of only a segment of an individual program, and would therefore be dependent upon the completion of preceding segments. The ability to assign priorities to entire programs or to individual segments is an important feature of the simulation program.

The statistical method of generating the characteristics of job programs or program segments gives generality in studying the performance of the system in processing different classes of problems. The equipment requirements, time durations, and interruptibility characteristics of each class of problems can be varied. Representation of the manner in which programs use individual modules (in terms of major and minor joint durations) makes possible the study of subordinate module sharing.

In addition to the number and variety of subordinate modules available in the system, the simulation program provides for arbitrary specification of the access time distribution for each module type, as well as the switching time required to establish connections between modules.

It is clear that flexibility in the study of master program functions cannot easily be provided through variable parameters alone. Since functions analogous to those of the master program in a real system must also be performed in the simulation, modifications of master program functions will involve reprogramming parts of the simulation program. However, the simulation has been designed to provide isolation between sections of the master program, so many modifications can be accommodated without undue difficulty. Important among these is the method used by the master program to schedule job requests for service, i.e., the queue discipline.

6.2. Limitations of the Simulation

The flexibility and detail of system modeling which has been introduced in this simulation has of course resulted in a reduction of simulation speed from what might be accomplished with a cruder model and a less flexible program. Section 5 has indicated that obtaining measurements from the simulation requires long and rather expensive computer runs in general, and in particular that measurements of state probabilities cannot practicably be obtained. The following table reproduces certain entries from Table XX in Section 5.4, and indicates the lengths of run required for those measurements that are at all feasible. These figures are based on the slowest simulation speed, since this case is representative of the detail of modeling for which the simulation would be expected to show its greatest utility.

Measurement	System Utilization Factor ($\simeq\rho$)	704 Computer Time
Mean throughput time	0.5	8,750 min
Mean number in the system	0.5	4,480
Mean first-passage time to state 7	0.5	4,470

The speed of the simulation could of course be considerably increased by a simplification of the model. But the complexity of the system being modeled here demands a rather complex simulation if one is to answer the questions that originally resulted in the choice of simulation as a means of investigation. The Monte Carlo allows the system to be modeled to a detail which cannot easily be obtained by analytical methods. An attempt to improve simulation speed by elimination of this detail would therefore reduce the value of the simulation approach.

Other possibilities for improvement of speed are coding of the simulation program in an assembly language rather than a compiler, refinements in programming technique, and use of a higher-speed computer. It is difficult to assess the degree of improvement that could be obtained in such ways, but it is unreasonable to anticipate much more than an order of magnitude improvement. This would be insufficient to bring state probability measurements within the realm of practicality, and would not appreciably alter the conclusion that simulation runs are expensive.

There is another alternative, given only limited consideration, which can also lead to an improvement in simulation speed. This is to incorporate results of analytical studies of parts of the system in the Monte Carlo simulation of the complete system. The analytical results to be incorporated would of course be distributions on time intervals, or the probabilities of events. The simulation would thereby be simplified. This procedure has been used to some extent in the AN/FSQ-27 simulation, as can be seen from the following example.

Consider the method used to generate exponentially distributed intervals for job request arrivals. The desired interval is obtained using the functional relationship between a random variable uniformly distributed over the unit interval and an exponentially distributed random variable (see Section A.2.6). The same result could have been obtained by specifying a constant probability of a job-request arrival in an incremental element of time dt, independent from interval to interval, and thereby generating the occurrence of a job request by a Monte Carlo modeling of the shot noise mechanism. The latter is obviously inefficient compared to the former. The point is simply that analytical models (i.e., the distribution functions obtained from their solutions) should be incorporated whenever possible. If large portions of the system can be handled in this manner, the utility of a simulation is enhanced.

Specifically, one might include the throughput time distributions obtained from the solutions of some simple queueing problems in a second-generation simulation program.

Although judgment of the utility of simulation is in the last analysis subjective, the above discussion supports the conclusion that a simulation, as described in Section 3, is an expensive research tool for a complex system like the AN/FSQ-27. This conclusion holds particularly when the purpose of a study is to derive general principles regarding system behavior. Such a study, of course, involves varying many different parameters over broad limits; in other words, a large number of runs. It is therefore desirable to use simplified analytical models, wherever possible, to establish broad principles of behavior, to guide the investigator to those aspects of system detail which are critical in performance, and to achieve simplification of simulation when this is possible. Further attention should be given to the problem of improving simulation techniques in the manner suggested in this section.

6.3. The Queueing Theory Model

The mathematical theory of queues has proven to be an effective approach to an analytical model for the AN/FSQ-27 system.

As applied to a polymorphic computer, queueing theory provides measures of system performance in two areas: the number of requests in the system, and the waiting and throughput time per job request.

It has been shown that an attempt to obtain high utilization for the job computers increases the mean queue length. This implies, of course, that the delay per request is also increased. The trade-off between efficient utilization of equipment and waiting time is therefore a basic problem in a polymorphic computer. Regardless of the measure employed (mean waiting time as in Section 4.4 or nonsaturation of storage as in Section 4.5), the penalty for high utilization of the job computers is increased delay. A graphic illustration of this point is the inefficient use of the job computers, i.e., increased service time per channel, when there is competition for a shared module. To the extent that the exponential service facilities described in Sections 4.4, 4.5, and 4.6 model the AN/FSQ-27 polymorphic computer, the conclusion can be drawn that the shared module utilization should be less than .75 to avoid queueing for a shared module.[10] Similarly, the system utilization factor ($\simeq\rho$) should not exceed .5 if the mean waiting time is to be small ($T_W < .1\ T_S$) compared to the mean computation time per job request.[11] Although a relative weighting between delay and equipment utilization is implicit in the above summarizing statement, the general principle still remains. The price for high equipment utilization is a degrading of the "time" performance of the system and, in general, this degradation becomes increasingly nonlinear as the utilization factor approaches one.

The generality with which the state of a system can be defined justifies the position that a queueing theory approach is a powerful analytical tool.[12] Priorities, modified arrival- and service-time distributions, sequential queues, and queue interaction are some of the problems which can be handled by redefining the system state. These, as well as additional studies, would constitute the basis for additional research.

---

[10]Figure 4, Section 4.6.

[11]Figure 1, Section 4.4.

[12]A more detailed discussion of this point is included in Section 4.2.3.

7. REFERENCES

1. P. M. Morse, Queues, Inventories and Maintenance, Operations Research Society of America Publications in Operations Research, No. 1 (New York: John Wiley and Sons, Inc., 1958).

2. R. A. Frazer, W. J. Duncan, and A. R. Collar, Elementary Matrices (Cambridge: University Press, 1955).

3. L. Saaty, "Time-Dependent Solutions of the Many-Server Poisson Queue," Journal of Operations Research (December, 1960), 755-772.

4. A. Erdelyi, W. Magnus, F. Oberhettinger, and F. G. Tricomi, eds., Tables of Integral Transforms (New York: McGraw-Hill Book Co., Inc., 1954), I, No. 4.

5. L. G. Peck and R. N. Hazelwood, Finite Queueing Tables (New York: John Wiley and Sons, Inc., 1958).

6. H. H. Goode and R. E. Machol, System Engineering (New York: McGraw-Hill Book Co., Inc., 1957).

7. W. Feller, An Introduction to Probability Theory and Its Applications, 2nd Ed. (New York: John Wiley and Sons, Inc., 1959), I.

8. J. W. Butler, "Machine Sampling from Given Probability Distributions," in Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery; XXVI, Annals of the Computation Laboratory of Harvard University (1951), 249.

9. A. Rotenburg, "A New Pseudo-Random Number Generator," J. Assoc. Comp. Mach. VI, No. 1 (January, 1959), 75-77.

10. H. G. Kuehn, "A 48-Bit Pseudo-Random Number Generator," Communications of the Assoc. Comp. Mach., IV, No. 8 (August, 1961), 350-352.

APPENDIX A

DETAILS OF THE SIMULATION PROGRAM

This appendix describes the simulation program in exhaustive detail. Operating instructions, including a definitive exposition of input card format, are covered in Section A.1. Flow charts and the complete coding of the program (using MAD and SAP)[13] are available in Section A.2.

A.1. Operating Instructions

A.1.1. THE PROGRAM DECK

There are two main programs and 10 subroutines which are combined to form a two-core Ping Pong deck, with one subroutine used in both core loads. The complete operating deck contains, in this order,

(1) two ID cards

(2) the program deck (Figure 40)

(3) a specification card -- "*$ DATA"

(4) the input data cards (Section A.1.3).

Certain variations in the simulation program are made possible by the use of override cards, placed in numerical order in the program deck after the last card of the routine which is being modified. In the case of the overrides to the TIMLG subroutine, the cards must be used in both core loads. The table below describes the effect of the overrides when inserted into the program deck.

Card Identification	Function
TIMLG 098, 099	Suppresses the printing of each new time log entry.

[13] It is assumed that the reader is familiar with the MAD compiler and the Executive System as used by The University of Michigan Computing Center with the IBM-704.

MPMIN 200	Causes the master computer to loop through joints 1 and 4 when not otherwise occupied.
MPMIN 201	Eliminates consideration of switching time.[14]
MPMIN 202, 203	Eliminates consideration of job program restart penalties.[14]

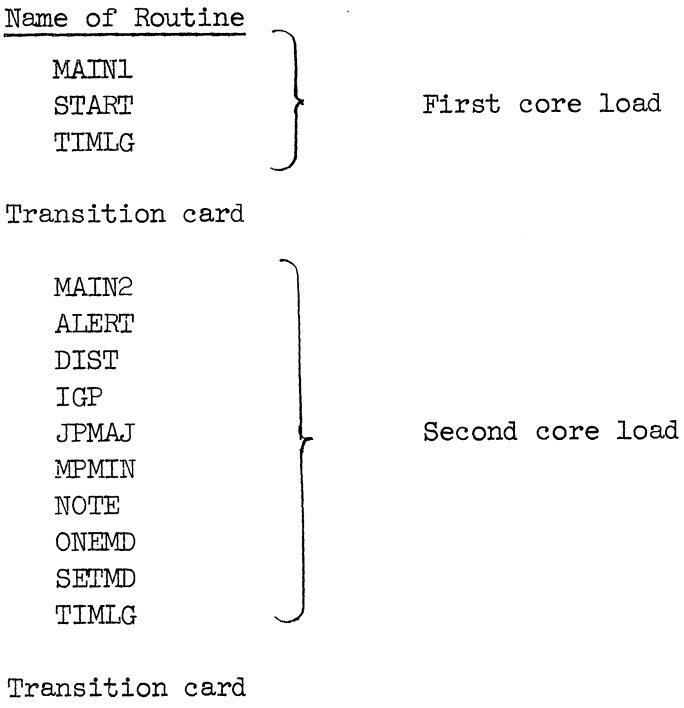

Figure 40. The program deck.

[14]It is faster to use these overrides than to make the appropriate parameters zero.

A.1.2. DISTRIBUTION PARAMETERS

All parameters P of the simulation which might be represented by a statistical distribution are carried in the program in the form of a code digit, F, and variables Pi. P is computed as a function of F and the Pi's each time that P is called for during the simulation.

F is used by the program to determine the type of function to be applied to Pi. This computation is performed in a subroutine accessible to all parts of the simulation program, so that the values of F (except for F = 1) and the functions so defined are applicable to all simulation parameters.

Table XXI describes all presently defined values of F. (r is a uniformly distributed random number, $0 \leqslant r \leqslant 1$.)

A.1.3. INPUT

All input to the simulation is in the form of one- or two-dimensional arrays of integers. These input arrays make up the initial conditions of the tables (Table I, Section 3.3.1) used by the simulation.

A.1.3.1. Card Format

A.1.3.1.1. The Run Number Card.--Each set of data to be used as input to a given simulation run must be given a run number. This is a 6-digit integer to be punched in columns 1 through 6 of every input card of a given run. A card with this information only, and the rest of the card blank, is called a run number card.

A.1.3.1.2. Data Cards.--Each array has a name of 6 or fewer alphabetic characters. The contents of each array, however, must be integers. As an example, let us assume we have a 5 x 2 array named ABCDE which contains the following information:

-1	27	666666666	0	75
0	15	-1023	35	0

The first card (Figure 41) contains the run number, the array name, a card number of 0, the number of words in each row of the array (as data word number 1), and the total number of words in the array (as data word number 2).

Succeeding cards of the array have the run number, the array name, and consecutive card numbers starting with 1. Each row of the array is punched starting at data word 1, using each data word field in succession through

TABLE XXI

THE DISTRIBUTION PARAMETERS

F	Distribution	Pi (input parameters)	P (subroutine output)
0	Constant	P1 = constant	P1
1*	Uniform over two intervals	Given the two intervals (a,b) and (c,d), with e = probability of P in (a,b): P1 = $e \cdot 10^6$ P2 = (b-a)/e P3 = a P4 = (d-c)/(1-e) P5 = d-P4	P2·r + P3 if r < e P4·r + P5 if r ⩾ e
2	Exponential	1/P2 = mean P1 = upper limit	$-P2 \cdot \log_e r$ if $-P2 \cdot \log_e r <$ lesser of (11·P2, P1) P1 otherwise
3	Normal	P2 = mean P1 = standard deviation	P will be given as positive between $\pm$ 4P1 + P2

*This is a special code to be used only for major joint lengths.

RUN NUMBER 1 6	ARRAY NAME (1) 8 13	CARD NO. (2) 14 16	DATA WORD NO. 1 (2) 18 30	DATA WORD NO. 2 (2) 32 44	DATA WORD NO. 3 (2) 46 58	DATA WORD NO. 4 (2) 60 72
112200	ABCDE	0	5	10		
112200	ABCDE	1	-1	27	666666666	0
112200	ABCDE	2	75			
112200	ABCDE	3	0	15	-1023	35
112200	ABCDE	4	0			
112200	XXX					

1. JUSTIFIED TO THE LEFT.
2. JUSTIFIED TO THE RIGHT. LEADING ZEROS NEED NOT BE PUNCHED.

Figure 41. Basic card format.

number 4, and then continuing with word 1 of the next card. A new row always starts with a new card, the card immediately following the last card of the preceding row.

Rules for keypunching these data are as follows:

(a) The array name is justified to the left.

(b) All numeric information is justified to the right.

(c) Leading zeros need not be punched. Thus, 027 may be punched as 27.

(d) A field which is entirely zero may be left blank.

(e) Plus signs need not be punched.

(f) Minus signs must be punched, but may be placed at any position to the left of the most significant digit. That is, there may be spaces between the sign (plus or minus) and the integer.

A.1.3.1.3. The Deck.—A complete set of data for a simulation run consists of one run number card followed by the data cards for the arrays. Within each array the cards must be in order by card number, but the order of arrays is immaterial.

All eleven arrays must be entered. However, if the array named XXX is not needed for a particular run, only one card need be punched (Figure 41). Data word 1, the number of words in each row of the array, may be zero in this case except for arrays QUEUE and PERBUF.

It is advisable first to keypunch the data without punching the run number, and then to make a copy of this deck. This will prove useful if the simulation is to be run, stopped, and then run again starting from the point at which the first run was stopped.

A.1.3.2. Input Arrays

There are eleven tables that must be entered at input data to the simulation. These arrays can be considered as belonging to three distinct groups. The parameter tables are the arrays containing the parameters of the hardware and the problem mix. The contents of these tables remain unchanged during the course of the simulation. The status tables show the condition of the equipment and the progress of the input problems. The contents of these tables change as the simulation proceeds. The queues are the waiting lines

that are generated. Not only the contents, but the lengths of these arrays change during the course of the program.

In the following paragraphs, the description of each array is headed by the array name, the maximum number of words for which space has been allotted in the present program, and the simulation names (if they exist) for the number of words in each row and for the number of words in the entire array. It should be noted that each row of an array must have the same number of words. The program assumes that the same units of time are given in all tables.

A.1.3.2.1. Parameter Tables.—IGPLOG, 196, DIG: This table describes related sequences of job programs to be used as input to the simulation (Section 3.3.3). If no such interdependent set of job programs is a part of the input problem mix to be simulated, the IGPLOG may be left blank.

Each sequence is given a sequence number r, which must be greater than 19. Each program to be run and each time delay between programs is considered an event of a sequence. These events are numbered from 1 through n. Each sequence is then described by n+1 rows of the IGPLOG array, with the first row used to describe the sequence in general, and one row for each event.

The first row describing each sequence has six relevant words: -1, r, F, P1, P2, and n. The distribution parameters F, P1, and P2 give the time intervals between succeeding initiations of the entire sequence.

A row describing an event which is a request for a job program has the following information: E, 0, the job program type, 0, 0, e_1, e_2,..., where the e_i's are the numbers of the events following the completion of this event, and E is the number of times that this event is listed as an e_i by other events.

A row describing a time delay between the completion of one job and the start of others has this information: E, 1, F, P1, P2, e_1, e_2,..., where E and e_i are as described above, and the distribution parameters give the length of the time delay.

All the sequences to be used as input in one run are put together into one IGPLOG array. The different sequences in one IGPLOG need not have the same number of events in each, and one sequence follows immediately after the other. Although the sequences themselves need not be in any order, the events within each sequence must be in order by event number, although this number is not keypunched on the array cards. One restriction on the form of the data is that event number 1 must be the first event to be performed. It must have an E of zero, and this must be the only event with E = 0.

Figure 42 illustrates the IGPLOG in tabular form for three sequences, the first of which is the example in Section 3.3.3. Figure 43 shows the same data in card layout form.

Event Number	ID or E	r or Event Code	Distribution Parameters or Program Number			n or e_1	e_2	e_3
	-1	28	0	20,000	0	5		
1	0	0	3			2		
2	1	1	0	1,000	0	3	5	
3	1	1	0	2,000	0	4		
4	2	0	1					
5	1	0	4			4		
	-1	26	0	500,000	0	6		
1	0	1	0	20		5	3	4
2	3	0	6			6		
3	1	0	8			2		
4	1	0	5				2	
5	1	1	0	30	0	2		
6	1	0	7					
	-1	30	2	20,000	30,000	3		
1	0	0	7				2	3
2	1	0	4					
3	1	0	6					

Figure 42. An IGPLOG table.

IMPLOG, 145, DMP: The master program may be considered as a series of connections between a computer module and a controlled module, followed by certain operations which involve the processing of job programs through the system. Figures 8 and 9 (Section 3.2.2), show these connections and the ensuing master program functions.

Each row of the IMPLOG array has data regarding one joint of the master program:

RUN NUMBER	ARRAY NAME (1)	CARD NO. (2)	DATA WORD NO. 1 (2)	DATA WORD NO. 2 (2)	DATA WORD NO. 3 (2)	DATA WORD NO. 4 (2)
1 6	8 13	14 16	18 30	32 44	46 58	60 72
112200	IGPLOG	0	8	136		
112200	IGPLOG	1	-1	28	0	20000
112200	IGPLOG	2	0	5	0	0
112200	IGPLOG	3	0	0	3	0
112200	IGPLOG	4	0	2	0	0
112200	IGPLOG	5	1	1	0	1000
112200	IGPLOG	6	0	3	5	0
112200	IGPLOG	⋮				
112200	IGPLOG	13	-1	26	0	500000
112200	IGPLOG	14	0	6	0	0
112200	IGPLOG	15	0	1	0	20
112200	IGPLOG	16	0	5	3	4
112200	IGPLOG	⋮				
112200	IGPLOG	33	1	0	6	0
112200	IGPLOG	34	0	0	0	0

1. JUSTIFIED TO THE LEFT.
2. JUSTIFIED TO THE RIGHT. LEADING ZEROS NEED NOT BE PUNCHED.

Figure 43. An IGPLOG card set.

(a) the controlled module, coded numerically (Figure 44)

(b) the interrupt criterion for that joint, where a "1" means that an alert will be accepted before the start of that joint, and a "0" means that the alert will be ignored

(c) F, P1, P2, giving the time needed to perform the function following the module connection.

Module Type	Numerical Code
BM-B	10
PB	20
CX	30
TM	40
CX memory	50
BM	60
DB	70
DM	80
PL	90
PR	100
TA	110

Figure 44. Numerical coding of module types.

There are four joints not depicted in Figures 8 and 9 (Section 3.2.2). Joints 19, 20, and 29 are needed for certain technical aspects of the simulation program. For these, the controlled module should be zero and the function times are ignored. Joint 28 is the minor joint of job programs, occurring before each major joint.

There are certain restrictions on the interrupt criteria. Joints 8, 20, 21,[15] 23, and 1 or 4 must be interruptible, while 7, 11 through 19, 22, and 24 through 28 cannot be interrupted. It should be noted that 8 and 23 are also interruptible at any point during the joint.

The controlled modules for joints 8 and 23 should be given as zero, as these joints represent a computer module not connected to any controlled module, waiting for an alert from another computer module. For this reason

[15] If a job computer receives an alert after the completion of a job program but before it can connect to the master drum to mark the job complete, the completion of the job by the job computer will be noted twice in the output, and the ASTRSK and INPROG totals will be incorrect accordingly (Figure 61).

also, the function times for these joints must be given as very large—any amount which will insure that the simulation comes to an end before these joints are over.

Joints 7, 13, 15, 22, 24, and 26, representing connections to the CX, should have a function time of zero, although this is not necessary to the simulation. The function time for joint 12 should also be zero, with the access time to the CX memory, given in the module characteristic table, specifying the time to load the memory (Section 3.3.7).

A sample IMPLOG in tabular form is given in Figure 45. Part of the IMPLOG in card layout form is shown in Figure 46. Note that it is not necessary to consider P2 if this is zero for all the joints. Although the cards must be in joint number order, the joint number does not appear on the cards.

MCT, 120, DMC: The module characteristic table contains the access time for each type of controlled module in the system which is used by the master program, i.e., listed in the IMPLOG. There is only one row for each different type of module. Each row gives the module type (coded numerically) and the access time in F, P1, P2 form. If P2 is not needed, it does not have to be included.

The central exchange memory is considered as a controlled module for this purpose, and is included in the table. Its access time represents the time needed to load the memory. The CX module need not be listed.

The rows of the array may be in any order. Figure 47 shows a sample MCT array.

PURVUE, 368, DPUR: The purview table contains information specifying the job type. It is assumed by the simulation program that the job program types are numbered consecutively starting with 1, that each row of the purview table has the data for one job type, and that the rows of the PURVUE array are in order by job type number, although these numbers are not punched.

The first word of each array is the priority number of the job. The next four sets of three words each are the distribution parameters F, P1, and P2 needed to compute the following data:

(a) the time interval after which (from the present time or from the time of the alert, whichever is later) the job will interrupt itself

(b) the time interval between successive requests for, or arrivals of, this program, a quantity used only when this job is in the input problem mix by itself and is not part of a sequence

Joint Number	Controlled Module	Interrupt Criterion	F	P1	P2
1	PB	1	0	1,000	0
2	MT	1	0	140,000	0
3	MD	1	0	3,000	
4	MD	1	0	55,000	
5	PB	1	0	1,000	
6	MD	1	0	260,000	
7	CX	0	0	0	
8	0	1	0	34,359,738,367	
9	MD	1	0	240,000	
10	MD	1	0	1,000	
11	MD	0	0	2,000	
12	CX memory	0	0	0	
13	CX	0	0	0	
14	MD	0	0	240,000	
15	CX	0	0	0	
16	PB	0	0	1,000	
17	MT	0	0	200,000	
18	MD	0	0	1,000	
19	0	0	0	0	
20	0	1	0	0	
21	MD	1	0	26,000	
22	CX	0	0	0	
23	0	1	0	34,359,738,367	
24	CX	0	0	0	
25	MD	0	0	1,000	
26	CX	0	0	0	
27	MT	0	0	40,000	
28	PB	0	0	1,000	
29	0	1	0	0	0

Figure 45. An IMPLOG table.

RUN NUMBER	ARRAY NAME (1)	CARD NO. (2)	DATA WORD NO. 1 (2)	DATA WORD NO. 2 (2)	DATA WORD NO. 3 (2)	DATA WORD NO. 4 (2)
1 6	8 13	14 16	18 30	32 44	46 58	60 72
112200	IMPLOG		4	116		
112200	IMPLOG	1	20	1		1000
112200	IMPLOG	2	40	1		140000
112200	IMPLOG	3	10	1		3000
112200	IMPLOG	4	10	1		55000
112200	IMPLOG	5	20	1		1000
112200	IMPLOG	6	10	1		260000
112200	IMPLOG	7	30			
112200	IMPLOG	8		1		34359738367
112200	IMPLOG	:				
112200	IMPLOG	29				

1. JUSTIFIED TO THE LEFT.
2. JUSTIFIED TO THE RIGHT. LEADING ZEROS NEED NOT BE PUNCHED.

Figure 46. An IMPLOG card set.

Module Number	F	P1
10	0	8,500
20	0	8,500
40	0	2,800,000
50	0	2,000

Figure 47. An MCT table.

(c) the length of a major joint

(d) the number of major joints.

All the modules required to run the job program are listed, in numerical order, starting with the 14^{th} word of the array. The computer module and shared modules must not be listed. The word immediately after the last module must be 999.

If the distribution code of F = 1 is used (for major joint lengths only), parameters P3, P4, and P5 are placed in words 23, 24, and 25 of the PURVUE array. These need not be included on the input cards if not used. This means that, in this situation no job can have more than 8 exclusive modules, and only 14 different problem types can be entered as input data. Words 26, 27, and 28 contain the distribution parameters for the penalty applied when a interrupted job is restarted. When job restart parameters are used, words 23, 24, and 25 must, obviously, be punched on the input cards. They may, of course, be set equal to zero. If restart penalties are not used and there is a possibility that jobs may be interrupted, either words 26-28 should contain zeros or the proper overrides (Section A.1.1) must be used. Figure 48 shows a purview table with three jobs. In this example, DPUR would be punched on the input cards as 28. However, if restart penalties and P3, P4, and P5 were not needed, DPUR could be 18.

A.1.3.2.2. Status Tables.—JCR, 35, DJC, JCEND: This is a record of the status of each job computer in the system. There are five words in each row.

The first word is 1 if the job computer is busy with a job program; otherwise it is 0. The second word indicates whether or not the computer is sending an alert to the master computer. A 0 indicates that no alert is being sent. A 1 indicates that the job computer is alerting the master computer to set the CX memory at the start of execution of a job program. A 2 shows that an alert is being sent to the master computer in response to an alert previously sent by the master computer. The third and fourth words are used as temporary storage by the simulation program when a job has been interrupted. They hold, respectively, the number of the major joint which has

Priority	Interrupt Criterion			Arrival Rate			Major Joint Length			No. Major Joints			Required Modules					(If Needed)			Restart Penalty		
	F	P1	P2	F	P1	P2	F	P1	P2	F	P1	P2						P3	P4	P5	F	P1	P2
5	0	25	0	0	1,000	0	0	200	0	0	5	0	40	60	60	999								
4	0	30	0	0	2,000	0	0	150	0	0	3	0	60	60	80	110	999				2	100	150
6	0	100	0	0	5,000	0	1	100,000	200	0	20	0	999					300	450	425			

Figure 48. A PURVUE table.

been interrupted and the length of time lost in that joint because of the interruption. The fifth word is a switch used by the simulation. It is set to 2 while the job computer is working on a job, and to 1 after the job has been completed.

To start the simulation from a point at which no job computer is working on a job program, set words 1 through 4 equal to 0, and set word 5 equal to 1. (See Figure 49 for the card layout forms for a system with two job computers.) To start the simulation at any other point, it would be advisable to study the flow charts and coding carefully before constructing the JCR.

It is assumed that the job computers are numbered consecutively starting with 0, and that the rows of the array are entered in this order. JCEND/DJC must equal the number of job computers represented in the time log.

MAT, 325, DMA, MAEND: The module availability table has one entry for every controlled module in the system, including the CX memory pseudo module, but excluding the CX pseudo module. Each row gives the status of a module:

(a) the module type, coded numerically

(b) the condition of activity, where 1 means that the module is connected to a controlling module, and 0 means that the module is free

(c) the operative status, where 1 means that the module is disabled and 0 implies the reverse[16]

(d) the module assignment, where -1 means that the module is free to be assigned to any job program, 0 indicates a module used by the master program (as shown in the IMPLOG), and any other number is the job number k to which this module is assigned. Where the assignment is other than 0, the activity status is irrelevant.

The MAT rows must be in numerical order. Figure 50 depicts a sample MAT, where only one job program in the system, k = 53, has any modules assigned.

MISC, 75: This table is a collection of miscellaneous information needed by the simulation mainly for internal communication between its various subroutines. Only the first six words and the 25th word of this array need be considered by the simulation user unless a simulation run is to be

---

[16]Since there is no mechanism now in the simulation to change this status, this word should be 0 for all modules.

RUN NUMBER	ARRAY NAME (1)	CARD NO. (2)	DATA WORD NO. 1 (2)	DATA WORD NO. 2 (2)	DATA WORD NO. 3 (2)	DATA WORD NO. 4 (2)
1 6	8 13	14 16	18 30	32 44	46 58	60 72
112200	JCR		5	10		
112200	JCR	1				
112200	JCR	2	1			
112200	JCR	3				
112200	JCR	4	1			

1. JUSTIFIED TO THE LEFT.
2. JUSTIFIED TO THE RIGHT. LEADING ZEROS NEED NOT BE PUNCHED.

Figure 49. A JCR card set.

Module Type	Activity	Operative Status	Assignment
10	1	0	0
20	1	0	0
40	0	0	0
40	0	0	-1
40	0	0	53
50	0	0	0
60	0	0	-1
60	0	0	0
60	0	0	53
60	0	0	53
80	0	0	-1
80	0	0	53
90	0	0	-1
110	0	0	-1

Figure 50. An MAT table.

started at a point where the first master program joint to be performed is not 1 or 4, or there are jobs which have already entered the system. However, all 26 words of the array must be punched as input (Figure 51).

Table XXII gives the name by which each word is called in the simulation program, plus a brief explanation of its use. It is necessary to study the flow charts and coding of the program to learn the exact use of MISC (6) through MISC (25).

A.1.3.2.3. Queues.—PERBUF, 160, DPBQ, PBQEND: This queue holds the job program requests in the peripheral buffer, in the order of their arrival. Each row of the array represents a job request, and has

(a) the problem type p

(b) the sequence number L

(c) the sequence type r

(d) e, the event number in the sequence L that this problem represents

(e) the job number k.

If this occurrence of the problem p is not in connection with any sequence of jobs, L must be zero, and r and e are irrelevant. DPBQ must always be 5. When the array is empty, PBQEND must be 0.

RUN NUMBER 1 6	ARRAY NAME[1] 8 13	CARD NO.[2] 14 16	DATA WORD NO. 1[2] 18 30	DATA WORD NO. 2[2] 32 44	DATA WORD NO. 3[2] 46 58	DATA WORD NO. 4[2] 60 72
112200	MISC		26	26		
112200	MISC	1	1	1	0	80
112200	MISC	2	2	11		
112200	MISC	3				
112200	MISC	4				
112200	MISC	5				
112200	MISC	6				
112200	MISC	7		34359738367		

1. JUSTIFIED TO THE LEFT.
2. JUSTIFIED TO THE RIGHT. LEADING ZEROS NEED NOT BE PUNCHED.

Figure 51. A MISC card set.

TABLE XXII

THE MISC TABLE

Names		Function
MISC(0)	LASTL	The sequence number L to be assigned to the next sequence of jobs initiated by the input generating program. If this input scheme is not used, L is irrelevant. If L is used, it must be greater than 0.
MISC(1)	LASTK	The job number k to be assigned to the next job request put into the peripheral buffer. k must be greater than 0.
MISC(2)	COMPLT	A count of the number of completed jobs in the program queue (the QUEUE). If the program is to start at master program joint 5 or 6, the coding should be examined to determine the proper value of COMPLT.
MISC(3)	CXTIME	The switching time of the central exchange.
MISC(4)	ALPHAI	A switch used by the simulation. This should be set to 1 initially.
MISC(5)	TLX	The location of the first (in time) item in the time log. Considering the TIMELG array as a vector with subscripts starting at 0, TLX is the subscript of the first word in that TIMELG row (item) where the time, given by TIMELG(TLX+1), is the earliest time in the array.
MISC(6)	NEWP	Used by the master program to store, temporarily, information about a new job request before that job is put into the program queue.
MISC(7)	NEWL	
MISC(8)	NEWR	
MISC(9)	NEWJ	
MISC(10)	NEWPR	
MISC(11)	Q	An index of the job in the program queue that the master program is presently working on.
MISC(12)	JC	A function of the number of a job computer that the master program is considering.
MISC(13)	QBUMP	An index for a job in the program queue whose equipment the master program might appropriate for a higher-priority job.
MISC(14)	GAMMAI	Switches used by the simulation when one computer module alerts another.
MISC(15)	PHII	
MISC(16)	PII	
MISC(17)	K	The job-program number of a job being processed by the master program.
MISC(18)	PURI	Needed for temporary storage by the subroutines which assign modules to job programs.
MISC(19)	PBQU	The initial values of the totals described in Section A.1.4.
MISC(20)	BACKLG	
MISC(21)	STANBY	
MISC(22)	INPROG	
MISC(23)	COMP	
MISC(24)	ASTRSK	
MISC(25)	UNIRAN	The initial value of the random number used by the DISTI subroutine to generate succeeding random numbers and to compute the required distributions.

Figure 52 illustrates the peripheral buffer queue with five job requests. Note that the first is a request for job 1 as occurring in the sequence shown in Figure 42. Job 1 has also been requested independent of any sequence.

P	L	r	e	k
1	3	28	4	17
3	0	0	0	18
1	0	0	0	19
2	0	0	0	20
2	0	0	0	21

Figure 52. A PERBUF table.

QUEUE, 250, DQ, QEND: The program queue lists all job program being processed by the system. Jobs are added to the queue shortly after being picked up from the peripheral buffer by the master program; they are removed from the queue shortly after they have been completed by a job computer.

The queue is ordered by priority number, and, within priority, by the job number k. Each row of the queue pertains to one job program, and has the following information:

(a) the priority number

(b) the job number k

(c) the job type p

(d) 0, unless this is a job which was started and then interrupted, in which case this word contains the number of the major joint which was interrupted

(e) 0, unless this is a job which was started and then interrupted, in which case this word contains the amount of time remaining to be executed in the major joint which was interrupted

(f) the sequence number L

(g) the sequence type r

(h) the present status of the job, where

63 means the job is in the backlog,
255 means the job is in standby,
1023 means the job has been completed,
$m < 16$ means that the job is in progress and is being executed by job computer m, and
$16 \leqslant m \leqslant 31$ means that job computer m-16 is about to start the job[17]

(i) e, the event number in the sequence L that this job represents

(j) n, the total number of major joints in this program (as derived from the purview table).

If this occurrence of the problem p is not in connection with any sequence of jobs, L must be zero and r and e are irrelevant.

When the QUEUE is empty, QEND must be wet equal to zero. Whether the QUEUE is empty or not, DQ should be given as some integer $\geqslant 10$, in accordance with the following discussion. The simulation program stops running and signals an error (Section A.1.6) when the number of jobs in the QUEUE times DQ exceeds 250. In this way, one can model a queue which holds only 4 jobs by setting DQ equal to any number between 51 and 62. A maximum length queue of 25 is achieved by setting DQ = 10.

Figure 53 depicts a program queue with five jobs in it, under the following conditions:

(a) the jobs are those given in the PERBUF table in Figure 52, and defined by the PURVUE table in Figure 48

(b) job 19 has been interrupted in the middle of the second major joint and is now in stand-by

(c) job 20 is being processed by job computer 0

(d) job 21 is about to be processed by job computer 1

(e) job 17 has been completed

(f) job 18 is in the backlog.

Note that job 19, whose original priority number was 5, now has a priority number of 4 because it was interrupted.

---

[17]Before this status is assigned to a job in the queue, the flow charts and coding must be studied carefully.

Priority	k	p	Joint Interrupted	Time Left	L	r	Status	e	n
4	19	1	2	100			255		5
4	20	2					0		3
4	21	2					17		3
5	17	1			3	28	1023	4	5
6	18	3					63		20

Figure 53. A QUEUE table.

SEQQ, 112, DCQ, CQEND: The sequence queue gives the status of the sequences of jobs in progress. The sequences themselves are described by the IGPLOG; their present status is given in the SEQQ. If the IPGLOG is not being used, the SEQQ can be empty.

The sequence queue is in order by the sequence number L. For each sequence (in progress) of n events, the queue has n words:

-L
the number of events antecedent to event 2 which have not yet been completed
.
.
.
the number of events antecedent to event n which have not yet been completed.

Thus, at the initiation of each sequence, the last n-1 words are the E's of the IGPLOG.

Figure 54 illustrates a sequence queue with three sequences in progress. Sequence number 3 is of type 28 (refer to Figure 42) and has just one more event to be done before it is completed. Sequence number 4 is also of type 28 and has just been started. Sequence 5 has obviously been completed, and sequence 6 is of type 30 and has just been started.

-L				
-3	0	0	1	0
-4	1	1	2	1
-6	1	1		

Figure 54. A SEQQ table.

The rows in one SEQQ need not have the same number of words in each, and one row follows immediately after the other (Figure 55). DCQ should always be 4, even when CQEND is zero, if the sequence queue is to be used by the program.

TIMELG, 440: The time log is the mechanism which runs the simulation. There is usually one entry in the log for each element in the AN/FSQ-27 system which is operating simultaneously with, and generally unrestricted by, the other parts of the system. Each such time-related, independent activity of the AN/FSQ-27 is represented by an entry in the time log. Examples of such activities include master program functions, execution of job programs, and the arrival of job requests.

Each entry in the time log corresponds to a row of 11 words in the TIMELG array. The first three words in each row contain:

(a) i, an indicator determined by the format and contents of the log

(b) t, the time at which this event is to occur

(c) s, a code determining the interpretation of the rest of the words in the row.

CONTENT: Five different values of s are used to define the different types of entries which can be represented in the time log.

Computer Module Activity (s = 0,1).—There must be one and only one entry in the log for each computer module in the system. The last eight words in each such entry contain:

(a) a, an alert indication—0 if this computer has not received an alert from another computer; otherwise, the time at which such an alert was sent (a is reset to 0 when the alert is accepted)

(b) k, the job number

(c) p, the job type

(d) j, the next joint of the main routine to be performed (a major joint for a job computer; a joint of the basic master program loop for the master computer)

(e) J', the next joint of a subroutine to be performed (a minor joint for a job computer; the alert-accepting section of the master program for the master computer)

RUN NUMBER 1 6	ARRAY NAME[1] 8 13	CARD NO.[2] 14 16	DATA WORD NO. 1[2] 18 30	DATA WORD NO. 2[2] 32 44	DATA WORD NO. 3[2] 46 58	DATA WORD NO. 4[2] 60 72
112200	SEQQ		4	13		
112200	SEQQ	1	-3	0	0	1
112200	SEQQ	2	0	-4	1	1
112200	SEQQ	3	2	1	-6	1
112200	SEQQ	4	1			

1. JUSTIFIED TO THE LEFT.
2. JUSTIFIED TO THE RIGHT. LEADING ZEROS NEED NOT BE PUNCHED.

Figure 55. A SEQQ card set.

(f) JC, the job computer number (-1 indicates the master computer)

(g) Y, the controlled module to which the computer module is presently connected (-1 indicates that the job computer is in a major joint)

(h) n, the number of major joints in the job program.

K, p, and n are 0 for the master computer.

Figure 56 illustrates several entries representing computer module activity which could be in a time log (individually). In order they depict:

(a) a master computer, not presently connected to any controlled module, which will attempt to start joint 1 at time 5

(b) a master computer, presently connected to module 50, which is due to start joint 13 at time 17, was alerted at time 7, and would have started joint 3 at some earlier time had it not been for the occurrence and acceptance of the alert

(c) job computer 2, at present in major joint 3 (j-1) of job type 1, which is due to start joint 28 at time 8

(d) job computer 0, presently connected to module 20, and due to start major joint 9 of problem type 2 at time 4.

Note that s = 1 when the next joint to be performed is a minor joint; s = 0 when the next joint is to be a major joint.

i	t	s	a	K	p	j	j'	JC	Y	n
	5	1	0	0	0	1	0	-1	0	0
	17	1	7	0	0	3	13	-1	50	0
	8	1	0	46	1	4	28	2	-1	5
	4	0	0	43	2	9	0	0	20	10

Figure 56. Computer module TIMELG entries.

Arrival of Job Request (s = 2).—For each problem type whose arrival is not related to a job sequence as given in an IGPLOG, there is one entry in the time log. The problem type p is given in the 6th word of that row.

The use of sequences as an input device means that there is at least one entry in the log for each such sequence being used. In this case, the data following s are: an event number e, the sequence number L, and the sequence type r. When e = 1, this entry indicates the start of a new progression through the sequence r. In this case, L should be zero, as this will be assigned by the simulation program. An e greater than 1 indicates that at time t, one event in the sequence r antecedent to event e will have been completed.

Figure 57 illustrates some typical situations, where:

(a) job type 2 will be requested at time 700

(b) one of the precedence requirements[18] for event 4 in sequence type 28 (Figure 42) will have been satisfied at time 720

(c) sequence type 28 will be started again at time 20,500.

i	t	s	e	L	r or p
	700	2	0	0	2
	720	2	4	3	28
	20,500	2	1	0	28

Figure 57. Job request TIMELG entries.

Restart Procedure (s = 5).—The word after s gives the time interval at which this procedure (the punching out of the contents of the tables for restart purposes) is repeated (Section 3.3.1).

End of Simulation (s = 4).—This entry must appear in the time log. The simulation program ends at time t.

[18]To be precise, only one of the precedence requirements for event 4 could be represented by an entry in the time log. When event 5, a job request, is completed, no entry is made in the time log, whether or not event 4 can be started. Event 3, on the other hand, will always be noted by an entry in the time log at the end of the interval which event 3 represents.

FORMAT: The TIMELG array is in the form of an associative list; that is, each item gives the location of the next item, in a chain fashion. Figure 58 illustrates what might be a complete set of time log entries. There is, first, a master computer, about to start joint 1. There are two job computers not working on any job and waiting to be alerted by the master computer. There is one restart entry, and it can be seen that the restart procedure will occur twice during the running of the simulation (and once at the end). There is only one problem type which will be input to this simulation run.

After all the entries in the log have been listed, the index location (left-hand column of Figure 58) for each item should be noted. For a situation in which the IGPLOG is not in use during a particular simulation run, this is always 0 for the first item, 11 for the second, etc. Next, parameter TLX in the MISC array should be set to the index location of the first (in time) entry in the time log to be executed. In this case, one could set TLX equal to 0 or 55, but since it would be useless to start the master computing going before there is anything for it to work on, TLX should be set at 55. Then, indicator i, the first word of every row in the array, is set to the index location of the next (in time) item in the log. The last i must be -1. There must always be at least 2 items in the log.

Where the IGPLOG is in use, the situation is slightly different. In this case, there must be available to the simulation a list of available slots into which new time log entries may be placed during the simulation run. To accomplish this, a blank entry (all words equal to zero) is placed before the first entry in the log (Figure 59). This entry then has the index location of 0. The items immediately following that one have indices starting with 11, 22, etc., and the chaining of the time log proceeds as described above.

The blank entry at location zero is now the first entry of a list of blank, available items which are chained together in exactly the same fashion as the rest of the time log. The last i in this chain must also be -1. There must be a number of blank entries equal to one more than the maximum number of sequence events that could occur simultaneously.

A.1.4. OUTPUT

The principal output from the simulation program is an historical listing of the events associated with each job as it passes through the various stages of processing.

Index Location		i	t	s	a, e, or Restart Interval	k or L	p or r	j	j'	JC	Y	n
0	Master computer	33	0	1				1		-1		
11	Job computers	22	34,359,738,367						23			
22		- 1	34,359,738,367						23	1		
33	Restart	44	20,000	5	20,000							
44	End	11	50,000	4								
55	Input	0	0	2			1					

↑ Not Punched

TLX = 55

Time log entries

Figure 58. A TIMELG table.

Index Location		i	t	s	a, e, or Restart Interval	k or L	p or r	j	j'	JC	Y	n
0	Available	77										
11		44	0	1				1		-1		
22		33	34,359,738,367	1					23			
33		-1	34,359,738,367	1					23	1		
44	Time log	55	20,000	5	20,000							
55		22	50,000	4								
66		11	0	2	1	1	26					
77		88										
88	Available	99										
99		-1										

TLX = 66

Figure 59. A TIMELG table.

A.1.4.1. Printed Output

The printed output from a simulation run is usually just one page, giving the run number, the initial entries in the time log, and the times at the beginning and end of each program segment. Program segments are those intervals of computation which exclude both the initial loading of the program and the input arrays, and the punching of the simulation tables as caused by the appearance of a restart or end of simulation entry in the time log. The first program segment starts after the input data have been read in. The last program segment ends before the contents of all tables are punched out. Intermediate segments end and begin before and after (respectively) the punching of the simulation tables due to the action of the restart procedure. Both real time and simulated time are printed.

At the option of the user, all time log entries can be printed as they are inserted into the time log.

A.1.4.2. Punched Cards

Three types of cards are punched as output from the program, and these cards can be distinguished by means of a punch in column 80.

A.1.4.2.1. Time Cards (2 in Column 80).—These cards contain the same information on the times of each program segment as is printed.[19]

A.1.4.2.2. Array Cards (Blank in Column 80).—A set of array cards is punched for each restart and end of simulation entry in the time log. These cards are exactly the same in format as the input array cards, and give the contents of the simulation tables at the time that the cards are punched. The only tables punched are those that change during the program. That is, the parameter tables IGPLOG, IMPLOG, PURVUE, and MCT are not punched. The run number on the first set of these output cards is one more than the run number on the input cards, and this number is increased by one for each succeeding set of output cards.

To restart the simulation from any point at which a set of these cards is punched, it is necessary only to add to the output cards a copy of the input cards for the parameter tables IGPLOG, IMPLOG, PURVUE, and MCT with a matching run number. This complete set will then form a new input data deck.

A.1.4.2.3. Program Progress Cards (1 in Column 80).—As each problem enters the AN/FSQ-27 computer, it goes through several stages of processing (Figure 60) before it is completed. In addition to the states of progress

[19]The real times printed and punched may differ slightly.

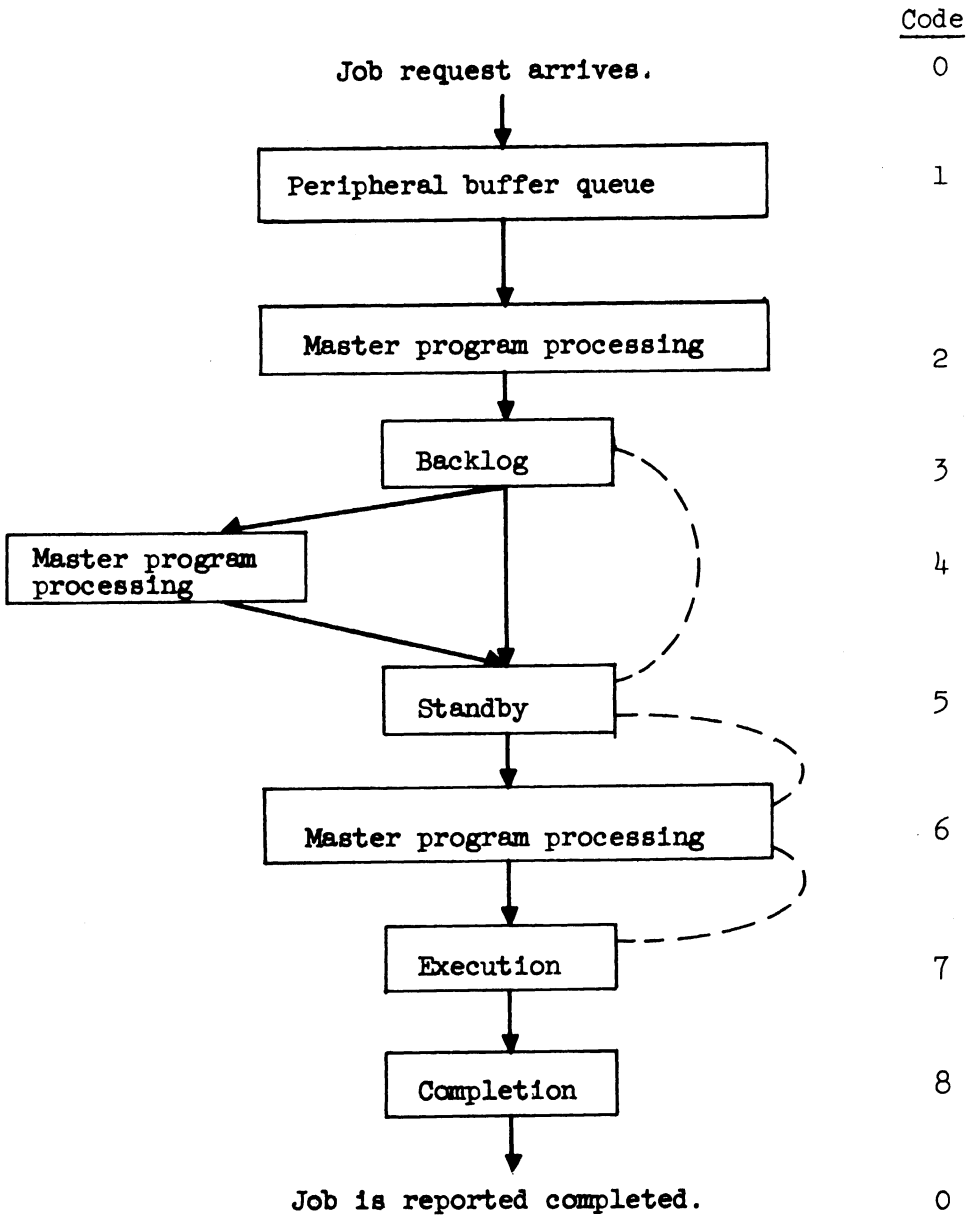

Figure 60. Job processing stages. Each job request is first received in the pheripheral buffer. It is then processed by the master program and put into the backlog. Some additional processing may or may not be required before the job is placed in the stand-by list. Master program processing is always necessary before execution can begin, and there is always a delay after the execution of a job before the master computer can send a completion message to the system operator. The dotted lines indicate the possible regression steps of a job because of its pre-emption by a higher-ranking job.

of a job shown in Figure 7 (Section 3.2.1), there are four more considered by the program. Three of these are the intermediary stages, when the master program is moving the job from the peripheral buffer to the backlog, from the backlog to the stand-by list, and from the stand-by list to the stage of actual execution of the job program. The fourth is the unimportant one where the job program is not in the computer at all. These stages are given numerical codes.

A program progress card (Figure 61) is punched for each job at every point at which the job changes stages. The time of this change, the processing stages involved, and job-program identification are punched. The processing stage code numbers form the first and third digits of a three-digit field (columns 10-12) on the card. If the job's transition between these two stages is accomplished at the cost of retrograding another job, the middle digit in this field will be a 1. Otherwise, the digit will be a 0. All the possible three-digit configurations are shown in Table XXIII, together with a brief description of the meaning of the configuration and its place of origin in the simulation program.

The simulation program maintains running totals of the number of problems in the system in each stage of processing, for stages, 1, 3, 5, 7, and 2, 4, and 6 combined. These totals, which change only as each job changes stages, are also punched on each program progress card.

Pseudo program progress cards are punched if the user of the simulation program has taken the option of eliminating the master program's repetition of the loop between joints 1 and 4 (Section A.1.1). In this case, a card is punched whenever the master program leaves or enters (effectively the simulated AN/FSQ-27 system. The three-digit identification codes (card columns 10-12) used are 706 and 607, respectively. The rest of the information on the card is as described in Figure 61. K and p are both 0.

A.1.5. TIMING

The speed of the simulation program is almost completely a function of the number of entries made in the time log. The time to process one time-log entry depends on a large number of factors, primarily the number of modules in the system, the number of items in the time log at any given time, and the types of distribution functions used. A figure of 10 to 15 milliseconds per time-log entry should give a reasonable estimate of the time for a given set of input cards. Note that moving down (in time) an item in the log because of the unavilability of a subordinate module constitutes another entry in the log.

Card Columns	Information	704 Names in NOTE Subroutine
1-6	The run number (as given on the input cards).	RUNNUM
10	The processing stage the job is leaving.	CODEA
11	An indication of whether or not another job has been retrograded to accomplish this change of stages. A 1 means yes; a 0, no.	CODEB
12	The processing stage the job is entering.	CODEC
18-29	The time at which the job is changing stages.	TIMELG(TLX+1)
33-35	The job number k.	KPRINT
38-39	The job type p.	PPRINT
49-52	The number of jobs in the peripheral buffer (stage 1).	PBQU
53-56	The number of jobs in the backlog (stage 3).	BACKLG
57-60	The number of jobs in the stand-by list (stage 5).	STANBY
61-64	The number of jobs being executed (stage 7).	INPROG
65-68	The number of jobs which have been completed, but not yet reported out (stage 8).	COMP
69-72	The number of jobs in all other processing phases (stages, 2, 4, and 6).	ASTRSK
80	1	

Figure 61. Program progress cards (primary simulation output).

TABLE XXIII

THE IDENTIFICATION OF A JOB'S PROGRESS ON A PROGRAM PROGRESS CARD

Identification	Origin in Program*	Meaning
001	IGP routine	The job has just arrived at the peripheral buffer.
102	MP joint 1	The MP has removed the job from the PB.
203	MP joint 3	The job has been placed in the backlog.
305	MP joints 4, 6, 9, 10, or 14	The job has been moved from the backlog to the stand-by list.
314	MP joints 4, 6, 9, 10, or 14	This job has been removed from the backlog, and the MP is alerting another job (in progress) to pre-empt its modules.
315	MP joints 4, 6, 9, 10, or 14	The job has been moved from the backlog to the stand-by list, while another job has been moved in the reverse direction.
415	MP joint 9	This notation will always follow a 314 card, and means that the alerted job has responded and has given up its modules. This job is now in the stand-by list.
503	MP joints 4, 6, 9, 10, or 14	This job has been moved to the backlog from stand-by.
506	MP joints 4, 6, 9, 10, 14, or 16	This job is about to be started. Either the MC is alerting a free JC, or a JC has picked up this job on its own from the program queue.
516	MP joints 4, 6, 9, 10, or 14	The MP is alerting a JC to stop processing its lower-ranking job to start work on this one.
605	MP joint 25	This job is put back into the stand-by list after being interrupted. This notation always follows a 706 card.
607	MP joint 19	This job has started execution.
706	MP joint 24	This job has just been interrupted.
708	MP joints 20 or 24	This job has been been completed.
800	MP joint 5	The MP is removing this job from the program queue and reporting its completion to the system operator via the peripheral buffer.

*When the output originates in the master program, the time punched on the card is the time at which the CX time and the access time to the subordinate module have been computed, but before the time to perform the MP function has been considered.

A.1.6. SCHEDULED ERROR STOPS

There are several points in the program at which an error may occur and be detected by the program itself. At these points, the program stops and a memory dump is taken. There are other, unscheduled errors which cause the program to end prematurely, and in some of these there is also a memory dump printout. In the former instances, however, the dumps give precise information about the cause of the error stop; namely, the two's complement of the address of the order which transfers control from the simulation program to the memory dump subroutine is found in index register 4, or C. Figure 62 lists the relative addresses of the entries to the dump routine, together with the absolute addresses of those words which might be a clue as to the source of the difficulty. The first three error stops listed occur because of an error in the input cards, and, in the first two cases, the program is stopped before any printed output from that run occurs. The last four error stops listed occur because a queue has exceeded capacity. These occur only after the output page for that run has been printed. Table XXIV will aid in the debugging of all types of errors, as it lists the locations in memory of all the tables which contain the simulation parameters. Addresses in both tables are given in octal.

TABLE XXIV

SIMULATION TABLES

Name	Octal Equivalent of Name	Location of First Word*
IGPLOG	+312747434627	333
IMPLOG	+314447434627	554
JCR	-012351606060	617
MAT	-042163606060	1324
MCT	-042363606060	1514
MISC	-043162236060	1627
PERBUF	-072551226426	2067
PURVUE	-076451656425	2647
QUEUE	-106425642560	3241
SEQQ	-222550506060	3421
TIMELG	-233144254327	4311

*Arrays are stored, in a sense, backwards. If the address of the first element is x, succeeding elements will be found in x-1, x-2, - etc. The information stored in these locations does not include the words on cards numbered zero in the input deck.

Name of Routine in Which Error is Detected	Relative Address of Exit to Dump Subroutine	Meaning of Error	Significant Information		
			Absolute Address*	Meaning of Contents of Address	Name of Variable
START	276	The card just read did not have the expected run number, array name, or card number. The first card (numbered 000) of an array was expected here.	4455 4456 4675 4700 4701 4673	Run no. as given on the run no. input card. Run no. as given on the last card read. Array name as given on the last card read. Card no. as given on the last card read. First data word on the last card read. Second data word on the last card read.	RUNNO RUNNUM ARRAY CARDNO DAR AREND
START	703	The card just read did not have the expected run number, array name, or card number. A card other than the first of an array was expected here.	4455 4456 4675 4674 4700 4677	Run no. as given on the run no. input card. Run no. as given on the last card read. Array name as given on the first card of the array. Array name as given on the last card read. Card no. on the next to last card read. Card no. on the next to last card read.	RUNNO RUNNUM ARRAY ARRAY1 CARDNO CARDNI
START	1100	All of the input cards have been read in, but one of the arrays has exceeded the space allotted to it in memory.	4347... 4335	These locations contain the no. of words in each array (in alphabetical order) as given on the input cards. Check these figures against the dimension statements in the MAD program and/or the figures given in Section A.1.3.	INPUT(45)... INPUT(55)
MPMIN	1032	The no. of items in the program queue has exceeded capacity.	4437 4434	Present no. of words in the program queue. Maximum no. of words allowed in the queue.	QEND QMAX
TIMLG	233	This error can occur only when the IGPLOG is being used. This stop results when a space is needed in the time log for a new entry.			
IGP	700	The no. of items in the peripheral buffer queue has exceeded capacity. The no. of words in the sequence queue has exceeded capacity.	4341 4433 4336 4432	Present no. of words in the PB queue. Maximum no. of words allowed in the queue. Present no. of words in the sequence queue. Maximum no. of words allowed in the sequence queue.	PBQUEND PBQMAX CQEND CQMAX

*The addresses given here, where they are derived from relative addresses, may change if any part of the program deck is changed by anything other than the overrides listed in Appendix A.1.

Figure 62. Debugging aids.

A.2.1. THE MAIN1 ROUTINE

The MAIN1 routine (Table XXV) is the main program for the first core load. The code examines the first entry in the time log and performs one of the following actions:

(a) enters the START routine to read in the input cards, and transfers control to the second core load

(b) enters the START routine to punch out the contents of the simulation tables, and either returns control to the second core load or performs (a) above.

A.2.2. THE START SUBROUTINE

This subroutine (Table XXVI) has two entries, START (Figure 63) and RESTAR (Figure 64). The first is used at the beginning of the program to read in the input deck. Checks are made to see that the format of the card is correct. The entry RESTAR is used whenever a restart or end of simulation item is found in the time log. The contents of the arrays are then punched out.

A.2.3. THE TIMELG SUBROUTINE

The TIMELG subroutine (Table XXVII) is concerned with the mechanical arranging of the entries in the time log. There are seven entries to this subroutine:

(a) FINDJC - The time log is searched for a particular job computer.

(b) FINDMC - The time log is searched for the master computer.

(c) MOVE1 - The time-log item at the top of the log has had its time changed by some subroutine. This item is moved to its proper new place.

(d) MOVET - The time-log item at a given position in the log has had its time changed by some subroutine. This item is moved to its proper new place.

(e) PUTT - A new item is inserted into the log.

TABLE XXV

MAIN1 ROUTINE

```
*COMPILE MAD,PUNCH OBJECT                                                MAIN1000
          NORMAL MODE IS INTEGER
          EQUIVALENCE(INPUT(30),DIG),(INPUT(31),DMP),(INPUT(32)
         1 ,DJC),(INPUT(33),DMA),(INPUT(34),DMC),(INPUT(36),DPBQ)
         2 ,(INPUT(37),DPUR),(INPUT(38),DQ),(INPUT(48),MAEND),(IN
         3 PUT(51),PBQEND),(INPUT(53),QEND),(INPUT(54),CQEND),(
         4 MISC(0),LASTL),(MISC(1),LASTK),(MISC(2),COMPLT),(MISC
         5 (3),CXTIME),(MISC(4),ALPHAI),(MISC(5),TLX),(MISC(6),NEWP),(
         6 MISC(7),NEWL),(MISC(8),NEWR),(MISC(9),NEWJ),(MISC(10),NEWPR
         7 ),(MISC(11),Q),(MISC(12),JC),(MISC(13),QBUMP),(MISC(14),
         8 GAMMAI),(MISC(15),PHII),(MISC(16),PII),(MISC(17),K),
         9 (MISC(18),PUR1) ,(INPUT(47),JCEND)
          EQUIVALENCE (MISC(19),PBQU),(MISC(20),BACKLG),(MISC(21),STA
         1 NBY),(MISC(22),INPROG),(MISC(23),COMP),(MISC(24),ASTRSK)
          PROGRAM COMMON IGPLOG,IMPLOG,JCR,MAT,MCT,MISC,PERBUF,PURVUE
         1 ,QUEUE,SEQQ,TIMELG,INPUT,INFIN,TL,J,L,R,CQMAX,PBQMAX,QMAX,T
         2 IME,PUR,OLDTL,OLDTLX,JX,MP,TIME3,RUNNO,RUNNUM
          DIMENSION IGPLOG(195),IMPLOG(144),JCR(34),MAT(324),
         1 MCT(119),INPUT(74),PERBUF(159),PURVUE(367),QUEUE(249),
         2 SEQQ(111),TIMELG(439),MISC(74),TIME3(9)
          TRANSFER TO DO (TIMELG (TLX +2))
DO(0)     EXECUTE START.
          INFIN=34359738367
          PBQMAX=32*DPBQ
          QMAX=250
          CQMAX=112
GO        EXECUTE SEQPGM.
DO(4)     EXECUTE RESTAR.
          TRANSFER TO DO(0)
DO(5)     EXECUTE RESTAR.
          TRANSFER TO GO
          VECTOR VALUES TIME3=$1H0,I4,I12,I2,I12,I15,I3,)15,
         1 I10,I13,I4,I10,I19 *$
          END OF PROGRAM
```

TABLE XXVI

START SUBROUTINE

```
*COMPILE MAD, DUMP, PUNCH OBJECT                                        START000
          EXTERNAL FUNCTION
          ENTRY TO START.
          NORMAL MODE IS INTEGER
          EQUIVALENCE(INPUT(30),DIG),(INPUT(31),DMP),(INPUT(32)
         1 ,DJC),(INPUT(33),DMA),(INPUT(34),DMC),(INPUT(36),DPBQ)
         2 ,(INPUT(37),DPUR),(INPUT(38),DQ),(INPUT(48),MAEND),(IN
         3 PUT(51),PBQEND),(INPUT(53),QEND),(INPUT(54),CQEND),(
         4 MISC(0),LASTL),(MISC(1),LASTK),(MISC(2),COMPLT),(MISC
         5 (3),CXTIME),(MISC(4),ALPHAI),(MISC(5),TLX),(MISC(6),NEWP),(
         6 MISC(7),NEWL),(MISC(8),NEWR),(MISC(9),NEWJ),(MISC(10),NEWPR
         7 ),(MISC(11),Q),(MISC(12),JC),(MISC(13),QBUMP),(MISC(14),
         8 GAMMAI),(MISC(15),PHII),(MISC(16),PII),(MISC(17),K),
         9 (MISC(18),PUR1) ,(INPUT(47),JCEND)
          EQUIVALENCE (MISC(19),PBQU),(MISC(20),BACKLG),(MISC(21),STA
         1 NBY),(MISC(22),INPROG),(MISC(23),COMP),(MISC(24),ASTRSK)
          PROGRAM COMMON IGPLOG,IMPLOG,JCR,MAT,MCT,MISC,PERBUF,PURVUE
         1 ,QUEUE,SEQQ,TIMELG,INPUT,INFIN,TL,J,L,R,CQMAX,PBQMAX,QMAX,T
         2 IME,PUR,OLDTL,OLDTLX,JX,MP,TIME3,RUNNO,RUNNUM
          DIMENSION IGPLOG(195),IMPLOG(144),JCR(34),MAT(324),
         1 MCT(119),INPUT(74),PERBUF(159),PURVUE(367),QUEUE(249),
         2 SEQQ(111),TIMELG(439),MISC(74),TIME3(9)
          VECTOR VALUES INPUT = $IGPLOG$,$IMPLOG$,$JCR$,$MAT$,$MCT$
         1 ,$MISC$,$PERBUF$,$PURVUE$,$QUEUE$,$SEQQ$,$TIMELG$
          THROUGH S581,FOR IN=15,1,IN.E.26
S581       INPUT(IN)=$1$
          VECTOR VALUES ALL = $I6, S1, C6, I3, 4(S1,I13)*$
          READ FORMAT ALL, RUNNO
S 58      READ FORMAT ALL, RUNNUM,ARRAY, CARDNO, DAR,AREND
S 59      THROUGH S 59,FOR IN= 0,1,IN.E.15.OR.RUNNUM.E.RUNNO.AND.
         1CARDNO.E.0.AND.INPUT(IN).E.ARRAY
          WHENEVER IN .E. 15
S63       EXECUTE ERROR.
          END OF CONDITIONAL
          WORDNO = 0
          INPUT(IN+30) = DAR
          INPUT(IN+45) = AREND
          INPUT(IN+15) = $ $
S 60      WHENEVER WORDNO.GE.AREND,TRANSFER TO S 64
          CARDCT = WORDNO
          WORDNO=WORDNO+DAR
S 61      WHENEVER CARDCT+3.GE.WORDNO
          END = WORDNO-1
          OTHERWISE
          END = CARDCT+3
          END OF CONDITIONAL
          TRANSFER TO GET(IN)
GET(0)     READ FORMAT ALL,RUNNUM,ARRAY1,CARDN1,IGPLOG(CARDCT)...IGPLOG
         1 (END)
          TRANSFER TO S 62
GET(1)     READ FORMAT ALL,RUNNUM,ARRAY1,CARDN1,IMPLOG(CARDCT)...IMPLOG
         1 (END)
          TRANSFER TO S 62
GET(2)     READ FORMAT ALL,RUNNUM,ARRAY1,CARDN1,JCR    (CARDCT)...JCR
         1 (END)
          TRANSFER TO S 62
GET(3)     READ FORMAT ALL,RUNNUM,ARRAY1,CARDN1,MAT    (CARDCT)...MAT
         1 (END)
          TRANSFER TO S 62
```

TABLE XXVI (Continued)

11-13-61

```
GET(4)     READ FORMAT ALL,RUNNUM,ARRAY1,CARDN1,MCT    (CARDCT)...MCT
          1 (END)
            TRANSFER TO S 62
GET(5)     READ FORMAT ALL,RUNNUM,ARRAY1,CARDN1,MISC   (CARDCT)...MISC
          1 (END)
            TRANSFER TO S 62
GET(6)     READ FORMAT ALL,RUNNUM,ARRAY1,CARDN1,PERBUF(CARDCT)...PERBUF
          1 (END)
            TRANSFER TO S 62
GET(7)     READ FORMAT ALL,RUNNUM,ARRAY1,CARDN1,PURVUE(CARDCT)...PURVUE
          1 (END)
            TRANSFER TO S 62
GET(8)     READ FORMAT ALL,RUNNUM,ARRAY1,CARDN1,QUEUE  (CARDCT)...QUEUE
          1 (END)
            TRANSFER TO S 62
GET(9)     READ FORMAT ALL,RUNNUM,ARRAY1,CARDN1,SEQQ   (CARDCT)...SEQQ
          1 (END)
            TRANSFER TO S 62
GET(10)    READ FORMAT ALL,RUNNUM,ARRAY1,CARDN1,TIMELG(CARDCT)...TIMELG
          1 (END)
            TRANSFER TO S 62
S 62        WHENEVER RUNNO .NE.RUNNUM.OR.ARRAY.NE.ARRAY1.OR.CARDN
          1 1-1.NE.CARDNO, EXECUTE ERROR.
            CARDNO = CARDN1
            CARDCT = CARDCT+4
            WHENEVER CARDCT .GE.WORDNO,TRANSFER TO S 60
            TRANSFER TO S 61
S 64        THROUGH S64 ,FOR IN=15,1,IN.E.26.OR.INPUT(IN).E.$1$
            WHENEVER IN.NE.26, TRANSFER TO S58
           PRINT FORMAT T1,RUNNO
           VECTOR VALUES T1=$27H1THE NUMBER OF THIS RUN IS ,I6*$
           PRINT FORMAT T2
           VECTOR VALUES T2=$61H4THE INITIAL VALUES IN THE TIME LOG ARE
          1(IN ORDER BY TIME)---*$
           PRINT FORMAT T3
           VECTOR VALUES T3=$101H0          TIME    S           A
          1      K  P              J      J-PRIME         JC   Y
          2N*$
           PRINT FORMAT T4
           VECTOR VALUES T4=$1H *$
           THROUGH S103,FOR TL=TLX,(TIMELG(TL)-TL),TL.E.-1
S103       PRINT FORMAT TIME3,0,TIMELG(TL+1)...TIMELG(TL+10)
           PRINT FORMAT T5
           VECTOR VALUES T5=$106H4THE FIGURES BELOW GIVE REAL TIME (IN T
          1HOUSANDTHS OF AN HOUR) AND SIMULATED TIME FOR EACH PROGRAM SE
          2GMENT.*$
           PRINT FORMAT T6
           VECTOR VALUES T6=$42H0                          REAL     SIMULAT
          1ED*$
           WHENEVER INPUT(45).G .196.OR.INPUT(46).G.145.OR.INPUT(47)
          1 .G .35.OR.INPUT(48).G .325.OR.INPUT(49).G .120.OR.INPUT(50)
          2.G.75.OR.INPUT(51).G .160.OR.INPUT(52).G .368.OR.INPUT(53)
          3 .G .250.OR.INPUT(54).G .112.OR.INPUT(55).G .440,
          4 EXECUTE ERROR.
            FUNCTION RETURN
            ENTRY TO RESTAR.
            TIMELG(TLX+1)=TIMELG(TLX+1)+TIMELG(TLX+3)
            EXECUTE MOVE1.
```

```
            VECTOR VALUES INPUT(60)=0,0,1,1,0,1,1,0,1,1,1,0,0,0,0
            RUNNO = RUNNO +1
            PUNCH FORMAT ALL,RUNNO
           THROUGH S104,FOR IN =0,1,IN.E.15
           WHENEVER INPUT(IN+60).NE.1,TRANSFER TO S104
S99         CARDNO=0
            DAR=INPUT(IN+30)
            AREND= INPUT(IN+45)
            WORDNO =0
            PUNCH FORMAT ALL,RUNNO,INPUT(IN),CARDNO,DAR,AREND
S100       WHENEVER WORDNO .GE.AREND,TRANSFER TO S104
            CARDCT = WORDNO
            WORDNO = WORDNO + DAR
S101        WHENEVER CARDCT+3.GE.WORDNO
            END=WORDNO-1
            OTHERWISE
            END=CARDCT+3
            END OF CONDITIONAL
            CARDNO = CARDNO+1
            TRANSFER TO PUT(IN)
PUT(0)      PUNCH FORMAT ALL,RUNNO,INPUT(IN),CARDNO,IGPLOG(CARDCT)...
          1 IGPLOG(END)
            TRANSFER TO S102
PUT(1)      PUNCH FORMAT ALL,RUNNO,INPUT(IN),CARDNO,IMPLOG(CARDCT)...
          1 IMPLOG(END)
            TRANSFER TO S102
PUT(2)      PUNCH FORMAT ALL,RUNNO,INPUT(IN),CARDNO,JCR   (CARDCT)...
          1 JCR   (END)
            TRANSFER TO S102
PUT(3)      PUNCH FORMAT ALL,RUNNO,INPUT(IN),CARDNO,MAT   (CARDCT)...
          1 MAT   (END)
            TRANSFER TO S102
PUT(4)      PUNCH FORMAT ALL,RUNNO,INPUT(IN),CARDNO,MCT   (CARDCT)...
          1 MCT   (END)
            TRANSFER TO S102
PUT(5)      PUNCH FORMAT ALL,RUNNO,INPUT(IN),CARDNO,MISC  (CARDCT)...
          1 MISC  (END)
            TRANSFER TO S102
PUT(6)      PUNCH FORMAT ALL,RUNNO,INPUT(IN),CARDNO,PERBUF(CARDCT)...
          1 PERBUF(END)
            TRANSFER TO S102
PUT(7)      PUNCH FORMAT ALL,RUNNO,INPUT(IN),CARDNO,PURVUE(CARDCT)...
          1 PURVUE(END)
            TRANSFER TO S102
PUT(8)      PUNCH FORMAT ALL,RUNNO,INPUT(IN),CARDNO,QUEUE (CARDCT)...
          1 QUEUE (END)
            TRANSFER TO S102
PUT(9)      PUNCH FORMAT ALL,RUNNO,INPUT(IN),CARDNO,SEQQ  (CARDCT)...
          1 SEQQ  (END)
            TRANSFER TO S102
PUT(10)     PUNCH FORMAT ALL,RUNNO,INPUT(IN),CARDNO,TIMELG(CARDCT)...
          1 TIMELG(END)
            TRANSFER TO S102
S102        CARDCT=CARDCT+4
            WHENEVER CARDCT.GE.WORDNO,TRANSFER TO S100
            TRANSFER TO S101
S104       CONTINUE
            FUNCTION RETURN
           END OF FUNCTION
```

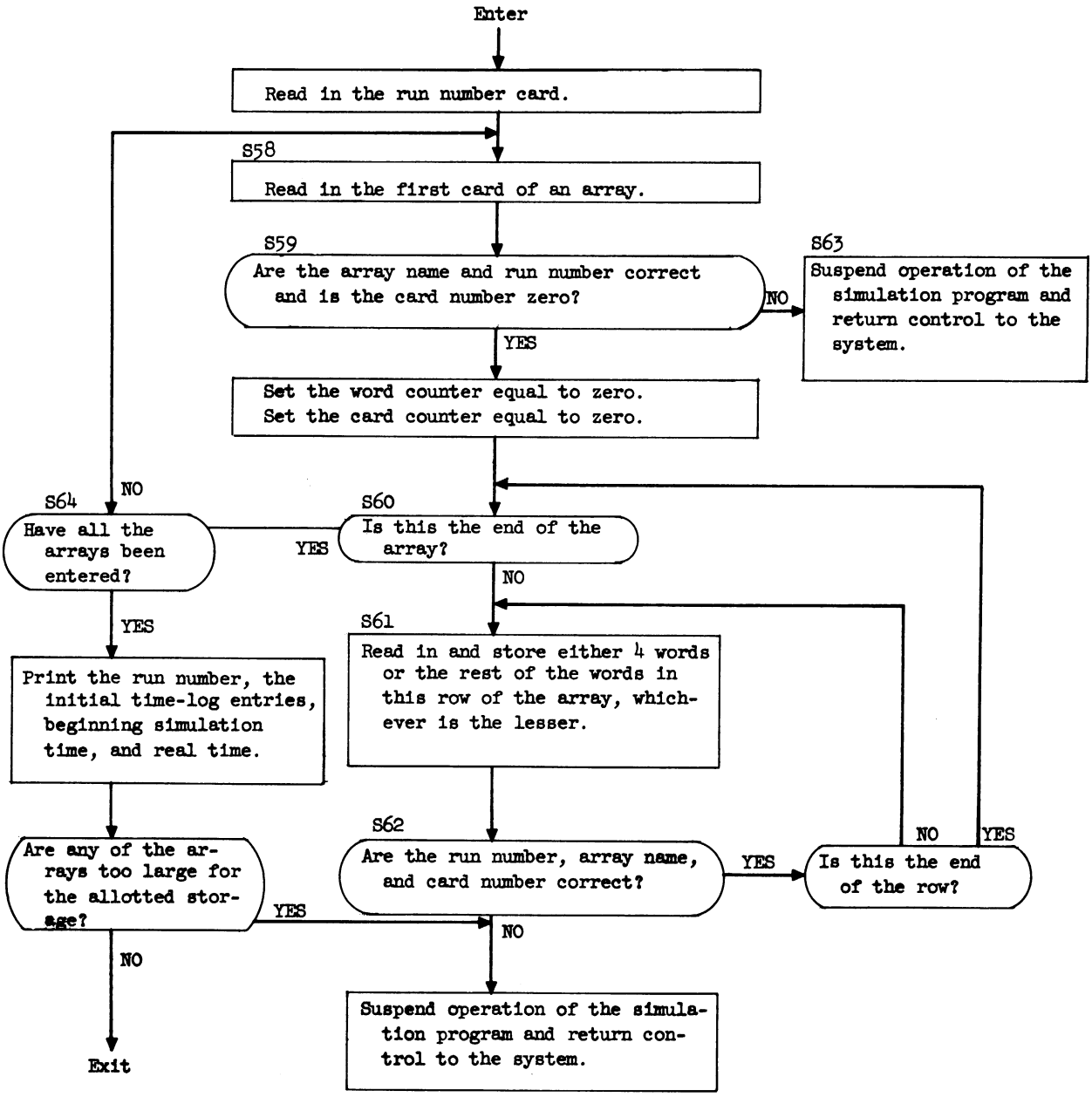

Figure 63. START subroutine—entry to START.

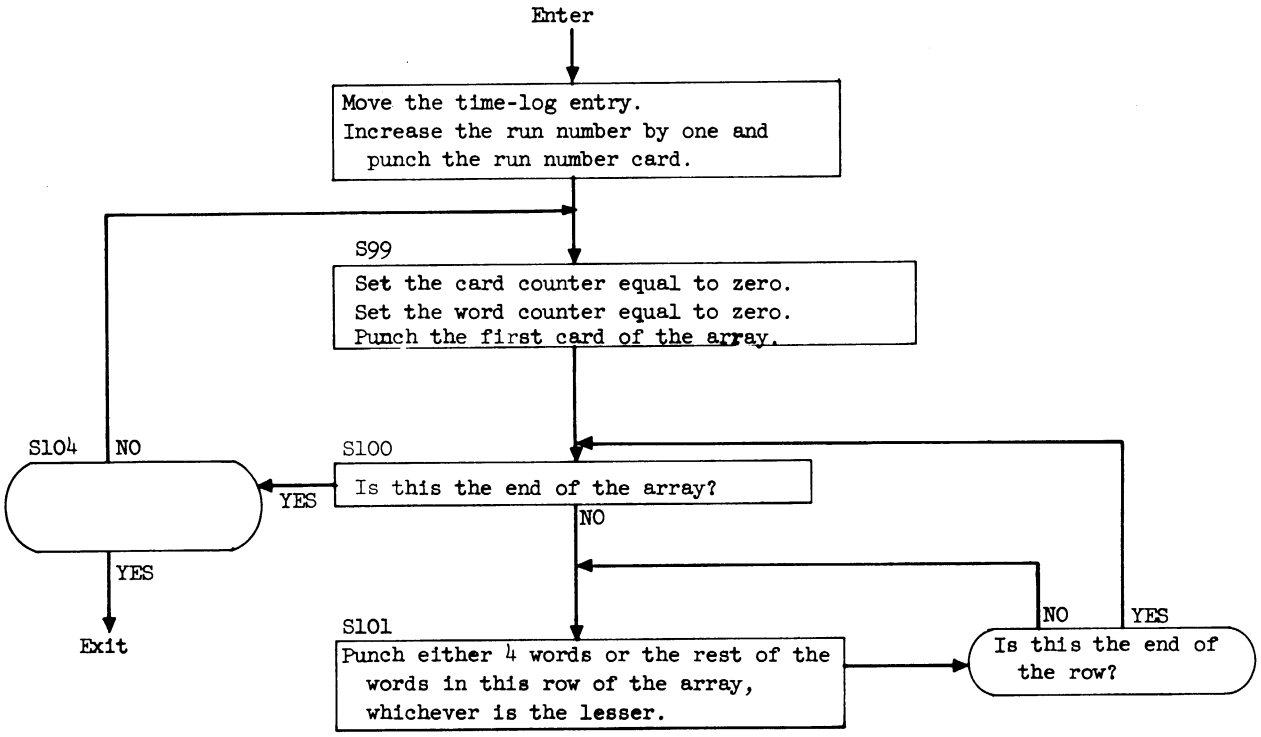

Figure 64. START subroutine—entry to RESTAR.

TABLE XXVII

TIMELG SUBROUTINE

```
*COMPILE MAD,PUNCH OBJECT                                                TIMLG000
          EXTERNAL FUNCTION                                                      537
          ENTRY TO FINDJC.                                                       538
         NORMAL MODE IS INTEGER
          EQUIVALENCE(INPUT(30),DIG),(INPUT(31),DMP),(INPUT(32)
         1 ,DJC),(INPUT(33),DMA),(INPUT(34),DMC),(INPUT(36),DPBQ)
         2 ,(INPUT(37),DPUR),(INPUT(38),DQ),(INPUT(48),MAEND),(IN
         3 PUT(51),PBQEND),(INPUT(53),QEND),(INPUT(54),CQEND),(
         4 MISC(0),LASTL),(MISC(1),LASTK),(MISC(2),COMPLT),(MISC
         5 (3),CXTIME),(MISC(4),ALPHAI),(MISC(5),TLX),(MISC(6),NEWP),(
         6 MISC(7),NEWL),(MISC(8),NEWR),(MISC(9),NEWJ),(MISC(10),NEWPR
         7 ),(MISC(11),Q),(MISC(12),JC),(MISC(13),QBUMP),(MISC(14),
         8 GAMMAI),(MISC(15),PHII),(MISC(16),PII),(MISC(17),K),
         9 (MISC(18),PUR1) ,(INPUT(47),JCEND)
          EQUIVALENCE (MISC(19),PBQU),(MISC(20),BACKLG),(MISC(21),STA
         1 NBY),(MISC(22),INPROG),(MISC(23),COMP),(MISC(24),ASTRSK)
          PROGRAM COMMON IGPLOG,IMPLOG,JCR,MAT,MCT,MISC,PERBUF,PURVUE
         1 ,QUEUE,SEQQ,TIMELG,INPUT,INFIN,TL,J,L,R,CQMAX,PBQMAX,QMAX,T
         2 IME,PUR,OLDTL,OLDTLX,JX,MP,TIME4,RUNNO,RUNNUM
          DIMENSION IGPLOG(195),IMPLOG(144),JCR(34),MAT(324),
         1 MCT(119),INPUT(74),PERBUF(159),PURVUE(367),QUEUE(249),
         2 SEQQ(111),TIMELG(439),MISC(74),TIME3(9)
          JCNUM = JC
S0        TL=TLX
S1        WHENEVER TIMELG(TL+2).GE.2.OR.TIMELG(TL+8).NE.JCNUM
          OLDTL=TL                                                               541
          TL=TIMELG(TL)                                                          542
          TRANSFER TO S1                                                         543
          END OF CONDITIONAL                                                     544
          FUNCTION RETURN                                                        545
          ENTRY TO FINDMC.
          JCNUM = -1
          TRANSFER TO S0
          ENTRY TO MOVE1.                                                        546
S4        TL=TIMELG(TLX)                                                         547
          WHENEVER TIMELG(TL+1).G.TIMELG(TLX+1)
          PRINT FORMAT TIME3,TIMELG(TLX)...TIMELG(TLX+10),RUNNO
          TRANSFER TO S3
          END OF CONDITIONAL
          OLDTLX=TLX                                                             549
          TLX=TL                                                                 550
S2        OLDTL=TL                                                               551
          TL=TIMELG(TL)                                                          552
          WHENEVER TL.NE.-1.AND.TIMELG(TL+1).LE.TIMELG(OLDTLX+1),
         1 TRANSFER TO S 2
          TIMELG(OLDTLX)=TL                                                      554
          PRINT FORMAT TIME3,TIMELG(OLDTLX)...TIMELG(OLDTLX+10),RUNNO
          TIMELG(OLDTL)=OLDTLX                                                   555
S3       FUNCTION RETURN
          ENTRY TO MOVET.                                                        557
          TIMELG(OLDTL)=TIMELG(TL)                                               558
          ENTRY TO PUTT.                                                         562
S5        TIMELG(TL)=TLX                                                         559
          TLX=TL                                                                 560
          TRANSFER TO S4                                                         561
          ENTRY TO FINDAV.                                                       564
          TL=TIMELG(0)                                                           565
          WHENEVER TL.NE.-1, TRANSFER TO S051                                    566
          EXECUTE ERROR.                                                         567
```

TABLE XXVII (Concluded)

```
S051        TIMELG(0)=TIMELG(TL)                                    568
            FUNCTION RETURN                                         569
            ENTRY TO ELIM1.                                         570
            NEWTLX=TIMELG(TLX)                                      571
            TIMELG(TLX)=TIMELG(0)                                   572
            TIMELG(0)=TLX                                           573
            TLX=NEWTLX                                              574
            FUNCTION RETURN                                         575
            END OF FUNCTION                                         576
```

(f) FINDAV - An available space, to be used by the IGP subroutime, is found in the log. If there is no such space, the operation of the simulation program is suspended and control is returned to the sytem.

(g) ELIM1 - The first item in the log is returned to the available list.

A.2.4. THE MAIN2 ROUTINE

The MAIN2 routine (Table XXVIII) is the main program for the second core load. This routine prints simulation time and real time, examines the first entry in the time log, and then performs one of the following actions (Figure 10):

(a) enters the JPMAJ subroutine

(b) enters the MPMIN subroutine

(c) enters the IGP routine

(d) prints simulation time and real time and returns control to the first core load (to MAIN1).

A.2.5. THE ALERT SUBROUTINE

This code (Table XXIX) has three entries. ALERTJ (Figure 65) is entered to alert a job computer. ALERTM (Figure 66) is entered to alert the master computer. ALERTF executes a pseudo alert to the master computer. When the simulation user has taken the option of eliminating the repetition of the joint 1-joint 4 loop by the master computer (Section A.1.1), the MC is notified of the need to re-enter the loop by the use of the ALERTF code.

A.2.6. THE DIST SUBROUTINE

The DIST subroutine (Figure 67, Table XXX) performs the functions described in Section A.1.2 and consists of five main sections and two subroutines.

The main sections of the DIST subroutine consist of a switch section (SWITCH) and a section for each of the four distributions (CODE 0,...,CODE 3). The SWITCH section determines from the code digit, F, which of the CODE sections is being called upon to supply a stochastic value, and transfers control to the appropriate CODE section. Sections CODE 0 and CODE 1 simply perform the functions indicated in Table XXI.

TABLE XXVIII

MAIN2 ROUTINE

```
*COMPILE MAD,PUNCH OBJECT                                              MAIN2000
          NORMAL MODE IS INTEGER
          EQUIVALENCE(INPUT(30),DIG),(INPUT(31),DMP),(INPUT(32)
         1 ,DJC),(INPUT(33),DMA),(INPUT(34),DMC),(INPUT(36),DPBQ)
         2 ,(INPUT(37),DPUR),(INPUT(38),DQ),(INPUT(48),MAEND),(IN
         3 PUT(51),PBQEND),(INPUT(53),QEND),(INPUT(54),CQEND),(
         4 MISC(0),LASTL),(MISC(1),LASTK),(MISC(2),COMPLT),(MISC
         5 (3),CXTIME),(MISC(4),ALPHAI),(MISC(5),TLX),(MISC(6),NEWP),(
         6 MISC(7),NEWL),(MISC(8),NEWR),(MISC(9),NEWJ),(MISC(10),NEWPR
         7 ),(MISC(11),Q),(MISC(12),JC),(MISC(13),QBUMP),(MISC(14),
         8 GAMMAI),(MISC(15),PHII),(MISC(16),PII),(MISC(17),K),
         9 (MISC(18),PUR1) ,(INPUT(47),JCEND)
          EQUIVALENCE (MISC(19),PBQU),(MISC(20),BACKLG),(MISC(21),STA
         1 NBY),(MISC(22),INPROG),(MISC(23),COMP),(MISC(24),ASTRSK)
          PROGRAM COMMON IGPLOG,IMPLOG,JCR,MAT,MCT,MISC,PERBUF,PURVUE
         1 ,QUEUE,SEQQ,TIMELG,INPUT,INFIN,TL,J,L,R,CQMAX,PBQMAX,QMAX,SP
         2 ARE,PUR,OLDTL,OLDTLX,JX,MP,TIME3,RUNNO,RUNNUM
          DIMENSION IGPLOG(195),IMPLOG(144),JCR(34),MAT(324),
         1 MCT(119),INPUT(74),PERBUF(159),PURVUE(367),QUEUE(249),
         2 SEQQ(111),TIMELG(439),MISC(74),TIME3(9)
          VECTOR VALUES P1 = $S10,12HBEGIN TIME =,I6,I12,I40*$
          VECTOR VALUES P2 =  $S3,19HINTERMEDIATE TIME =,I6,I12,I40*$
         PRINT FORMAT TIME3
         PRINT FORMAT P1,TIME.(0),TIMELG(TLX+1)
          PUNCH FORMAT P1,TIME.(0),TIMELG(TLX+1),2
          REWIND TAPE 2
BEGIN     TRANSFER TO DO(TIMELG (TLX+2))
DO(0)     EXECUTE JPMAJ.
          TRANSFER TO BEGIN
DO(1)     EXECUTE MPMIN.
          TRANSFER TO BEGIN
DO(2)     EXECUTE IGP.
          TRANSFER TO BEGIN
DO(4)    PRINT FORMAT P2,TIME.(0),TIMELG(TLX+1)
          PUNCH FORMAT P2,TIME.(0),TIMELG(TLX+1),2
          EXECUTE SEQPGM.
DO(5)     TRANSFER TO DO(4)
          END OF PROGRAM
```

TABLE XXIX

ALERT SUBROUTINE

```
*COMPILE MAD,PUNCH OBJECT                                                ALERT000
          EXTERNAL FUNCTION                                                      505
          ENTRY TO ALERTJ.                                                       506
         NORMAL MODE IS INTEGER
          EQUIVALENCE(INPUT(30),DIG),(INPUT(31),DMP),(INPUT(32)
         1 ,DJC),(INPUT(33),DMA),(INPUT(34),DMC),(INPUT(36),DPBQ)
         2 ,(INPUT(37),DPUR),(INPUT(38),DQ),(INPUT(48),MAEND),(IN
         3 PUT(51),PBQEND),(INPUT(53),QEND),(INPUT(54),CQEND),(
         4 MISC(0),LASTL),(MISC(1),LASTK),(MISC(2),COMPLT),(MISC
         5 (3),CXTIME),(MISC(4),ALPHAI),(MISC(5),TLX),(MISC(6),NEWP),(
         6 MISC(7),NEWL),(MISC(8),NEWR),(MISC(9),NEWJ),(MISC(10),NEWPR
         7 ),(MISC(11),Q),(MISC(12),JC),(MISC(13),QBUMP),(MISC(14),
         8 GAMMAI),(MISC(15),PHII),(MISC(16),PII),(MISC(17),K),
         9 (MISC(18),PUR1) ,(INPUT(47),JCEND)
          EQUIVALENCE (MISC(19),PBQU),(MISC(20),BACKLG),(MISC(21),STA
         1 NBY),(MISC(22),INPROG),(MISC(23),COMP),(MISC(24),ASTRSK)
          PROGRAM COMMON IGPLOG,IMPLOG,JCR,MAT,MCT,MISC,PERBUF,PURVUE
         1 ,QUEUE,SEQQ,TIMELG,INPUT,INFIN,TL,J,L,R,CQMAX,PBQMAX,QMAX,T
         2 IME,PUR,OLDTL,OLDTLX,JX,MP,TIME3,RUNNO,RUNNUM
          DIMENSION IGPLOG(195),IMPLOG(144),JCR(34),MAT(324),
         1 MCT(119),INPUT(74),PERBUF(159),PURVUE(367),QUEUE(249),
         2 SEQQ(111),TIMELG(439),MISC(74),TIME3(9)
          EXECUTE FINDJC.                                                        507
          TALERT=TIMELG(TLX+1)                                                   508
          WHENEVER TIMELG(TL+7).E.23
          TIMELG(TL+1)=TALERT
          OR WHENEVER TIMELG(TL+9).GE.0
          TRANSFER TO S50
          OTHERWISE                                                              512
          TINT=TALERT-INFIN+DIST.(PURVUE((TIMELG(TL+5)-1)*DPUR+1))
          WHENEVER TINT .G.0
          TINT=INFIN
          OTHERWISE
          TINT = TINT + INFIN
          END OF CONDITIONAL
          WHENEVER TINT.G .TIMELG(TL+1), TRANSFER TO S50                         514
          JCR(JC*DJC+2)=TIMELG(TL+6)-1
          JCR(JC*DJC+3)=TIMELG(TL+1)-TINT                                        516
          TIMELG(TL+1)=TINT                                                      517
          END OF CONDITIONAL                                                     518
          TIMELG(TL+7)=24                                                        519
          EXECUTE MOVET.                                                         520
          FUNCTION RETURN                                                        521
          ENTRY TO ALERTM.                                                       522
          EXECUTE FINDMC.
          WHENEVER TIMELG(TL+3).NE.0, FUNCTION RETURN                            525
          TALERT=TIMELG(TLX+1)                                                   527
          WHENEVER TIMELG(TL+6) .E.8
          TIMELG(TL+7)=11                                                        530
          TIMELG(TL+1)=TALERT                                                    529
          EXECUTE MOVET.                                                         531
          OTHERWISE
S50       TIMELG(TL+3)=TALERT                                                    534
          END OF CONDITIONAL                                                     533
          FUNCTION RETURN                                                        535
          ENTRY TO ALERTF.
          EXECUTE FINDMC.
          WHENEVER TIMELG(TL+6).E.1.AND.TIMELG(TL+1).E.INFIN
          EXECUTE NOTE.(6,0,7,0,0)
```

TABLE XXIX (Concluded)

```
TIMELG(TL+1)=TIMELG(TLX+1)
EXECUTE MOVET.
END OF CONDITIONAL
FUNCTION RETURN
END OF FUNCTION                                          536

                                                          C.C.     0064
```

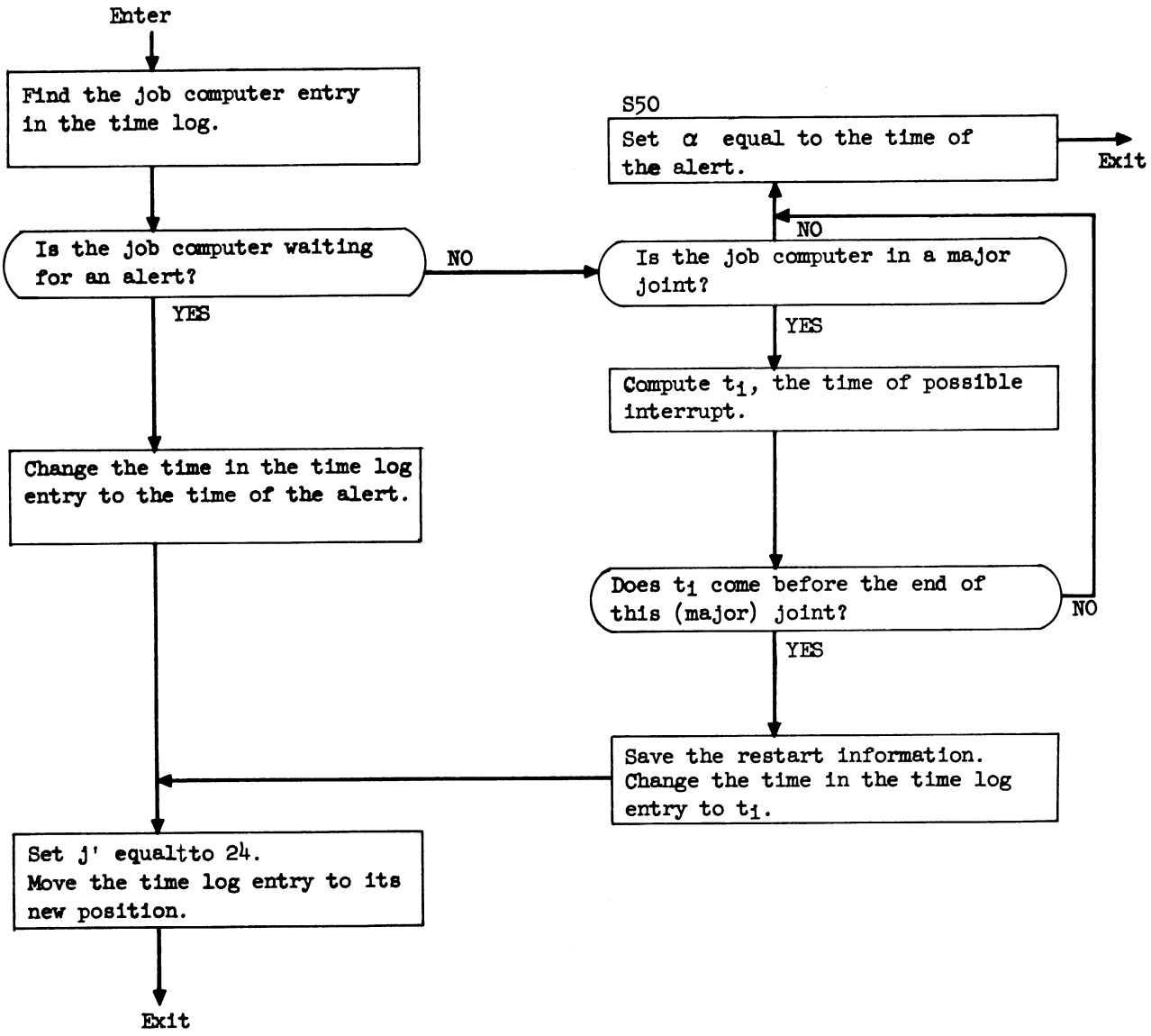

Figure 65. ALERT subroutine—entry to ALERTJ.

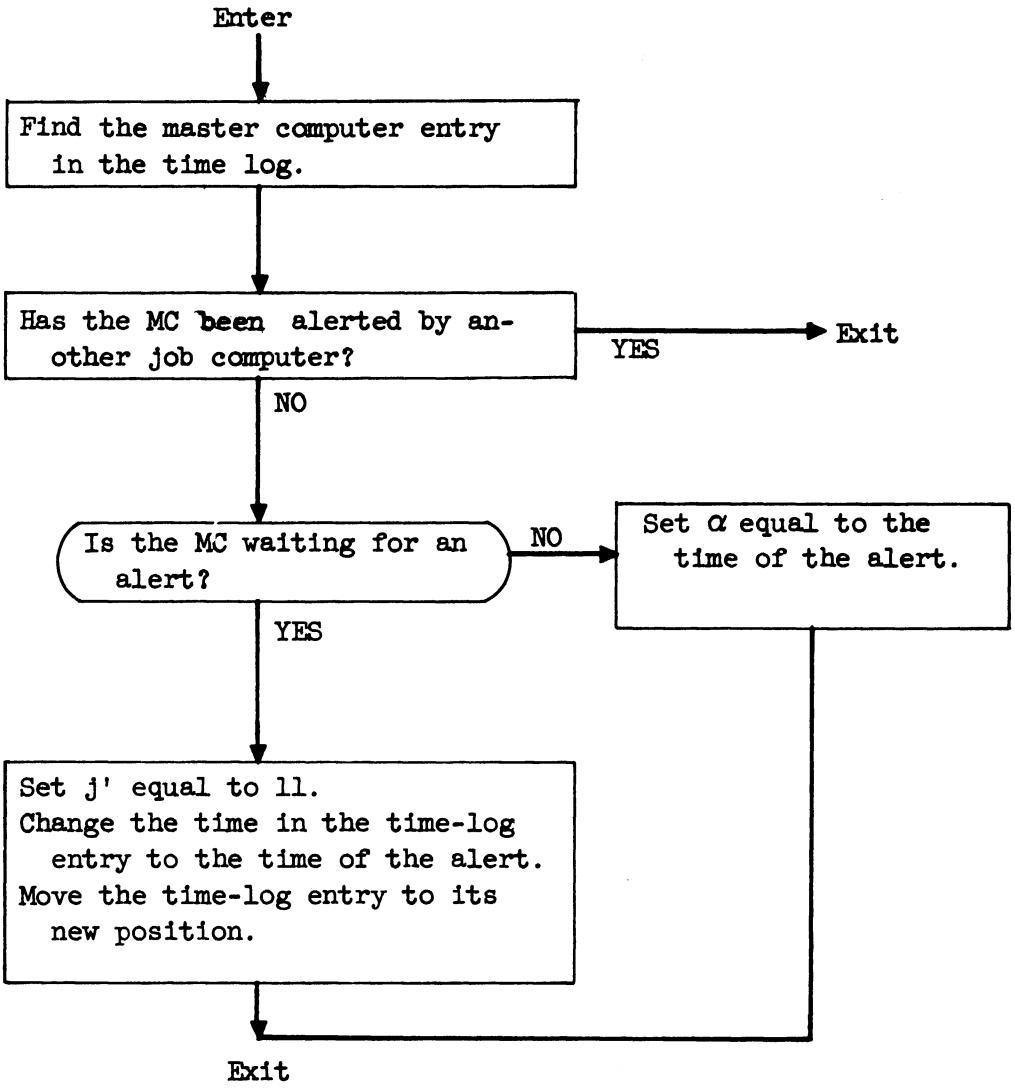

Figure 66. ALERT subroutine—entry to ALERTM.

TABLE XXX

DIST SUBROUTINE

```
* ASSEMBLE ,PUNCH OBJECT                                    DIST 000
        ORG 0
        PGM
        PZE LAST,,1
        PZE XXX-3
        BCD 1DIST
        PZE SWITCH
        ORG 0
        REL
ERROR   BCD 1ERROR
SWITCH  CLA CI
        SUB 1,4
        PAX 0,2
        CLA 0,2
        TNZ *+3
CODE0   CLA -1,2
        TRA 2,4
        TMI ERROR1
        PAX 0,1
        TXL CODE1,1,1
        TXL CODE2,1,2
        TXL CODE3,1,3
ERROR1  TSX ERROR,4
   CI   OCT 300000100000
CODE1   TSX RANGEN,1
        LDQ UNIRAN
        MPY MLLION
        SUB -1,2
        TMI BRANCH
        LDQ UNIRAN
        MPY -16,2
        ADD -17,2
        TRA 2,4
BRANCH  LDQ UNIRAN
        MPY -2,2
        ADD -15,2
        TRA 2,4
MLLION  DEC 1000000B35
CODE2   TSX RANGEN,1
        ARS 3
        LXA NORM,1
        CAS CUTOFF
        TRA *+3
        TRA CHOP
        TRA CHOP
        CAS NORM
        TRA *+4
        TRA *+3
        ALS 1
        TXI *-4,1,512
        SXD SPARE,1
        TSX FXPTLG,1
        ARS 6
        SUB SPARE
        DVH INVLN2
        MPY -2,2
        LLS 7
        CAS -1,2
CHOP    CLA -1,2
```

```
        TRA 2,4
        TRA 2,4
CUTOFF  OCT 1000000
NORM    OCT 100000000000
INVLN2  DEC -1.44269504181
SPARE   PZE 0
CODE3   SXD XXX-1,4
        LXA COUNT,4
        CLS MEAN
        STO XXX-2
        TSX RANGEN,1
        ARS 6
        ADD XXX-2
        TIX *-4,4,1
        DVH SQRT2
        TQP POSMQ
        CLS LIMIT
        TLQ OUTRNG
INRNG   MPY -1,2
        LLS 4
SHIFT   ADD -2,2
        TPL *+2
COUNT   PXD 6,0
        LXD XXX-1,4
        TRA 2,4
LIMIT   OCT 100000000000
MEAN    OCT 060000000000
POSMQ   CLA LIMIT
        TLQ INRNG
OUTRNG  CLM XXX-2
        ORA -1,2
        ALS 2
        TRA SHIFT
RANGEN  CLA UNIRAN
        ALS 9
        ADD UNIRAN
        ADD ONE
        STO UNIRAN
        TRA 1,1
ONE     OCT 1
UNIRAN  EQU81576
FXPTLG  ADD HLFSQ2
        STO XXX-1
        SUB SQRT2
        DVH XXX-1
        STQ XXX-1
        MPR XXX-1
        STO XXX-2
        ARS 2
        STO XXX-3
        ARS 3
        SUB XXX-3
        ADD XXX-2
        ARS 4
        SUB A
        STO XXX-2
        CLS B
        DVH XXX-2
```

TABLE XXX (Concluded)

```
                                                                    11-13-61  3
        STQ XXX-3
        CLA C
        ADD XXX-2
        ADD XXX-3
        STO XXX-2
        LDQ XXX-1
        MPR XXX-2
        SUB HALF
        TRA 1,1
HLFSQ2  DEC .7071067811878B1
SQRT2   DEC 1.414213562374B1
      A DEC .2596385542982
      B DEC .2645155161B4
      C DEC 2.1261061779882
HALF    OCT 40000000000
   XXX  SYN 0
LAST    SYN *
```

C.C. 0134

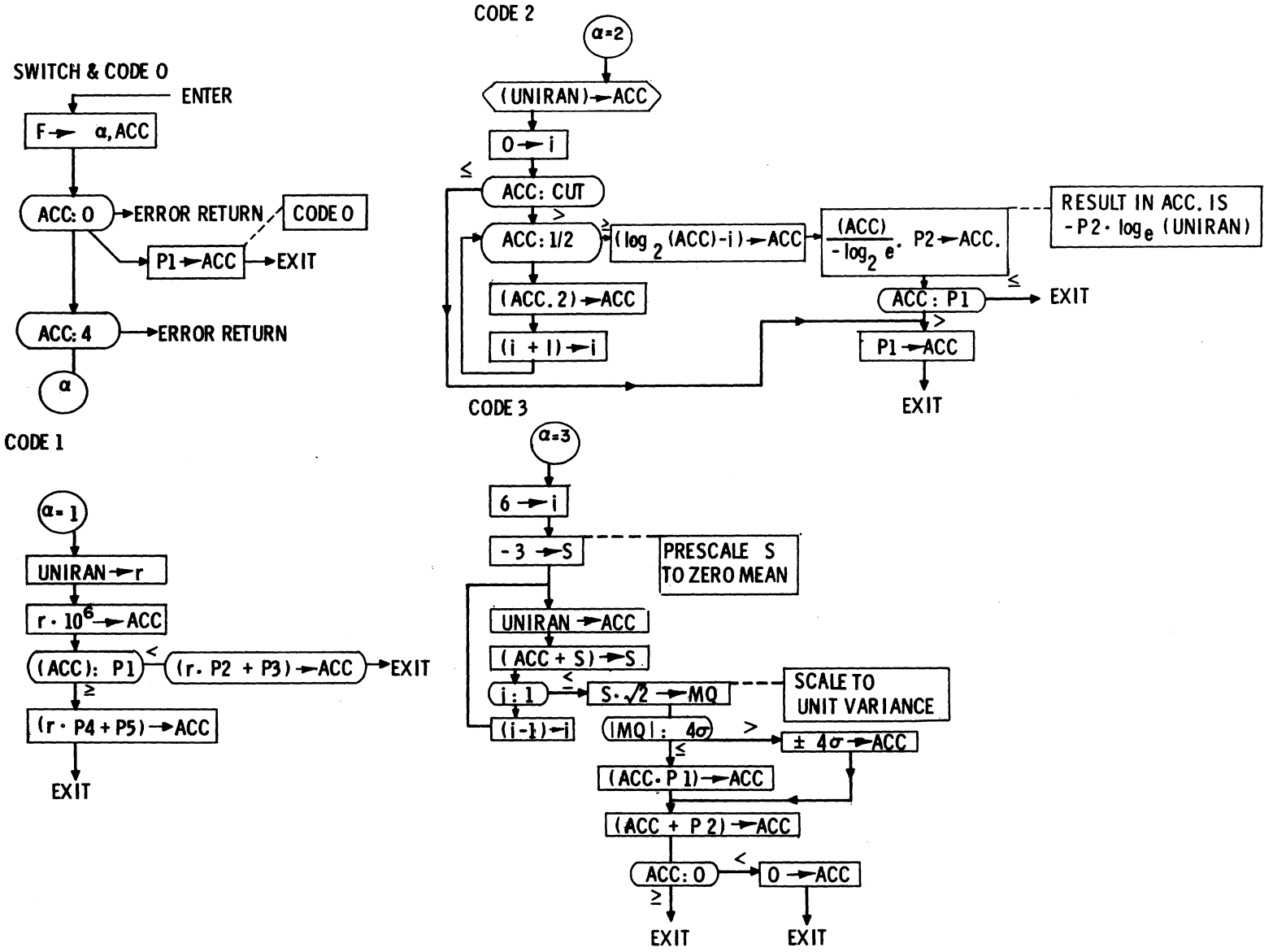

Figure 67. DIST subroutine.

CODE 2 uses the direct method of generation (Ref. 8) to obtain a value of a random variable, t_i, having an exponential density distribution, $p(t) = 1/P2 \cdot e^{-t/P2}$. Namely, the computation

$$t_i = -P2 \log_e r_j$$

is performed, where

$1/P2$ is the mean value of the random variable t per unit time

and

r_j is a value of a random variable, r, uniformly distributed on the unit interval.

The parameter P1 provides a truncation to the exponential distribution. However, the limitations of the program impose a truncation[20] at 11. P2, which therefore takes priority over P1. When either truncation occurs, the accumulator contains P1 upon leaving DIST.

The CODE 3 section makes use of the central limit theorem to generate a value of a random variable, t_i, having a normal density distribution,

$$p(t) = \frac{1}{\sqrt{2\pi}P1} e^{-(P1 \cdot t - P2)^2/2(P1)^2}$$

where

P1 is the standard deviation of t

and

P2 is the mean of t.

Six values of a random variable, uniformly distributed on the unit interval, are summed, shifted, and scaled to represent the value of the normal deviate, t_i, in the range $\pm 4P1+P2$. In case t_i falls outside this range, the value of t_i is taken to the nearest truncation point (either $t_i = P2+4P1$ or $t_i = P2-4P1$). Another truncation is imposed so that negative values of t_i will not occur. For a negative deviate the value of t_i is set to zero.

---

[20]This truncation would occur only .0015% of the time.

The choice of summing six values of a uniform random variable was felt to achieve a suitable balance between the required computing speed and accuracy of generation in these experiments. A chi-squared goodness-of-fit test of the distribution of more than 3000 variates generated in this way showed no statistically significant deviation from the normal distribution.

Two subroutines, RANGEN and FXPTLG, were coded for use by the DIST subroutine. Subroutine RANGEN uses the technique described in Ref. 9 to generate pseudo-random numbers. Successive entries to this subroutine produce successive values in the accumulator and in location UNIRAN according to the formula

$$x_{i+1} = [(2^9+1)x_i + 1] \pmod{2^{35}}$$

Results of the tests for the randomness of this generator are described in Refs. 9 and 10.

Subroutine FXPTLG uses a rational approximation,[21] namely,

$$\log_2 X = Z\left[C + (DZ^2-A) - \frac{B}{DZ^2-A}\right] - 1/2$$

where

$$Z = \frac{2X - \sqrt{2}}{2X + \sqrt{2}}$$

and

$$A = 0.25963855429$$

$$B = 0.26455155161$$

$$C = 2.12610617798$$

$$D = 0.1953125$$

to compute the logarithm to the base 2 of a normalized ($1/2 \leqslant x < 1$) prescaled number. The absolute error in this approximation is less than $\pm 3 \times 10^{-8}$. The value of the computed logarithm is in the accumulator when it leaves this subroutine.

---

[21] Adapted from the SHARE No. 665 write-up of the IBLOG 3 subroutine for the 709.

A.2.7. THE IGP SUBROUTINE

This routine has two entries (Figure 68, Table XXXI). The IGP entry is used whenever an IGP item appears in the time log, signaling a request for a job or a sequence of jobs. The SETIGP entry is used whenever a job is completed. This section of the code checks to see if the completed job was a part of a sequence and, if so, if another event in the sequence can now be started.

A.2.8. THE JPMAJ SUBROUTINE

This subroutine (Figure 69, Table XXXII), starts the next major joint of a job program. The entry JPMAJ is entered from the MAIN2 routine when a time-log entry with s = 2 occurs at the top of the time log. The entry BEGJOB is used by the MPMIN subroutine, joint 19, when starting a new job program.

A.2.9. THE MPMIN SUBROUTINE

This routine (Table XXXIII) is entered by the MAIN2 program whenever there is a time-log entry calling for the performance of a master program joint. One section of the code (Figure 70) is common to all joints (Figure 71-89). It should be noted that a flow chart ending in "return" means a return to the common section of code, whereas "exit" means that control is transferred back to MAIN2. The following joints are too short to warrant a flow chart:

Joint Number	Next Joint
8	8
16	23
17	18
18	19
23	23
27	16
28	(next major joint of the job program)

A.2.10. THE NOTE SUBROUTINE

The function of this routine (Table XXXIV) is to punch the program progress cards and to keep track of the total number of jobs in each stage of processing (Section A.1.4). The three-digit code configurations listed in Table XXIII are input to this routine and indicate which job totals are to be updated. If the job type is given as zero, then these totals are not changed, as this is the situation in which the master program is, so to speak, entering or leaving the system.

TABLE XXXI

IGP SUBROUTINE

```
$COMPILE MAD, PUNCH OBJECT, PRINT OBJECT                                IGP  000
          EXTERNAL FUNCTION                                                    398
          ENTRY TO IGP.                                                        399
          NORMAL MODE IS INTEGER
          EQUIVALENCE(INPUT(30),DIG),(INPUT(31),DMP),(INPUT(32)
         1 ,DJC),(INPUT(33),DMA),(INPUT(34),DMC),(INPUT(36),DPBQ)
         2 ,(INPUT(37),DPUR),(INPUT(38),DQ),(INPUT(48),MAEND),(IN
         3 PUT(51),PBQEND),(INPUT(53),QEND),(INPUT(54),CQEND),(
         4 MISC(0),LASTL),(MISC(1),LASTK),(MISC(2),COMPLT),(MISC
         5 (3),CXTIME),(MISC(4),ALPHAI),(MISC(5),TLX),(MISC(6),NEWP),(
         6 MISC(7),NEWL),(MISC(8),NEWR),(MISC(9),NEWJ),(MISC(10),NEWPR
         7 ),(MISC(11),Q),(MISC(12),JC),(MISC(13),QBUMP),(MISC(14),
         8 GAMMAI),(MISC(15),PHII),(MISC(16),PII),(MISC(17),K),
         9 (MISC(18),PUR1) ,(INPUT(47),JCEND)
          EQUIVALENCE (MISC(19),PBQU),(MISC(20),BACKLG),(MISC(21),STA
         1 NBY),(MISC(22),INPROG),(MISC(23),COMP),(MISC(24),ASTRSK)
          PROGRAM COMMON IGPLOG,IMPLOG,JCR,MAT,MCT,MISC,PERBUF,PURVUE
         1 ,QUEUE,SEQQ,TIMELG,INPUT,INFIN,TL,J,L,R,CQMAX,PBQMAX,QMAX,T
         2 IME,PUR,OLDTL,OLDTLX,JX,MP,TIME3,RUNNO,RUNNUM
          DIMENSION IGPLOG(195),IMPLOG(144),JCR(34),MAT(324),
         1 MCT(119),INPUT(74),PERBUF(159),PURVUE(367),QUEUE(249),
         2 SEQQ(111),TIMELG(439),MISC(74),TIME3(9)
          R = TIMELG (TLX + 5)                                                 400
          WHENEVER R.LE.19                                                     401
          ALPHAI = 0
          PUR=(R-1)*DPUR+4                                                     406
          PERBUF(PBQEND+1)=0                                                   403
S6        PERBUF(PBQEND)=R                                                     402
          EXECUTE NOTE.(0,0,1,LASTK,PERBUF(PBQEND))
          EXECUTE ALERTF.
          TIME = TIMELG(TLX+1)-INFIN + DIST.(PURVUE(PUR))
          WHENEVER TIME .G.0
          TIMELG(TLX+1)=INFIN
          OTHERWISE
          TIMELG(TLX+1)=TIME + INFIN
         END OF CONDITIONAL
          EXECUTE MOVE1.                                                       408
          TRANSFER TO BETA
          END OF CONDITIONAL
          ALPHAI = 1
S7        THROUGH S7,FOR IG=0,DIG,IGPLOG(IG+1).E.R.AND.IGPLOG(IG).E.-1
          FIG=IG                                                               412
          N = IGPLOG(FIG + 5)
          TIME=TIMELG(TLX+1)                                                   414
          E = TIMELG(TLX+3)
          WHENEVER E .E. 1
S8        TOME = TIME -INFIN+DIST.(IGPLOG(IG+2))
          L=LASTL                                                              419
          LASTL=LASTL+1                                                        418
          SEQQ(CQEND) = -L
          CQ = CQEND
          CQEND = CQEND + 1
          FIGEND = FIG + (N+1)*DIG
          THROUGH S9, FOR IG=FIG+DIG+DIG, DIG, IG.E.FIGEND
          SEQQ(CQEND)=IGPLOG(IG)                                               424
S9        CQEND=CQEND+1                                                        425
          WHENEVER CQEND .G.CQMAX,EXECUTE ERROR.
          OTHERWISE                                                            427
S10       L=TIMELG(TLX+4)                                                      428
```

TABLE XXXI (Continued)

```
                EXECUTE ELIM1.                                          429  11-13-61  2
S11             THROUGH S11,FOR CQ=0,1,SEQQ(CQ).E.-L
                SEQQ(CQ-1 +E)=SEQQ(CQ-1+E)-1
                WHENEVER SEQQ(CQ-1+E).NE.0, TRANSFER TO ALPHA(ALPHAI)
                END OF CONDITIONAL
                CQ1 = CQ
S1              CQ1 = CQ1 +1
                WHENEVER CQ1.NE.CQEND
                WHENEVER SEQQ(CQ1).E.0,TRANSFER TO S1
                WHENEVER SEQQ(CQ1).G.0,TRANSFER TO S2
                THROUGH HERE, FOR CQ=CQ,1,CQEND .E.CQ+N
HERE            SEQQ(CQ)=SEQQ(CQ+N)
                END OF CONDITIONAL                                      435
                CQEND = CQ
S2              IG = FIG +1+E*DIG
                WHENEVER IGPLOG(IG).E.0                                 437
                PERBUF(PBQEND)=IGPLOG(IG+1)                             438
                PERBUF(PBQEND+1)=L                                      439
                PERBUF(PBQEND+2)=R                                      440
                PERBUF(PBQEND +3)=E
                EXECUTE NOTE.(0,0,1,LASTK,PERBUF(PBQEND))
                EXECUTE ALERTF.
BETA            PERBUF(PBQEND+4)=LASTK
                LASTK = LASTK+1
                PBQEND=PBQEND+DPBQ                                      442
                 WHENEVER PBQEND .G.PBQMAX,EXECUTE ERROR.
                OTHERWISE                                               444
S14             TUME = TIME - INFIN + DIST.(IGPLOG(IG+1))
                 WHENEVER TUME .G.0
                 TUME = INFIN
                 OTHERWISE
                 TUME = TUME + INFIN
                END OF CONDITIONAL
                 IG1=IG+DIG-5
S162             WHENEVER IGPLOG(IG+4).NE.0
S15              EXECUTE FINDAV.                                        452
                 TIMELG(TL+1) = TUME
                 TIMELG(TL+2)=2
                 TIMELG(TL+3)=IGPLOG(IG+4)                              455
                 TIMELG(TL+4)=L                                         456
                 TIMELG(TL+5)=R                                         457
                 EXECUTE PUTT.                                          458
                 END OF CONDITIONAL                                     434
                 IG=IG+1                                                460
                 WHENEVER IG.NE.IG1,TRANSFER TO S162
                 END OF CONDITIONAL                                     410
                 TRANSFER TO ALPHA(ALPHAI)
                 ENTRY TO SETIGP.                                       465
                 WHENEVER L.E.0,FUNCTION RETURN
S16              THROUGH S16,FOR FIG = 0,DIG,IGPLOG(FIG+1).E.R.AND.IGPLOG(FIG
               1 ).E.-1
                 TIME = TIMELG(TLX+1)
                 ALPHAI =2
                 IG2 = J * DIG + FIG + 5
                 FIGEND = IG2 + DIG - 5
                 N = IGPLOG(FIG+5)
S163             E = IGPLOG(IG2)
                 IG2 = IG2 + 1
```

TABLE XXXI (Concluded)


```
          WHENEVER E.NE.0,TRANSFER TO S11
ALPHA(2)  WHENEVER IG2.NE.FIGEND,TRANSFER TO S163
ALPHA(1)  WHENEVER E.E.1
          WHENEVER TOME .G.0
          TIMELG(TLX+1)=INFIN
          OTHERWISE
          TIMELG(TLX+1) = TOME + INFIN
          END OF CONDITIONAL
          EXECUTE MOVE1.                                    417
          END OF CONDITIONAL
ALPHA(0)  FUNCTION RETURN
          END OF FUNCTION                                   474
```

C.C. 0129

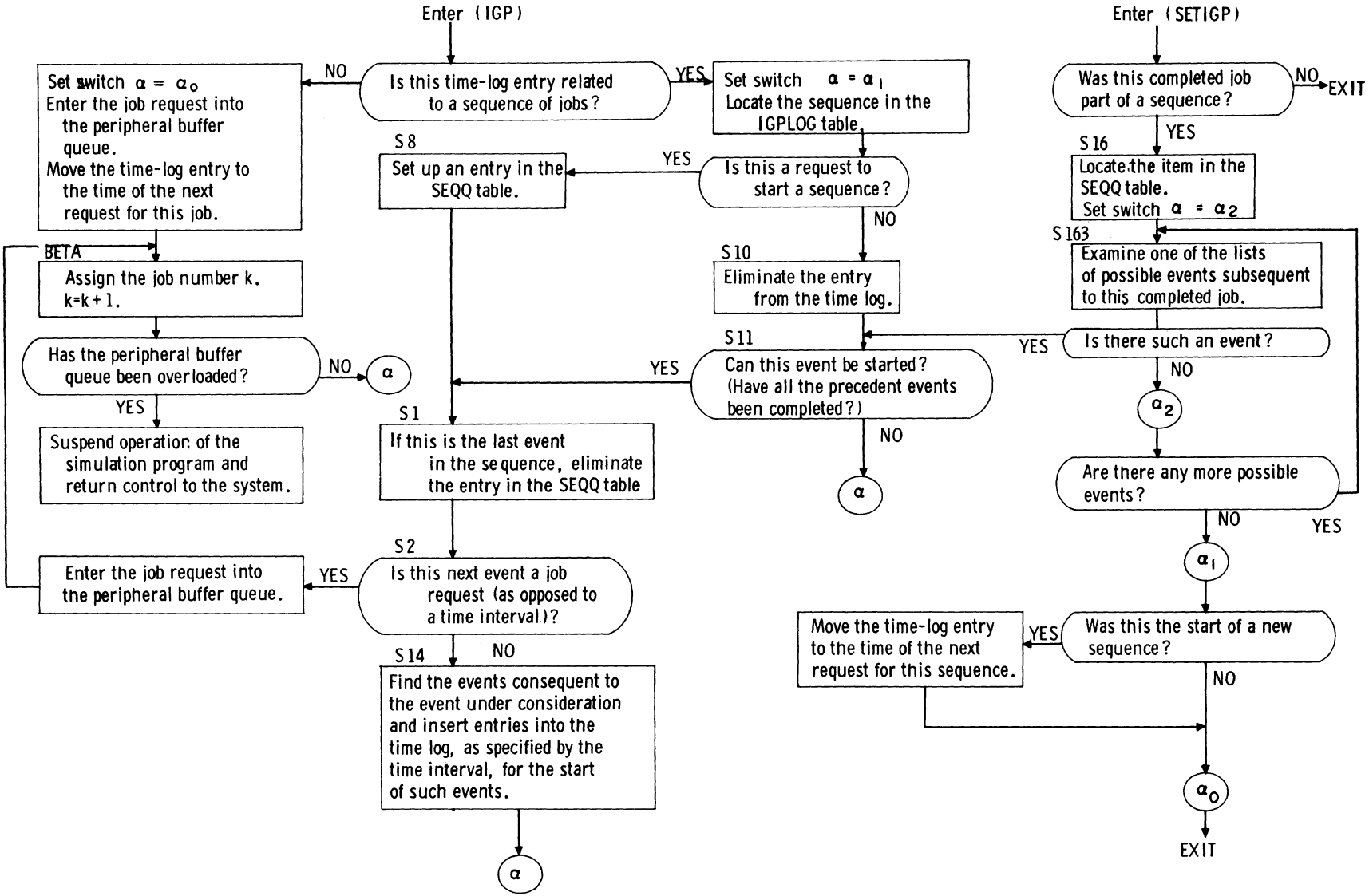

Figure 68. IGP subroutine.

TABLE XXXII

JPMAJ SUBROUTINE

```
*COMPILE MAD,PUNCH OBJECT                                                    JPMAJ000
          EXTERNAL FUNCTION                                                          475
          ENTRY TO JPMAJ.                                                            476
         NORMAL MODE IS INTEGER
          EQUIVALENCE(INPUT(30),DIG),(INPUT(31),DMP),(INPUT(32)
        1 ,DJC),(INPUT(33),DMA),(INPUT(34),DMC),(INPUT(36),DPBQ)
        2 ,(INPUT(37),DPUR),(INPUT(38),DQ),(INPUT(48),MAEND),(IN
        3 PUT(51),PBQEND),(INPUT(53),QEND),(INPUT(54),CQEND),(
        4 MISC(0),LASTL),(MISC(1),LASTK),(MISC(2),COMPLT),(MISC
        5 (3),CXTIME),(MISC(4),ALPHAI),(MISC(5),TLX),(MISC(6),NEWP),(
        6 MISC(7),NEWL),(MISC(8),NEWR),(MISC(9),NEWJ),(MISC(10),NEWPR
        7 ),(MISC(11),Q),(MISC(12),JC),(MISC(13),QBUMP),(MISC(14),
        8 GAMMAI),(MISC(15),PHII),(MISC(16),PII),(MISC(17),K),
        9 (MISC(18),PUR1) ,(INPUT(47),JCEND)
          EQUIVALENCE (MISC(19),PBQU),(MISC(20),BACKLG),(MISC(21),STA
        1 NBY),(MISC(22),INPROG),(MISC(23),COMP),(MISC(24),ASTRSK)
          PROGRAM COMMON IGPLOG,IMPLOG,JCR,MAT,MCT,MISC,PERBUF,PURVUE
        1 ,QUEUE,SEQQ,TIMELG,INPUT,INFIN,TL,J,L,R,CQMAX,PBQMAX,QMAX,T
        2 IME,PUR,OLDTL,OLDTLX,JX,MP,TIME3,RUNNO,RUNNUM
          DIMENSION IGPLOG(195),IMPLOG(144),JCR(34),MAT(324),
        1 MCT(119),INPUT(74),PERBUF(159),PURVUE(367),QUEUE(249),
        2 SEQQ(111),TIMELG(439),MISC(74),TIME3(9)
DISCON    Y=TIMELG(TLX+9)                                                            623
          WHENEVER Y.LE.0.OR.Y.E.30,TRANSFER TO S25
S24       THROUGH S24, FOR MA=0,DMA,MAT(MA+1).E.1.AND.MAT(MA+3).E.0.AN               625
        1 D.MAT(MA).E.Y
          MAT(MA+1)=0
S25       TIMELG(TLX+9)=0
S22       TIME=TIMELG(TLX+1)                                                         478
          PUR=(TIMELG(TLX+5)-1)*DPUR+1                                               479
          TOME = TIME - INFIN + DIST.(PURVUE(PUR+6))
          WHENEVER TOME .G.0
          TIMELG(TLX+1)=INFIN
          OTHERWISE
          TIMELG(TLX+1)=TOME + INFIN
         END OF CONDITIONAL
S181      WHENEVER TIMELG(TLX+6).E.TIMELG(TLX+10)                                    498
          TIMELG(TLX+7)=20
          OTHERWISE                                                                  500
          TIMELG(TLX+7)=28
          END OF CONDITIONAL                                                         502
          ENTRY TO BEGJOB.                                                           481
          WHENEVER TIMELG(TLX+3).E.0, TRANSFER TO S21                                481
          WHENEVER TIMELG(TLX+3).G.TIME
S19       TINT = TIMELG(TLX+3)-INFIN+DIST.(PURVUE(PUR))
          OTHERWISE
          TINT=TIME-INFIN+DIST.(PURVUE(PUR))
          END OF CONDITIONAL
          WHENEVER TINT.G.0
          TINT=INFIN
          OTHERWISE
          TINT = TINT + INFIN
         END OF CONDITIONAL
          WHENEVER TINT.G.TIMELG(TLX+1), TRANSFER TO S21                             483
S20       JCJ = TIMELG(TLX+8)
          TIMELG(TLX+7)=24                                                           485
          JCR(JCJ*DJC+2)=TIMELG(TLX+6)
          JCR(JCJ*DJC+3)= TIMELG(TLX+1)- TINT
          TIMELG(TLX+1)=TINT                                                         488
```

TABLE XXXII (Concluded)

```
S21     TIMELG(TLX+2)=1                                  489        11-13-61  2
        TIMELG(TLX+9)=-1                                 490
        TIMELG(TLX+6)=TIMELG(TLX+6)+1                    491
        EXECUTE MOVE1.                                   492
        FUNCTION RETURN
        END OF FUNCTION                                  492

                                                                C.C.    0065
```

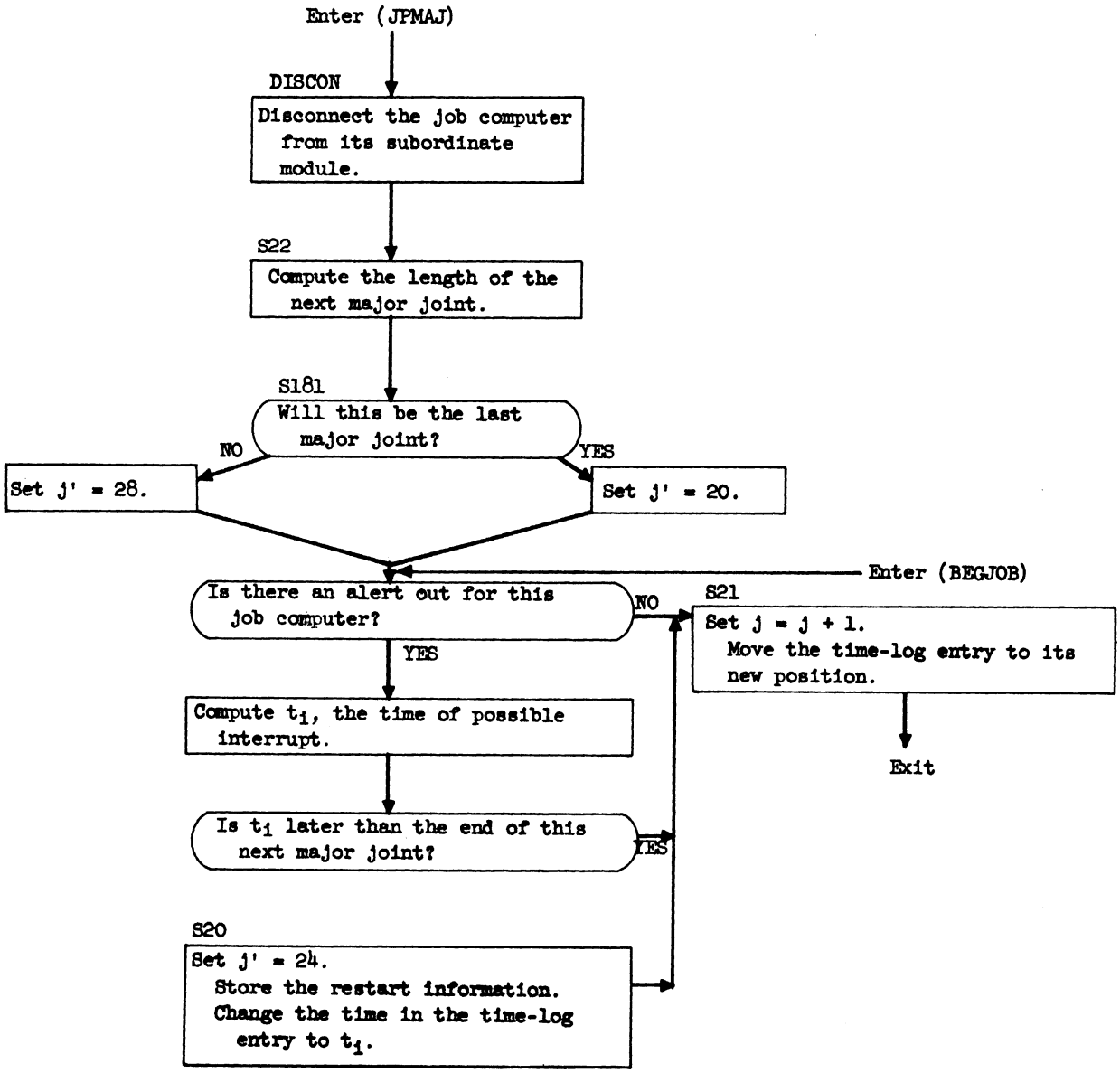

Figure 69. JPMAJ subroutine.

TABLE XXXIII

MPMIN SUBROUTINE

```
*COMPILE MAD,PUNCH OBJECT                                                MPMIN000
          EXTERNAL FUNCTION                                                      125
          ENTRY TO MPMIN.                                                        126
          NORMAL MODE IS INTEGER
          EQUIVALENCE(INPUT(30),DIG),(INPUT(31),DMP),(INPUT(32)
         1 ,DJC),(INPUT(33),DMA),(INPUT(34),DMC),(INPUT(36),DPBQ)
         2 ,(INPUT(37),DPUR),(INPUT(38),DQ),(INPUT(48),MAEND),(IN
         3 PUT(51),PBQEND),(INPUT(53),QEND),(INPUT(54),CQEND),(
         4 MISC(0),LASTL),(MISC(1),LASTK),(MISC(2),COMPLT),(MISC
         5 (3),CXTIME),(MISC(4),ALPHAI),(MISC(5),TLX),(MISC(6),NEWP),(
         6 MISC(7),NEWL),(MISC(8),NEWR),(MISC(9),NEWJ),(MISC(10),NEWPR
         7 ),(MISC(11),Q),(MISC(12),JC),(MISC(13),QBUMP),(MISC(14),
         8 GAMMAI),(MISC(15),PHII),(MISC(16),PII),(MISC(17),K),
         9 (MISC(18),PUR1) ,(INPUT(47),JCEND)
          EQUIVALENCE (MISC(19),PBQU),(MISC(20),BACKLG),(MISC(21),STA
         1 NBY),(MISC(22),INPROG),(MISC(23),COMP),(MISC(24),ASTRSK)
          PROGRAM COMMON IGPLOG,IMPLOG,JCR,MAT,MCT,MISC,PERBUF,PURVUE
         1 ,QUEUE,SEQQ,TIMELG,INPUT,INFIN,TL,J,L,R,CQMAX,PBQMAX,QMAX,T
         2 IME,PUR,OLDTL,OLDTLX,JX,MP,TIME3,RUNNO,RUNNUM
          DIMENSION IGPLOG(195),IMPLOG(144),JCR(34),MAT(324),
         1 MCT(119),INPUT(74),PERBUF(159),PURVUE(367),QUEUE(249),
         2 SEQQ(111),TIMELG(439),MISC(74),TIME3(9)
          DIMENSION QTEMP(9)
          WHENEVER TIMELG(TLX+7).E.0                                             127
          JX=0                                                                   128
          J=TIMELG(TLX+6)                                                        129
          OTHERWISE                                                              130
          JX=1                                                                   131
          J=TIMELG(TLX+7)                                                        132
          END OF CONDITIONAL                                                     133
          WHENEVER TIMELG(TLX+3).LE.TIMELG(TLX+1).AND.IMPLOG((J-1)
         1 *DMP+1).NE.0.AND.TIMELG(TLX+3).NE.0
          JX=1
          J=11                                                                   138
          END OF CONDITIONAL                                                     142
DISCON    Y=TIMELG(TLX+9)
          WHENEVER Y.LE.0.OR.Y.E.30,TRANSFER TO S25
S24       THROUGH S24, FOR MA=0,DMA,MAT(MA+1).E.1.AND.MAT(MA+3).E.0.AN           625
         1 D.MAT(MA).E.Y                                                         625
          MAT(MA+1)=0                                                            626
S25       TIMELG(TLX+9)=0
          TIME = TIMELG(TLX+1)-INFIN+CXTIME
          WHENEVER TIME .G.0
          TIMELG(TLX+1)=INFIN
          OTHERWISE
          TIMELG(TLX+1)=TIME + INFIN
         END OF CONDITIONAL
S26       MP=(J-1)*DMP                                                           146
          Y=IMPLOG(MP)                                                           147
          WHENEVER Y.E.0.OR.Y.E.30,TRANSFER TO S27
          MA=0                                                                   151
S281      WHENEVER MAT(MA+1).NE.0.OR.MAT(MA+3).NE.0.OR.MAT(MA).NE.Y              152
          MA=MA+DMA                                                              153
          WHENEVER MA.NE.MAEND, TRANSFER TO S281                                 154
S29       TL=TIMELG(TLX)
S291      WHENEVER TIMELG(TL+9).NE.Y
          TL=TIMELG(TL)                                                          157
          TRANSFER TO S291                                                       158
          END OF CONDITIONAL                                                     159
```

TABLE XXXIII (Continued)

```
                TIMELG(TLX+1)=TIMELG(TL+1)                                  160   11-13-61  2
                EXECUTE MOVE1.                                              161
                FUNCTION RETURN                                             162
                END OF CONDITIONAL                                          163
S292            MAT(MA+1)=1                                                 164
S30             THROUGH S30, FOR MC=0,DMC,MCT(MC).E.Y                       165
                TIME = TIMELG(TLX+1)-INFIN + DIST.(MCT(MC+1))
                WHENEVER TIME .G.0
                TIMELG(TLX+1)=INFIN
                OTHERWISE
                TIMELG(TLX+1)=TIME + INFIN
                END OF CONDITIONAL
S27             TIMELG(TLX+9)=Y
                TRANSFER TO JOINT(J)                                        169
SETJ(0)         TIMELG(TLX+6)=NEXTJ                                         170
                TRANSFER TO NEXT                                            171
SETJ(1)         TIMELG(TLX+7)=NEXTJ                                         172
                TRANSFER TO NEXT                                            173
RESETJ          TIMELG(TLX+7)=0                                             174
                TRANSFER TO SETJ(0)                                         175
REUSEJ          TIMELG(TLX+7)=0                                             176
NEXT            TIME = TIMELG(TLX+1)-INFIN + DIST.(IMPLOG(MP+2))
                WHENEVER TIME .G.0
                TIMELG(TLX+1)=INFIN
                OTHERWISE
                TIMELG(TLX+1)=TIME + INFIN
                END OF CONDITIONAL
                EXECUTE MOVE1.                                              178
S221            FUNCTION RETURN                                             179
JOINT(1)        WHENEVER PBQEND.NE.0                                        180
                ALPHAI=1                                                    181
                NEWP=PERBUF(0)                                              182
                NEWL=PERBUF(1)                                              183
                NEWR=PERBUF(2)                                              184
                NEWJ=PERBUF(3)                                              185
                K = PERBUF(4)
                EXECUTE NOTE.(1,0,2,K,NEWP)
                PBQEND=PBQEND-DPBQ                                          186
                THROUGH S31, FOR PBQ=0,1,PBQ.E.PBQEND                       187
S31             PERBUF(PBQ)=PERBUF(PBQ+DPBQ)                                188
                NEXTJ=2                                                     189
                OTHERWISE                                                   190
                NEXTJ=4                                                     191
                END OF CONDITIONAL                                          192
                TRANSFER TO SETJ(JX)                                        193
JOINT(2)        NEWPR=PURVUE((NEWP-1)*DPUR)                                 194
                NEXTJ=3                                                     195
                TRANSFER TO SETJ(JX)                                        196
JOINT(3)        Q=QEND-DQ
                QEND=QEND+DQ                                                199
                WHENEVER QEND.G.QMAX                                        200
                EXECUTE ERROR.                                              201
                END OF CONDITIONAL                                          202
S34             WHENEVER Q.L.0, TRANSFER TO S35                             203
S32             WHENEVER QUEUE(Q).G.NEWPR                                   204
                THROUGH S33, FOR Q1=0,1,Q1.E.DQ                             205
S33             QUEUE(Q+DQ+Q1)=QUEUE(Q+Q1)                                  206
                Q=Q-DQ                                                      207
```

TABLE XXXIII (Continued)

```
                TRANSFE* TO S34                                          208  11-13-61  3
                END OF CONDITIONAL                                       209
S35             Q=Q+DQ                                                   210
                QUEUE(Q)=NEWPR                                           211
                QUEUE(Q+1)=K
                QUEUE(Q+2)=NEWP                                          213
                QUEUE(Q+3)=0                                             214
                QUEUE(Q+4)=0                                             215
                QUEUE(Q+5)=NEWL                                          216
                QUEUE(Q+6)=NEWR                                          217
                QUEUE(Q+7)=63                                            218
                QUEUE(Q+8)=NEWJ                                          219
                PUR=(NEWP-1)*DPUR+10
                QUEUE(Q+9)=DIST.(PURVUE(PUR))                            221
                EXECUTE NOTE.(2,0,3,K,NEWP)
                NEXTJ =1
                TRANSFER TO SETJ(JX)
ALPHA(2)        NEXTJ = 29
                TRANSFER TO SETJ(JX)                                     223
JOINT(4)        WHENEVER COMPLT.E.0, TRANSFER TO ALPHA(ALPHAI)            224
                ALPHAI=1                                                 225
                Q=7                                                      226
S36             WHENEVER QUEUE(Q).NE.1023                                227
                Q=Q+DQ                                                   228
                TRANSFER TO S36                                          229
                END OF CONDITIONAL                                       230
                COMPLT=COMPLT-1                                          231
                K=QUEUE(Q-6)                                             232
                MA=3                                                     233
S37             WHENEVER MAT(MA).E.K,MAT(MA)=-1                          234
                MA=MA+DMA                                                235
                WHENEVER MA.L.MAEND, TRANSFER TO S37                     236
                NEXTJ=5                                                  237
                TRANSFER TO SETJ(JX)                                     238
JOINT(5)        L=QUEUE(Q-2)                                             239
                R=QUEUE(Q-1)                                             240
                J=QUEUE(Q+1)                                             241
               EXECUTE SETIGP.                                          242
                EXECUTE NOTE.(8,0,0,K,QUEUE(Q-5))
                NEXTJ=6                                                  243
                TRANSFER TO SETJ(JX)                                     244
JOINT(6)        QEND=QEND-DQ                                             245
                THROUGH S38, FOR Q=Q-7,1,Q.E.QEND                        246
S38             QUEUE(Q)=QUEUE(Q+DQ)                                     247
                TRANSFER TO JOINT(4)
ALPHA(1)        Q=7                                                      249
                ALPHAI=2                                                 250
S40             WHENEVER Q.G.QEND, TRANSFER TO ALPHA(2)                  251
                WHENEVER QUEUE(Q).NE.63                                  252
GAMMA(1)        Q=Q+DQ                                                   253
                TRANSFER TO S40                                          254
                END OF CONDITIONAL                                       255
                EXECUTE ONEMOD.(YES1,NO1)                                256
YES1            EXECUTE SETONE.                                          257
                EXECUTE NOTE.(3,0,5,QUEUE(Q-6),QUEUE(Q-5))
JOINT(10)       JC=0
S200            WHENEVER JCR(JC).NE.0
                JC=JC+DJC
```

TABLE XXXIII (Continued)


```
            WHENEVER JC.NE.JCEND,TRANSFER TO S200
            QBUMP=QEND+7
S48         QBUMP=QBUMP-DQ
            WHENEVER QBUMP.E.Q,TRANSFER TO GAMMA(1)
            WHENEVER QUEUE(QBUMP).GE. 16,TRANSFER TO S48
            JC=QUEUE(QBUMP)
            QUEUE(Q)=JC+16
            PII=2
            NEXTJ=7
            EXECUTE NOTE.(5,1,6,QUEUE(Q-6),QUEUE(Q-5))
            OTHERWISE
            WHENEVER QUEUE(Q).E.255
            EXECUTE NOTE.(5,0,6,QUEUE(Q-6),QUEUE(Q-5))
            END OF CONDITIONAL
            QUEUE(Q)=JC/DJC+16
            JC=JC+1
            GAMMAI=1
S52         NEXTJ=12
            END OF CONDITIONAL
            TRANSFER TO SETJ(JX)                                        271
NO1         OUT=0                                                       272
            THROUGH NO2, FOR QBUMP=QEND-DQ+7,-DQ,QBUMP.E.Q              273
            JC=QUEUE(QBUMP)                                             274
            WHENEVER JC.E.255.OR.JC.L. 16.AND.OUT.E.0
            EXECUTE TWOMOD.(YES(OUT),NO2)                               276
YES(0)      WHENEVER JC.E.255, TRANSFER TO YES(1)                       278
            QSAVE=QBUMP                                                 278
            OUT=1                                                       279
            END OF CONDITIONAL                                          279
NO2         CONTINUE                                                    280
            WHENEVER OUT .E.0, TRANSFER TO GAMMA(1)                     281
            QBUMP=QSAVE                                                 281
            JC=QUEUE(QBUMP)
S49         PII=1                                                       282
            NEXTJ=7                                                     283
            EXECUTE NOTE.(3,1,4,QUEUE(Q-6),QUEUE(Q-5))
            TRANSFER TO SETJ(JX)                                        284
YES(1)      QUEUE(QBUMP)=63                                             285
            EXECUTE NOTE.(5,0,3,K,QUEUE(QBUMP-5))
S51         ALPHAI=1                                                    286
            EXECUTE UPSET.                                              287
            WHENEVER JC.E.255
            EXECUTE NOTE.(3,1,5,QUEUE(Q-6),QUEUE(Q-5))
            OTHERWISE
            EXECUTE NOTE.(4,1,5,QUEUE(Q-6),QUEUE(Q-5))
            END OF CONDITIONAL
            TRANSFER TO JOINT(10)
JOINT(7)    EXECUTE ALERTJ.                                             289
            QCHECK=QBUMP-7
            QTEMP(0)=QUEUE(QCHECK)-1
            THROUGH S510,FOR COUNT =1,1,COUNT.E.10
S510        QTEMP(COUNT)=QUEUE(QCHECK+COUNT)
S511        QCHECK=QCHECK-DQ
            WHENEVER QUEUE(QCHECK).L.QTEMP(0).OR.(QUEUE(QCHECK).E.QTEMP(
          1 0).AND.QUEUE(QCHECK+1).L.QTEMP(1))
          2 .OR.QCHECK.L.0
            THROUGH S512,FOR COUNT=0,1,COUNT.E.10
S512        QUEUE(QCHECK+DQ+COUNT)=QTEMP(COUNT)
```

TABLE XXXIII (Continued)

```
                QBUMP=QCHECK+DQ+7                                          11-13-61  5
                WHENEVER QBUMP.LE.Q,Q=Q+DQ
                NEXTJ=8                                                    290
                TRANSFER TO SETJ(JX)                                       291
                OTHERWISE
                THROUGH S513,FOR COUNT=0,1,COUNT.E.10
S513            QUEUE(QCHECK+DQ+COUNT)=QUEUE(QCHECK+COUNT)
                TRANSFER TO S511
                END OF CONDITIONAL
JOINT(8)        NEXTJ=8                                                    292
                TRANSFER TO SETJ(JX)                                       293
JOINT(9)        WHENEVER QUEUE(QBUMP).E.1023,TRANSFER TO S51               294
                WHENEVER QUEUE(QBUMP).NE.255,TRANSFER TO NO1
                TRANSFER TO YES(1)                                         295
JOINT(11)       PHII=1                                                     296
                GAMMAI=2                                                   297
GAMMA(2)        THROUGH ERASE(0), FOR JC=1,DJC,JC.G.JCEND                  298
                TRANSFER TO ERASE(JCR(JC))                                 299
ERASE(1)        JCR(JC)=0                                                  300
                TRANSFER TO S52                                            301
ERASE(2)        PHII=2                                                     302
                JCR(JC)=0                                                  303
                JCR(JC-1)=0                                                304
                TRANSFER TO GAMMA(2)                                       305
ERASE(0)        CONTINUE                                                   306
                NEXTJ=15                                                   307
                TRANSFER TO SETJ(JX)                                       308
JOINT(12)       TL=TIMELG(TLX)                                             309
                CXMTIM=TIMELG(TLX+1)                                       310
S53             WHENEVER CXMTIM.G.TIMELG(TL+1)                             311
                TIMELG(TL+1)=CXMTIM                                        312
                TL=TIMELG(TL)                                              313
                TRANSFER TO S53                                            314
                END OF CONDITIONAL                                         315
                NEXTJ=13                                                   316
                TRANSFER TO SETJ(JX)                                       317
JOINT(13)       JC=(JC-1)/DJC                                              318
                EXECUTE ALERTJ.                                            319
                NEXTJ=14                                                   320
                TRANSFER TO SETJ(JX)                                       321
JOINT(14)       JCR(JC*DJC)=1                                              322
                TRANSFER TO GAMMA(GAMMAI)                                  323
JOINT(15)       TIMELG(TLX+3)=0                                            324
                TRANSFER TO PHI(PHII)                                      325
PHI(1)          TRANSFER TO REUSEJ                                         326
PHI(2)          TRANSFER TO PI(PII)                                        327
PI(1)           NEXTJ=9                                                    328
                TRANSFER TO RESETJ                                         329
PI(2)           NEXTJ=10                                                   330
                TRANSFER TO RESETJ                                         331
JOINT(20)       EXECUTE NOTE.(7,0,8,TIMELG(TLX+4),TIMELG(TLX+5))
                TIMELG(TLX+7)=21
                J=21
                TRANSFER TO S26
JOINT(21)       JCJ=TIMELG(TLX+8)
S54             THROUGH S54, FOR QJ=7,DQ,QUEUE(QJ).E.JCJ                   332
                QUEUE(QJ)=1023                                             333
                JCR(JCJ*DJC+4)=1                                           334
                                                                           335
```

TABLE XXXIII (Continued)

```
                COMPLT=COMPLT+1                                          336
                EXECUTE ALERTF.
S55             THROUGH S55, FOR QJ=7,DQ,QJ.G.QEND.OR.QUEUE(QJ).E.255    337
                WHENEVER QJ.G.QEND                                       338
                JCR(JCJ*DJC)=0                                           339
JOINT(16)       NEXTJ=23
S56             OTHERWISE                                                341
                QUEUE(QJ)=JCJ+16                                         342
                JCR(JCJ*DJC+1)=1                                         343
                EXECUTE NOTE.(5,0,6,QUEUE(QJ-6),QUEUE(QJ-5))
S59             NEXTJ=22                                                 344
                END OF CONDITIONAL                                       345
                TRANSFER TO SETJ(1)                                      346
JOINT(22)       EXECUTE ALERTM.                                          347
                TRANSFER TO JOINT(23)                                    348
JOINT(23)       NEXTJ=23                                                 349
                TRANSFER TO SETJ(1)                                      350
JOINT(24)       TIMELG(TLX+3)=0                                          351
                TRANSFER TO RHO(JCR((TIMELG(TLX+8))*DJC+4))
RHO(1)          NEXTJ=17
                TRANSFER TO SETJ(1)                                      354
RHO(2)          NEXTJ=25                                                 355
                WHENEVER TIMELG(TLX+10).E.TIMELG(TLX+6)-
              1 1.AND.JCR(3+(TIMELG(TLX+8))*DJC).E.0
                EXECUTE NOTE.(7,0,8,TIMELG(TLX+4),TIMELG(TLX+5))
                OTHERWISE
                EXECUTE NOTE.(7,0,6,TIMELG(TLX+4),TIMELG(TLX+5))
                END OF CONDITIONAL
                TRANSFER TO SETJ(1)                                      356
JOINT(25)       JCJ=(TIMELG(TLX+8))*DJC                                  357
                JCR(JCJ+4)=1                                             358
                JCR(JCJ+1)=2                                             359
                TLEFTJ=JCR(JCJ+3)                                        360
                FIRSTJ=JCR(JCJ+2)                                        361
                JCJ=JCJ/DJC                                              362
S58             THROUGH S58, FOR QJ=7,DQ,QUEUE(QJ).E.JCJ                 364
                WHENEVER TIMELG(TLX+10).E.FIRSTJ.AND.TLEFTJ.E.0          363
                QUEUE(QJ)=1023                                           365
                COMPLT=COMPLT+1                                          366
                TRANSFER TO S59                                          367
                END OF CONDITIONAL                                       368
                QUEUE(QJ-4)=FIRSTJ
                QUEUE(QJ-3)=TLEFTJ
                QUEUE(QJ)=255
                NEXTJ=26                                                 372
                EXECUTE NOTE.(6,0,5,QUEUE(QJ-6),QUEUE(QJ-5))
                TRANSFER TO SETJ(1)                                      373
JOINT(26)       EXECUTE ALERTM.                                          374
                NEXTJ=27                                                 375
                TRANSFER TO SETJ(1)                                      376
JOINT(27)       NEXTJ=16
                TRANSFER TO SETJ(1)                                      378
JOINT(17)       NEXTJ=18
                TRANSFER TO SETJ(1)                                      380
JOINT(18)       NEXTJ=19
                TRANSFER TO SETJ(1)                                      382
JOINT(19)       JCJ=TIMELG(TLX+8)
S57             THROUGH S57,FOR QJ=7,DQ, QUEUE(QJ).E.JCJ+16              384
```

TABLE XXXIII (Concluded)


```
                QUEUE(QJ)=JCJ                                                    385
                JCJ=JCJ*DJC                                                      386
                TIMELG(TLX+4)=QUEUE(QJ-6)                                        387
                JCR(JCJ+4)=2                                                     388
                TIMELG(TLX+5)=QUEUE(QJ-5)                                        389
                TIMELG(TLX+6)= QUEUE(QJ-4)
                EXECUTE NOTE.(6,0,7,QUEUE(QJ-6),QUEUE(QJ-5))
                PUR=(TIMELG(TLX+5)-1)*DPUR+1                                     393
                WHENEVER QUEUE(QJ-4).NE.0
                TIME = TIMELG(TLX+1)+QUEUE(QJ-3)-INFIN+DIST.(PURVUE(PUR+24))
                END OF CONDITIONAL
                WHENEVER TIME .G.0
                TIMELG(TLX+1)= INFIN
                OTHERWISE
                TIMELG(TLX+1)=TIME + INFIN
                END OF CONDITIONAL
                TIMELG(TLX+10)=QUEUE(QJ+2)                                       392
                WHENEVER TIMELG(TLX+6).E.TIMELG(TLX+10)
                TIMELG(TLX+7)=20
                OTHERWISE
                TIMELG(TLX+7)=28
                END OF CONDITIONAL
                EXECUTE BEGJOB.                                                  394
                TRANSFER TO S221                                                 394
JOINT(28)       TIMELG(TLX+2)=0
                TRANSFER TO REUSEJ                                               396
JOINT(29)       WHENEVER PBQEND .E.0.AND.COMPLT.E.0
                EXECUTE NOTE.(7,0,6,0,0)
                TIMELG(TLX+1)=INFIN
                TIMELG(TLX+6)=1
                EXECUTE MOVE1.
                FUNCTION RETURN
                END OF CONDITIONAL
                J=1
                TIMELG(TLX+6)=1
                TRANSFER TO S26
                END OF FUNCTION                                                  397
```

C.C. 0386

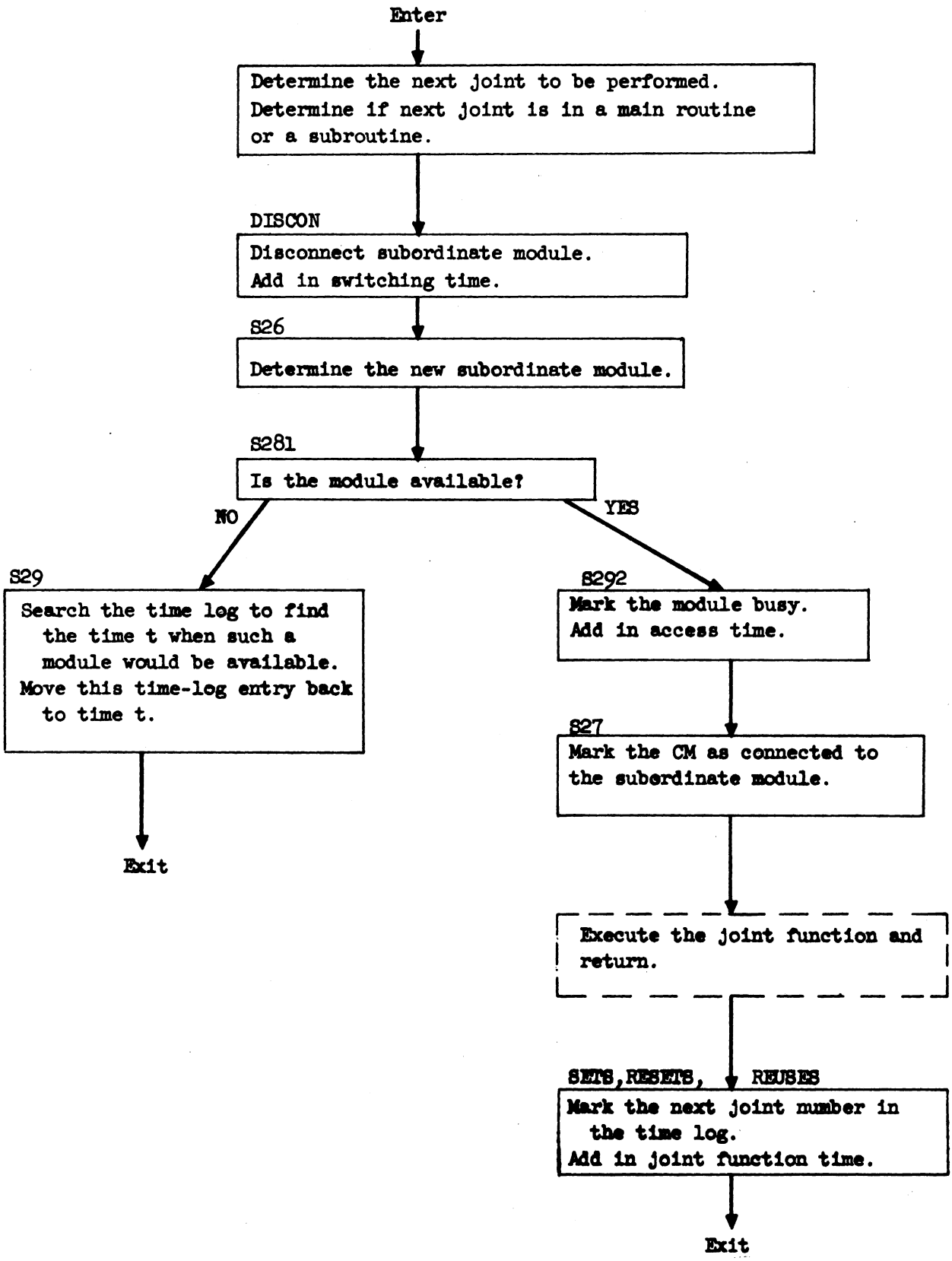

Figure 70. MPMIN subroutines—common section.

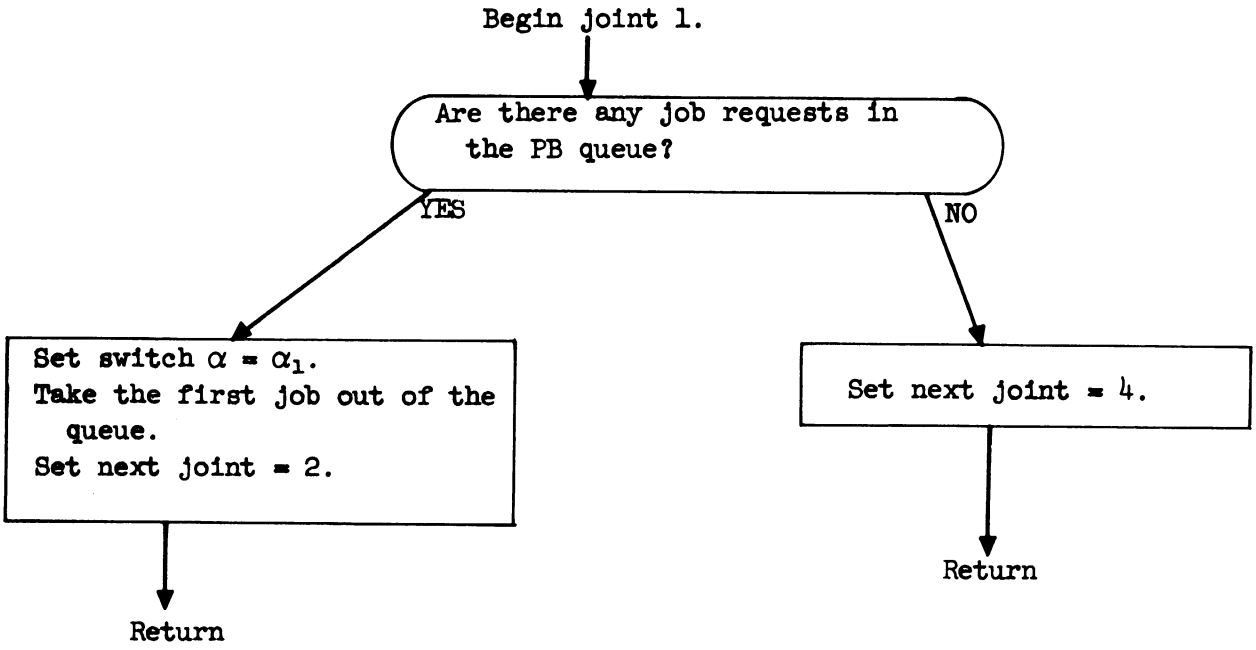

Figure 71. MPMIN subroutine.

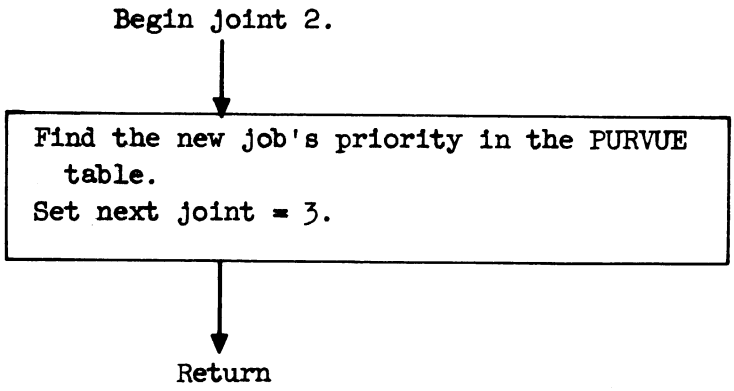

Figure 72. MPMIN subroutine.

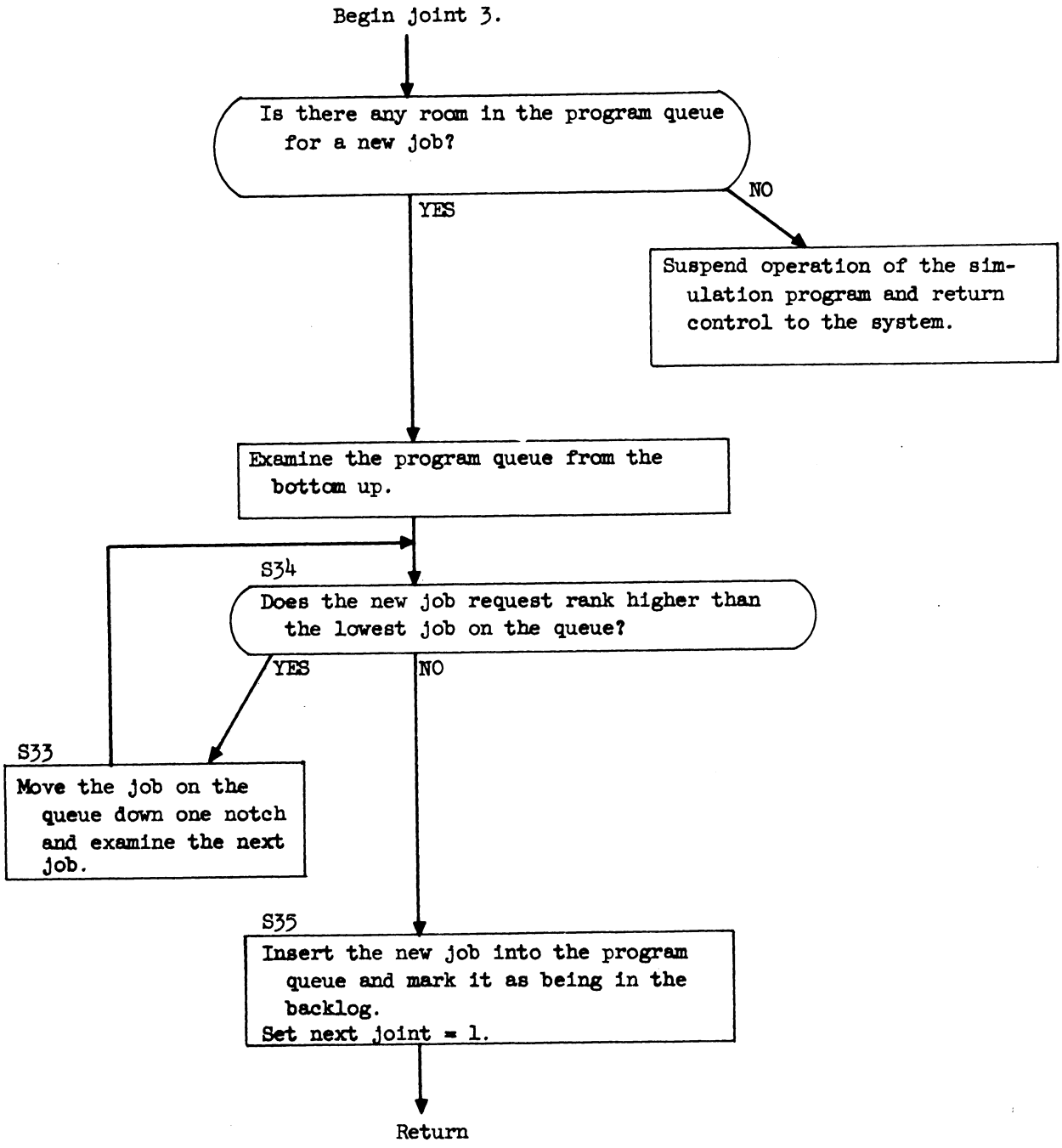

Figure 73. MPMIN subroutine.

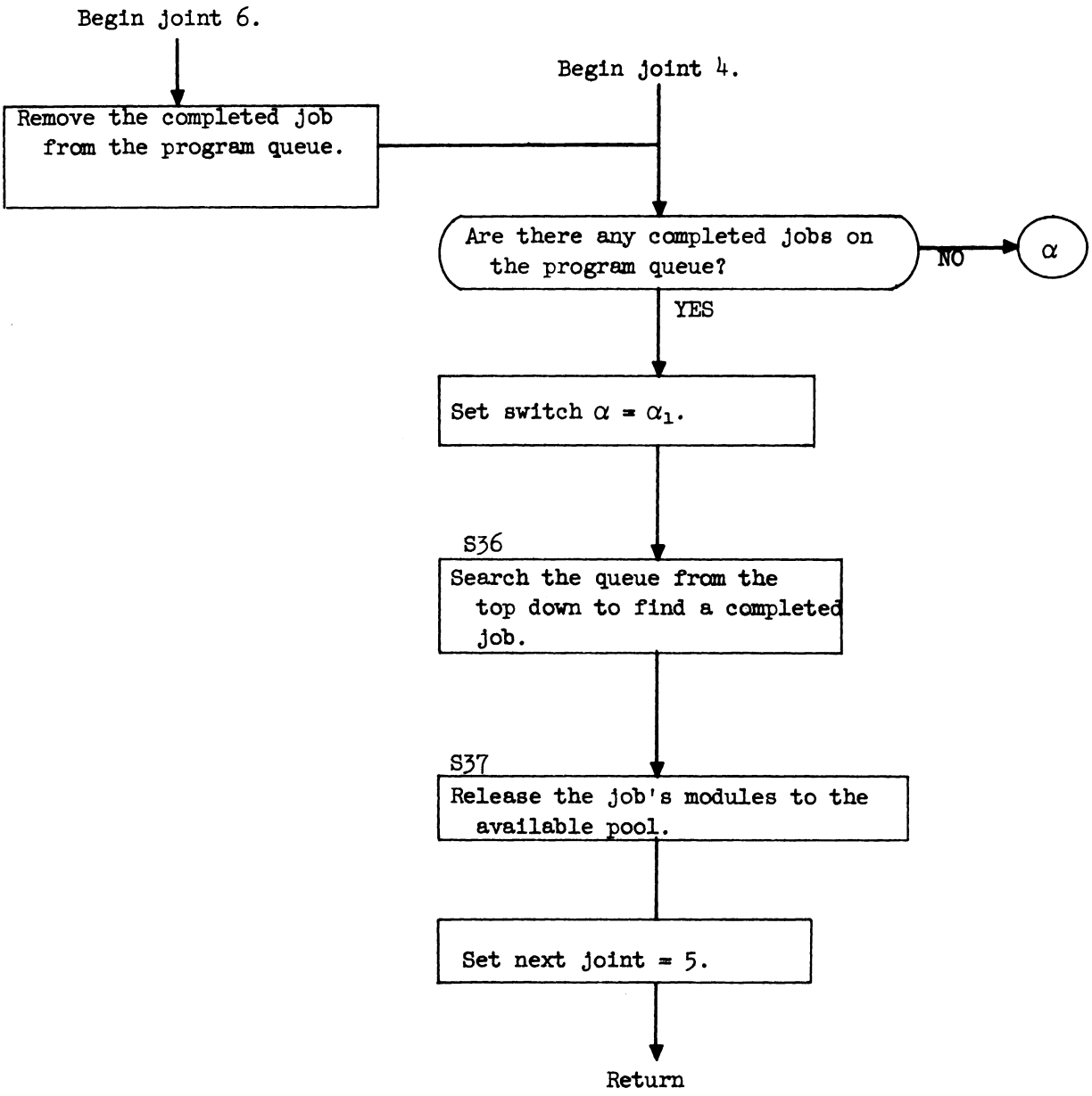

Figure 74. MPMIN subroutine.

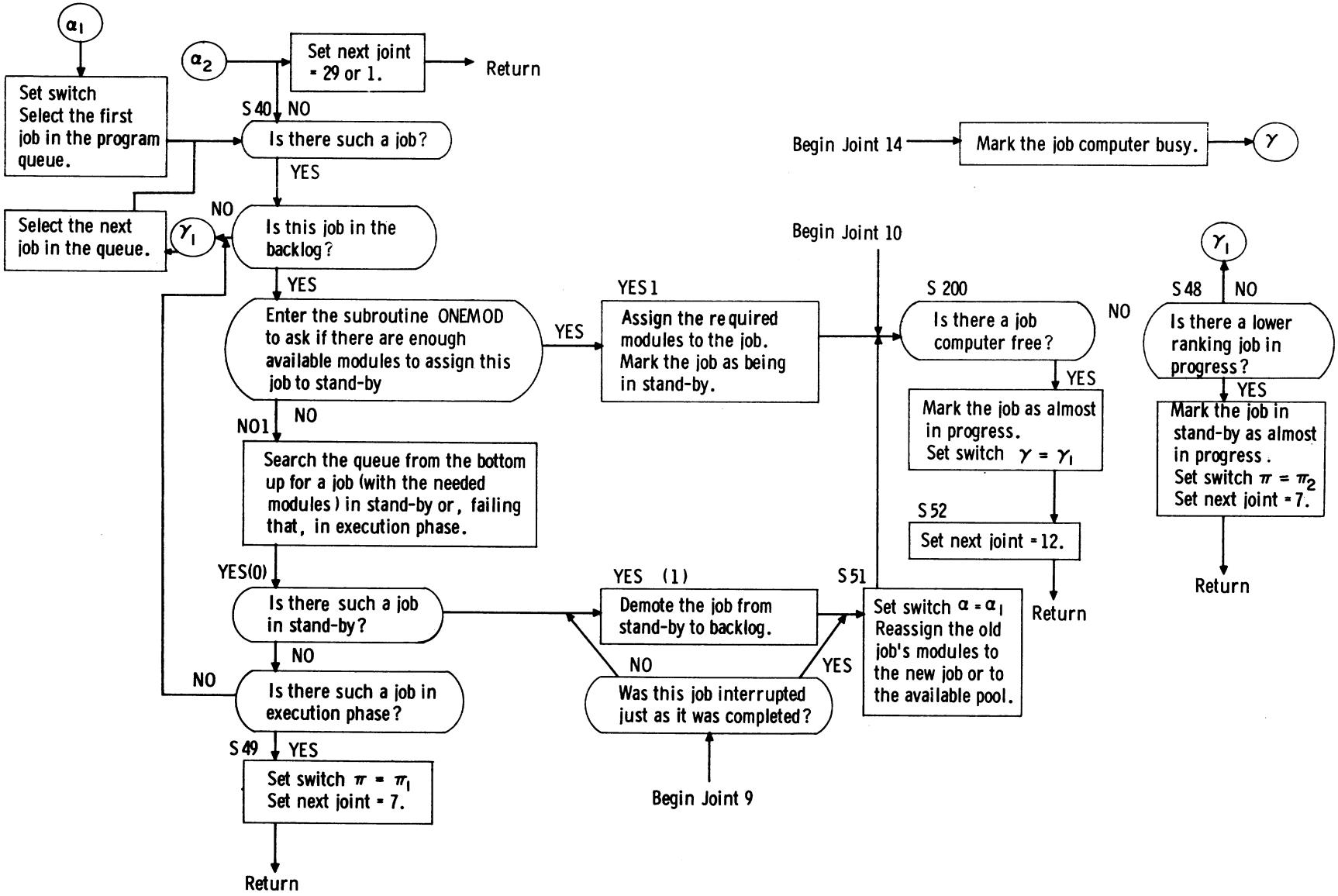

Figure 75. MPMIN subroutine.

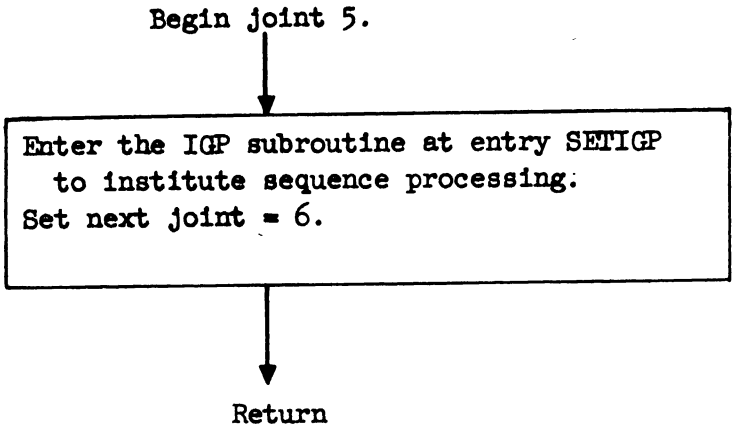

Figure 76. MPMIN subroutine.

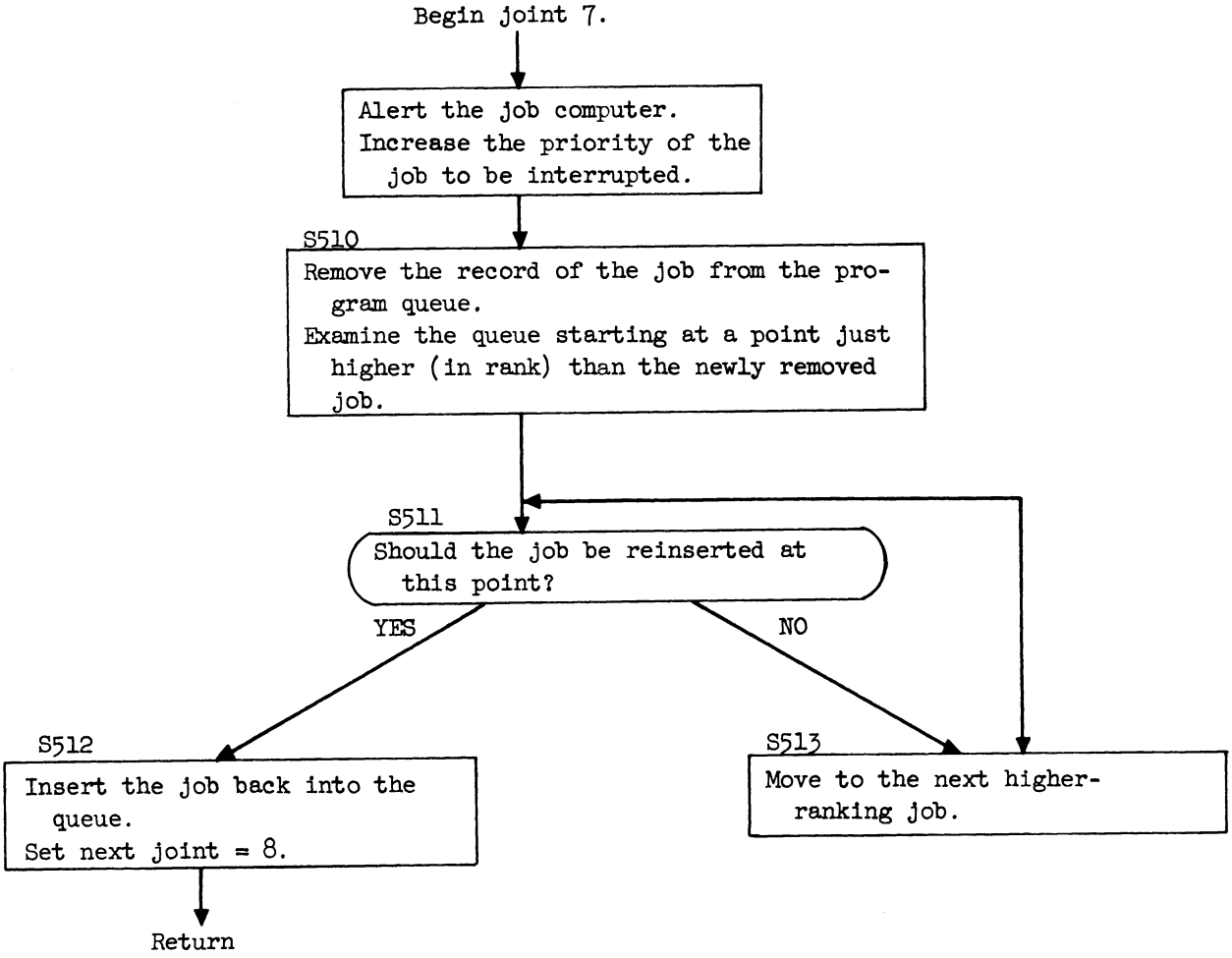

Figure 77. MPMIN subroutine.

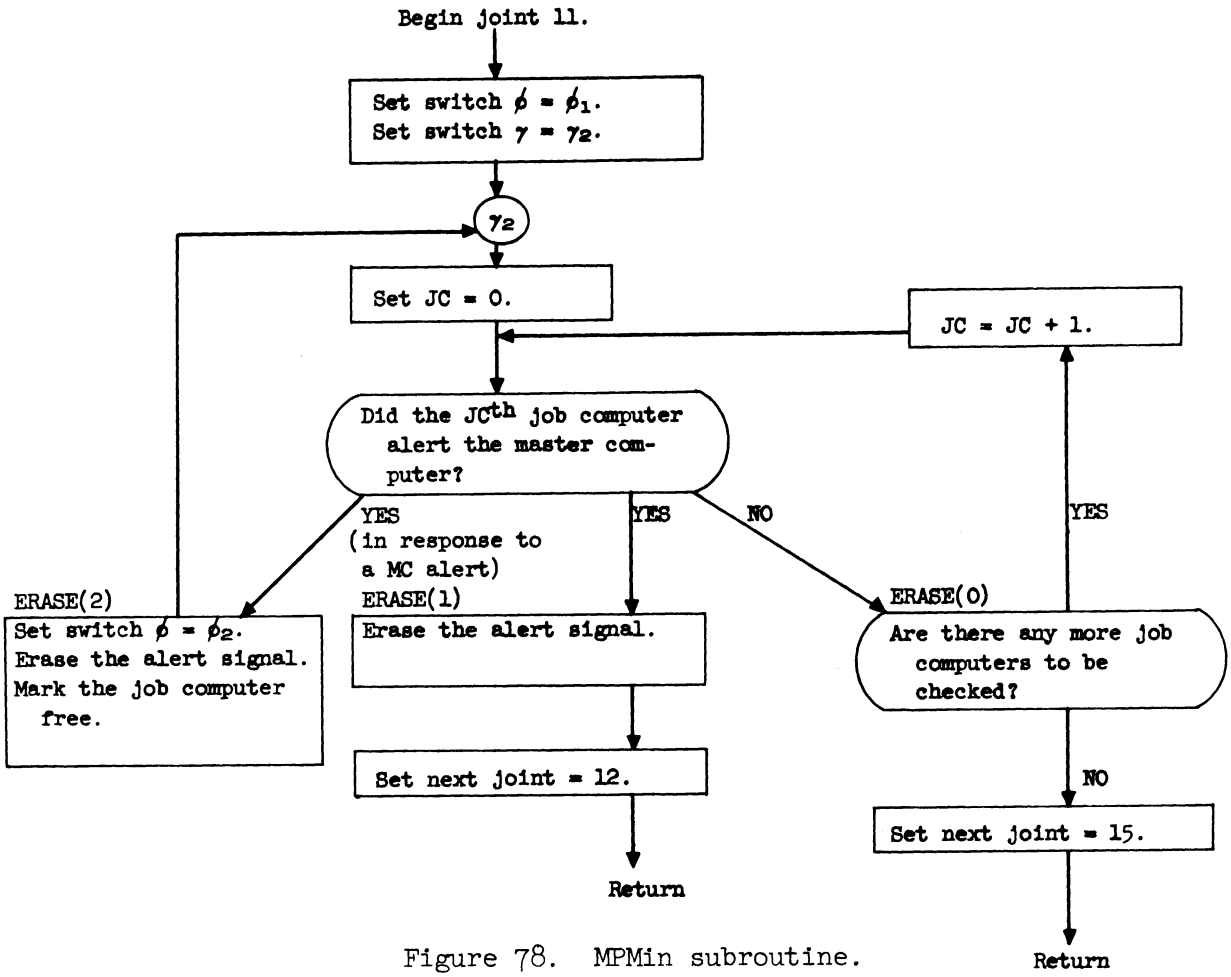

Figure 78. MPMin subroutine.

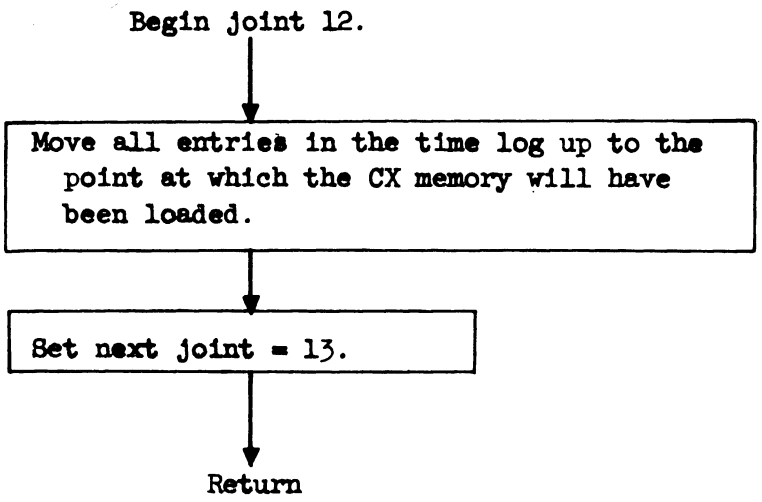

Figure 79. MPMIN subroutine.

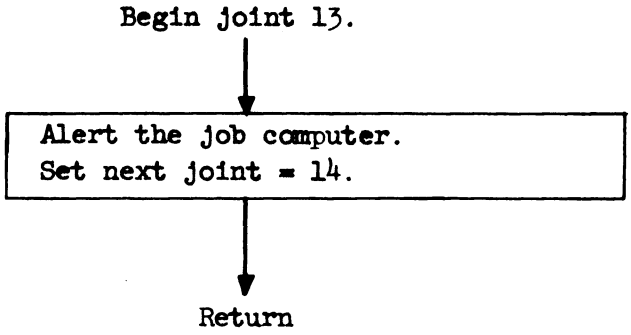

Figure 80. MPMIN subroutine.

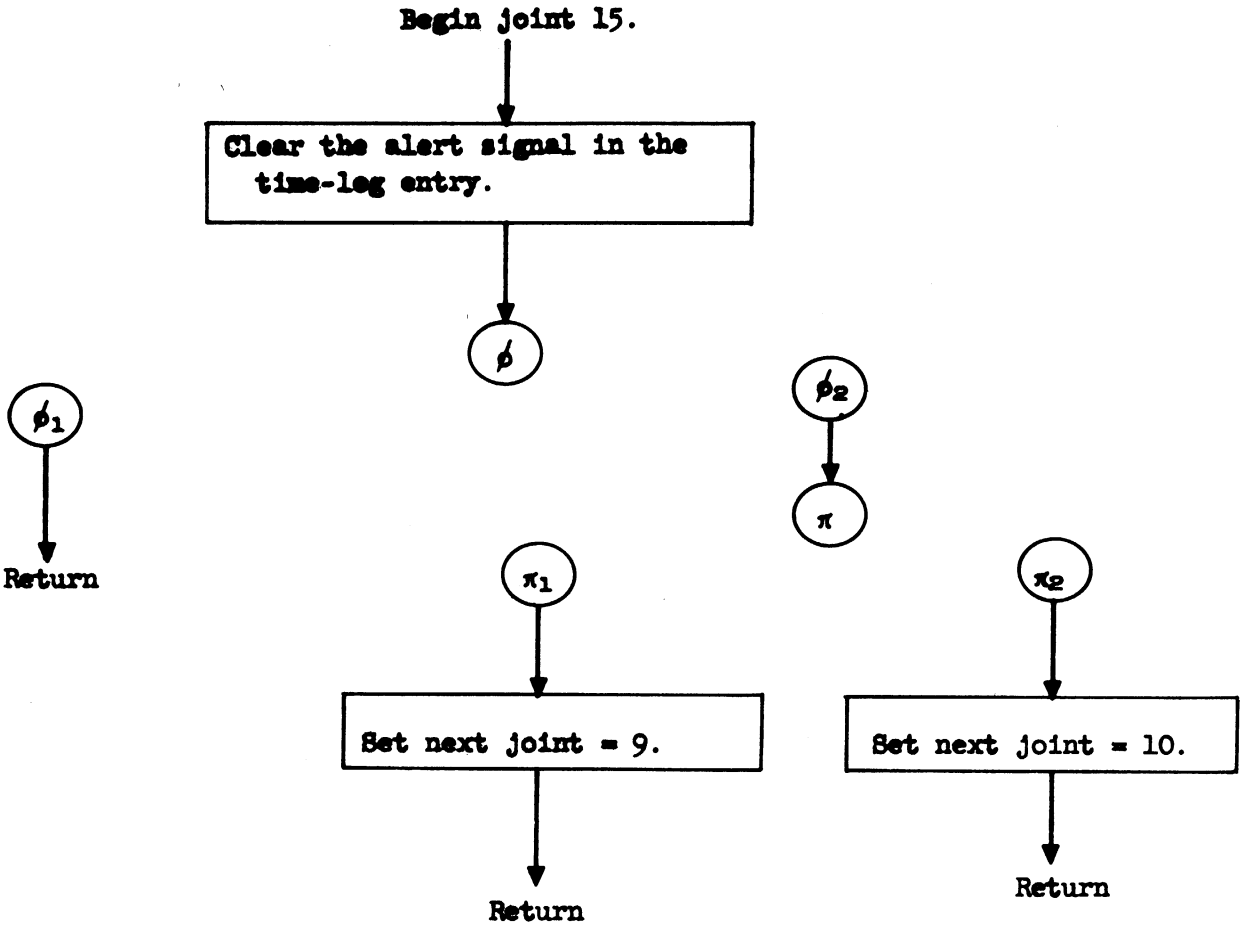

Figure 81. MPMIN subroutine.

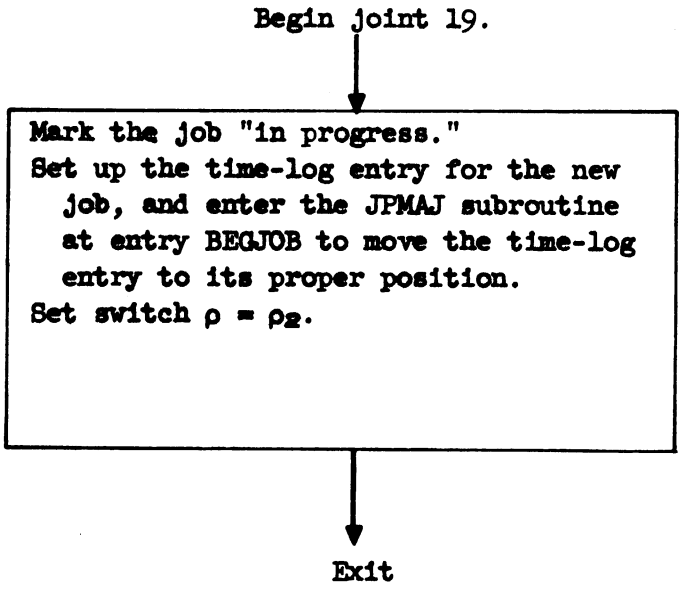

Figure 82. MPMIN subroutine.

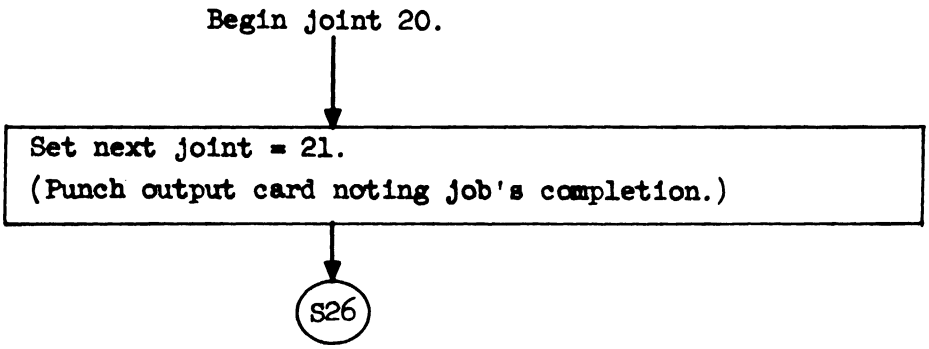

Figure 83. MPMIN subroutine.

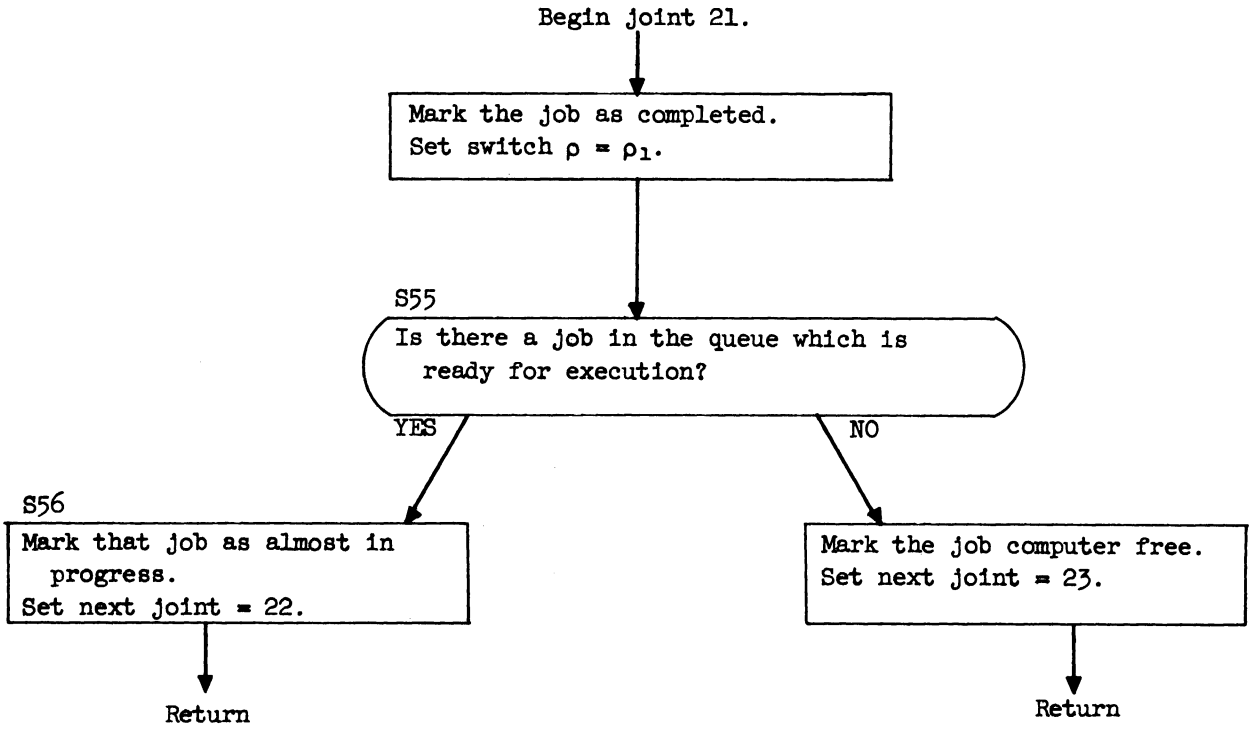

Figure 84. MPMIN subroutine.

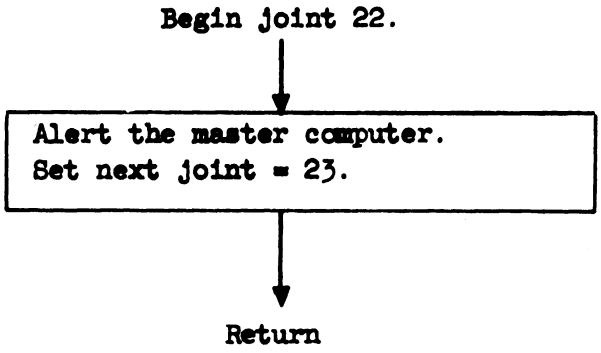

Figure 85. MPMIN subroutine.

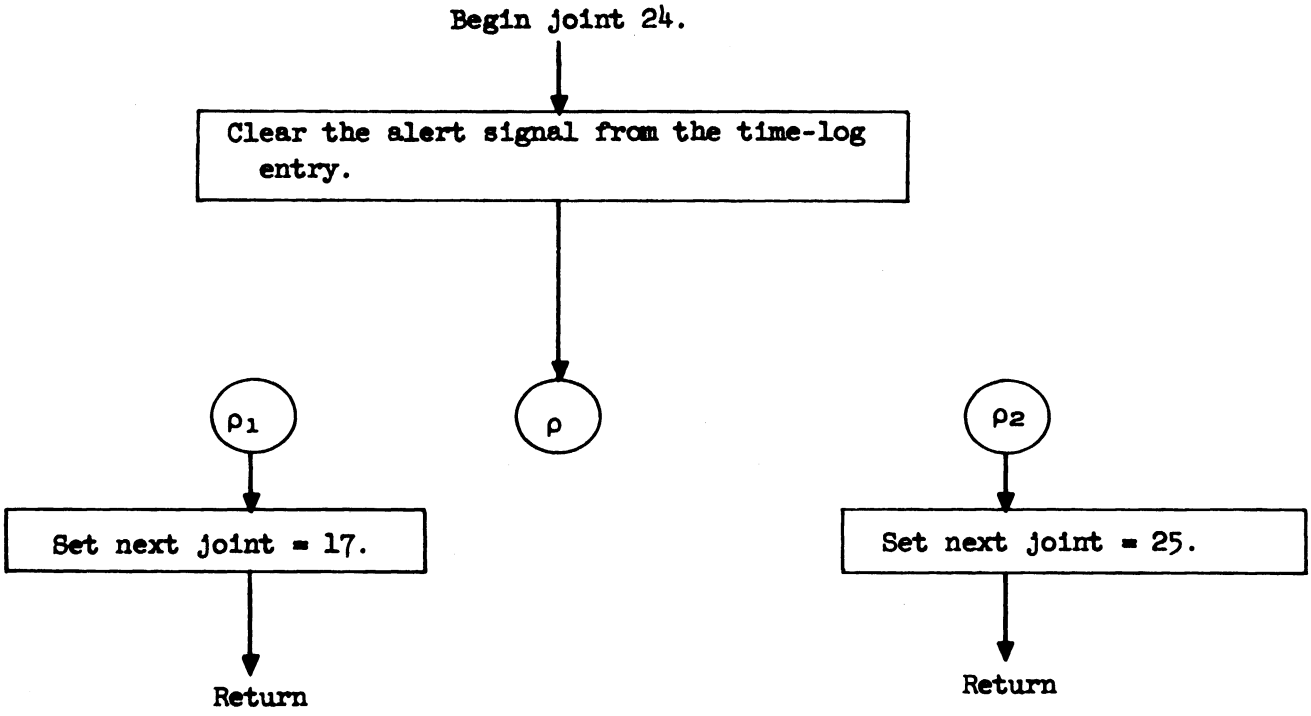

Figure 86. MPMIN subroutine.

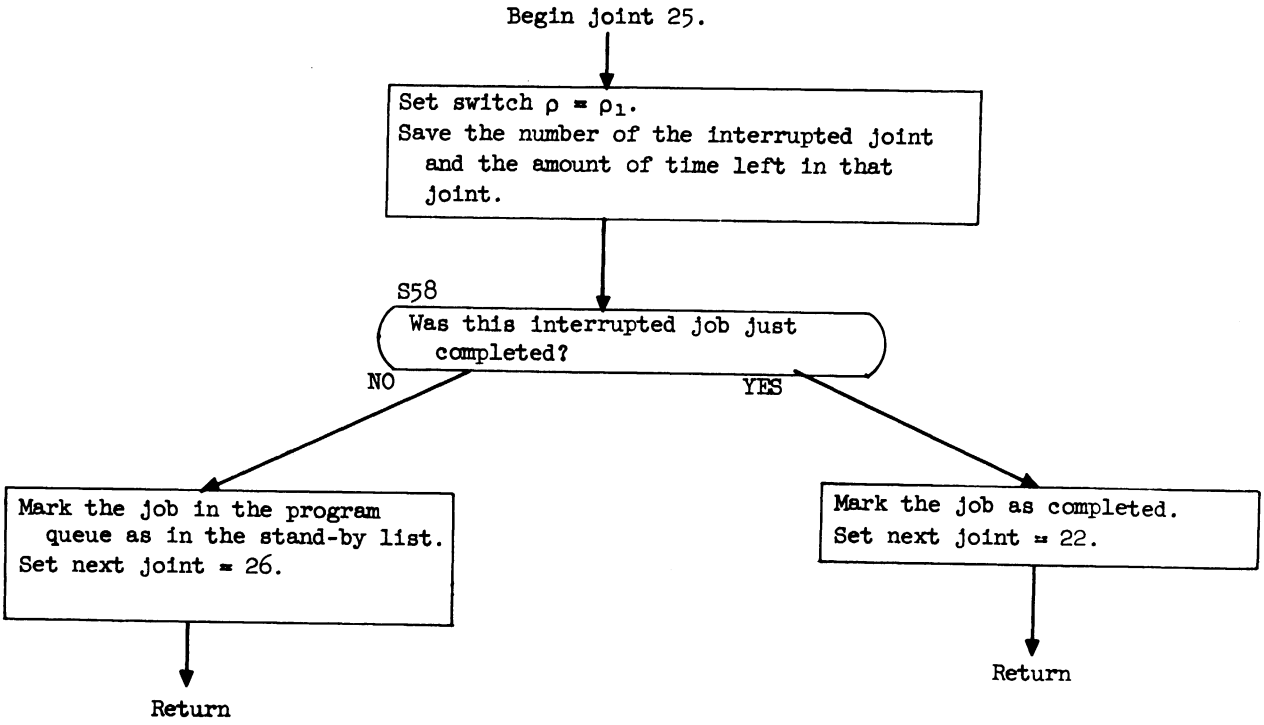

Figure 87. MPMIN subroutine.

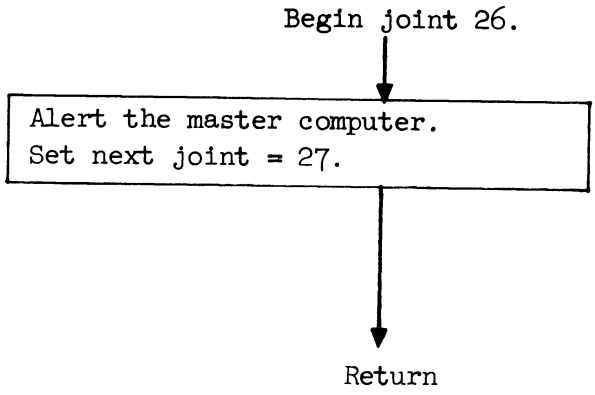

Figure 88. MPMIN subroutine.

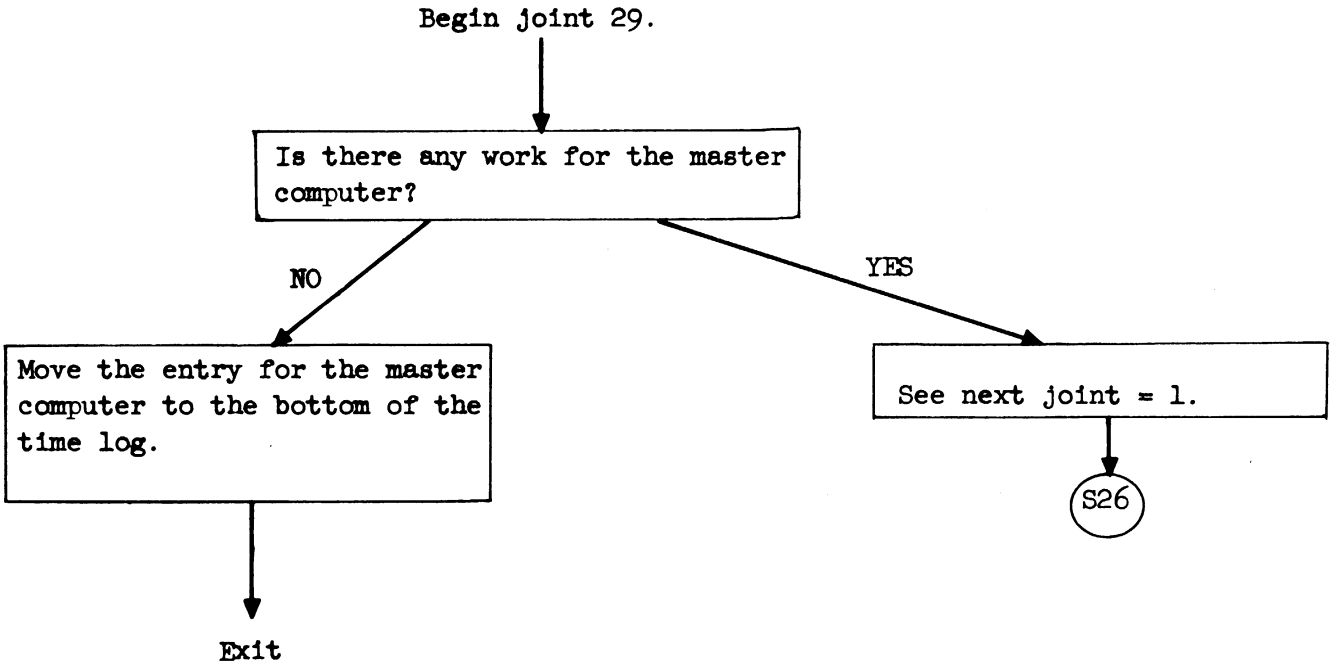

Figure 89. MPMIN subroutine.

TABLE XXXIV

NOTE SUBROUTINE

```
*COMPILE MAD,PUNCH OBJECT                                                NOTE 000
            EXTERNAL FUNCTION(CODEA,CODEB,CODEC,KPRINT,PPRINT)
            ENTRY TO NOTE.
            NORMAL MODE IS INTEGER
            EQUIVALENCE(INPUT(30),DIG),(INPUT(31),DMP),(INPUT(32)
           1 ,DJC),(INPUT(33),DMA),(INPUT(34),DMC),(INPUT(36),DPBQ)
           2 ,(INPUT(37),DPUR),(INPUT(38),DQ),(INPUT(48),MAEND),(IN
           3 PUT(51),PBQEND),(INPUT(53),QEND),(INPUT(54),CQEND),(
           4 MISC(0),LASTL),(MISC(1),LASTK),(MISC(2),COMPLT),(MISC
           5 (3),CXTIME),(MISC(4),ALPHAI),(MISC(5),TLX),(MISC(6),NEWP),(
           6 MISC(7),NEWL),(MISC(8),NEWR),(MISC(9),NEWJ),(MISC(10),NEWPR
           7 ),(MISC(11),Q),(MISC(12),JC),(MISC(13),QBUMP),(MISC(14),
           8 GAMMAI),(MISC(15),PHII),(MISC(16),PII),(MISC(17),K),
           9 (MISC(18),PUR1) ,(INPUT(47),JCEND)
            EQUIVALENCE (MISC(19),PBQU),(MISC(20),BACKLG),(MISC(21),STA
           1 NBY),(MISC(22),INPROG),(MISC(23),COMP),(MISC(24),ASTRSK)
            PROGRAM COMMON IGPLOG,IMPLOG,JCR,MAT,MCT,MISC,PERBUF,PURVUE
           1 ,QUEUE,SEQQ,TIMELG,INPUT,INFIN,TL,J,L,R,CQMAX,PBQMAX,QMAX,T
           2 IME,PUR,OLDTL,OLDTLX,JX,MP,TIME3,RUNNO,RUNNUM
            DIMENSION IGPLOG(195),IMPLOG(144),JCR(34),MAT(324),
           1 MCT(119),INPUT(74),PERBUF(159),PURVUE(367),QUEUE(249),
           2 SEQQ(111),TIMELG(439),MISC(74),TIME3(9)
            WHENEVER PPRINT .E.0, TRANSFER TO DOTOO(0)
           C=CODEC
            TRANSFER TO DO(CODEA)
DO(0)       TRANSFER TO DOTOO(C)
DO(1)       PBQU = PBQU-1
            TRANSFER TO DOTOO(C)
DO(2)       ASTRSK = ASTRSK -1
            TRANSFER TO DOTOO(C)
DO(3)       BACKLG = BACKLG -1
            TRANSFER TO DOTOO(C)
DO(4)       ASTRSK = ASTRSK -1
            TRANSFER TO DOTOO(C)
DO(5)       STANBY = STANBY-1
            TRANSFER TO DOTOO(C)
DO(6)       ASTRSK = ASTRSK -1
            TRANSFER TO DOTOO(C)
DO(7)       INPROG = INPROG-1
            TRANSFER TO DOTOO(C)
DO(8)       COMP = COMP-1
            TRANSFER TO DOTOO(C)
DOTOO(1)    PBQU = PBQU+1
            TRANSFER TO DOTOO(0)
DOTOO(2)    ASTRSK = ASTRSK +1
            TRANSFER TO DOTOO(0)
DOTOO(3)   BACKLG = BACKLG+1
            TRANSFER TO DOTOO(0)
DOTOO(4)    ASTRSK = ASTRSK +1
            TRANSFER TO DOTOO(0)
DOTOO(5)    STANBY = STANBY+1
            TRANSFER TO DOTOO(0)
DOTOO(6)    ASTRSK = ASTRSK +1
            TRANSFER TO DOTOO(0)
DOTOO(7)    INPROG = INPROG+1
            TRANSFER TO DOTOO(0)
DOTOO(8)    COMP = COMP+1
DOTOO(0)    PUNCH FORMAT OUTPUT,RUNNUM,CODEA,CODEB,CODEC,TIMELG(TLX+1),
           1 KPRINT,PPRINT,PBQU,BACKLG,STANBY,INPROG,COMP,ASTRSK
```

TABLE XXXIV (Concluded)

```
 VECTOR VALUES OUTPUT = $I6,I4,I1,I1,I17,I6,I4,S9,6(I4),S7,1                    11-13-61  2
1 H1#$
  FUNCTION RETURN
  END OF FUNCTION

                                                                      C.C.      0063
```

A.2.11. THE ONEMD SUBROUTINE

This section of the program (Table XXXV) is used by the MPMIN subroutine to see if a job program can be assigned to the stand-by list. There are, therefore, two exits to the routine, "no" and "yes."

There are also two entries. Entry ONEMOD is used first. This checks the job's requirements against the list of available modules. Entry TWOMOD is used only after a prior entry to ONEMOD with a negative exit. TWOMOD compares the job's requirements with both the modules available and the modules assigned to a particular lower-ranking job k.

This routine assumes that the modules listed both in the MAT table and in the PURVUE table are in numerical order. In the flow charts in Figure 90, "MA" represents the number of a module in the MAT table, and "PUR," a module in the PURVUE table.

A.2.12. THE SETMD SUBROUTINE

This code (Table XXXVI), Figure 91), is used by the MPMIN subroutine to assign a job program to the stand-by list. Entry SETONE is used when the job's required modules are all obtainable from the list of available modules in the MAT table. Entry UPSET is used when it is necessary to demote a job k from standy-by to backlog, to obtain eough modules for the new job. "MA" and "PUR" are used as in Section A.2.11.

```
*COMPILE MAD,PUNCH OBJECT                                          ONEMD000
          EXTERNAL FUNCTION (YES,NO)                                        577
          ENTRY TO ONEMOD.                                                  578
         NORMAL MODE IS INTEGER
          EQUIVALENCE(INPUT(30),DIG),(INPUT(31),DMP),(INPUT(32)
         1 ,DJC),(INPUT(33),DMA),(INPUT(34),DMC),(INPUT(36),DPBQ)
         2 ,(INPUT(37),DPUR),(INPUT(38),DQ),(INPUT(48),MAEND),(IN
         3 PUT(51),PBQEND),(INPUT(53),QEND),(INPUT(54),CQEND),(
         4 MISC(0),LASTL),(MISC(1),LASTK),(MISC(2),COMPLT),(MISC
         5 (3),CXTIME),(MISC(4),ALPHAI),(MISC(5),TLX),(MISC(6),NEWP),(
         6 MISC(7),NEWL),(MISC(8),NEWR),(MISC(9),NEWJ),(MISC(10),NEWPR
         7 ),(MISC(11),Q),(MISC(12),JC),(MISC(13),QBUMP),(MISC(14),
         8 GAMMAI),(MISC(15),PHII),(MISC(16),PII),(MISC(17),K),
         9 (MISC(18),PUR1) ,(INPUT(47),JCEND)
          EQUIVALENCE (MISC(19),PBQU),(MISC(20),BACKLG),(MISC(21),STA
         1 NBY),(MISC(22),INPROG),(MISC(23),COMP),(MISC(24),ASTRSK)
          PROGRAM COMMON IGPLOG,IMPLOG,JCR,MAT,MCT,MISC,PERBUF,PURVUE
         1 ,QUEUE,SEQQ,TIMELG,INPUT,INFIN,TL,J,L,R,CQMAX,PBQMAX,QMAX,T
         2 IME,PUR,OLDTL,OLDTLX,JX,MP,TIME3,RUNNO,RUNNUM
          DIMENSION IGPLOG(195),IMPLOG(144),JCR(34),MAT(324),
         1 MCT(119),INPUT(74),PERBUF(159),PURVUE(367),QUEUE(249),
         2 SEQQ(111),TIMELG(439),MISC(74),TIME3(9)
          STATEMENT LABEL YES,NO                                            577
          DELTAI=1                                                          579
          PUR=(QUEUE(Q-5)-1)*DPUR+13                                        580
          PUR1=PUR                                                          581
          WHENEVER PURVUE(PUR).E.999, TRANSFER TO YES
          TRANSFER TO S41                                                   582
          ENTRY TO TWOMOD.                                                  583
          DELTAI=2                                                          584
          PUR=PUR1
          K=QUEUE(QBUMP-6)                                                  585
S41       MA=0                                                              586
S42       WHENEVER MAT(MA).G.PURVUE(PUR), TRANSFER TO NO                    587
          WHENEVER MAT(MA).L.PURVUE(PUR)                                    588
DELTA(1)  MA=MA+DMA                                                         589
          WHENEVER MA.NE.MAEND, TRANSFER TO S42                             590
          TRANSFER TO NO                                                    591
          END OF CONDITIONAL                                                592
S43       WHENEVER MAT(MA+3).NE.-1, TRANSFER TO DELTA(DELTAI)               593
S44       WHENEVER MAT(MA+2).NE.0, TRANSFER TO DELTA(1)                     594
          PUR=PUR+1                                                         595
          WHENEVER PURVUE(PUR).E.999, TRANSFER TO YES                       596
          TRANSFER TO DELTA(1)                                              597
DELTA(2)  WHENEVER MAT(MA+3).E.K, TRANSFER TO S44                           598
          TRANSFER TO DELTA(1)                                              599
          END OF FUNCTION                                                   600
```

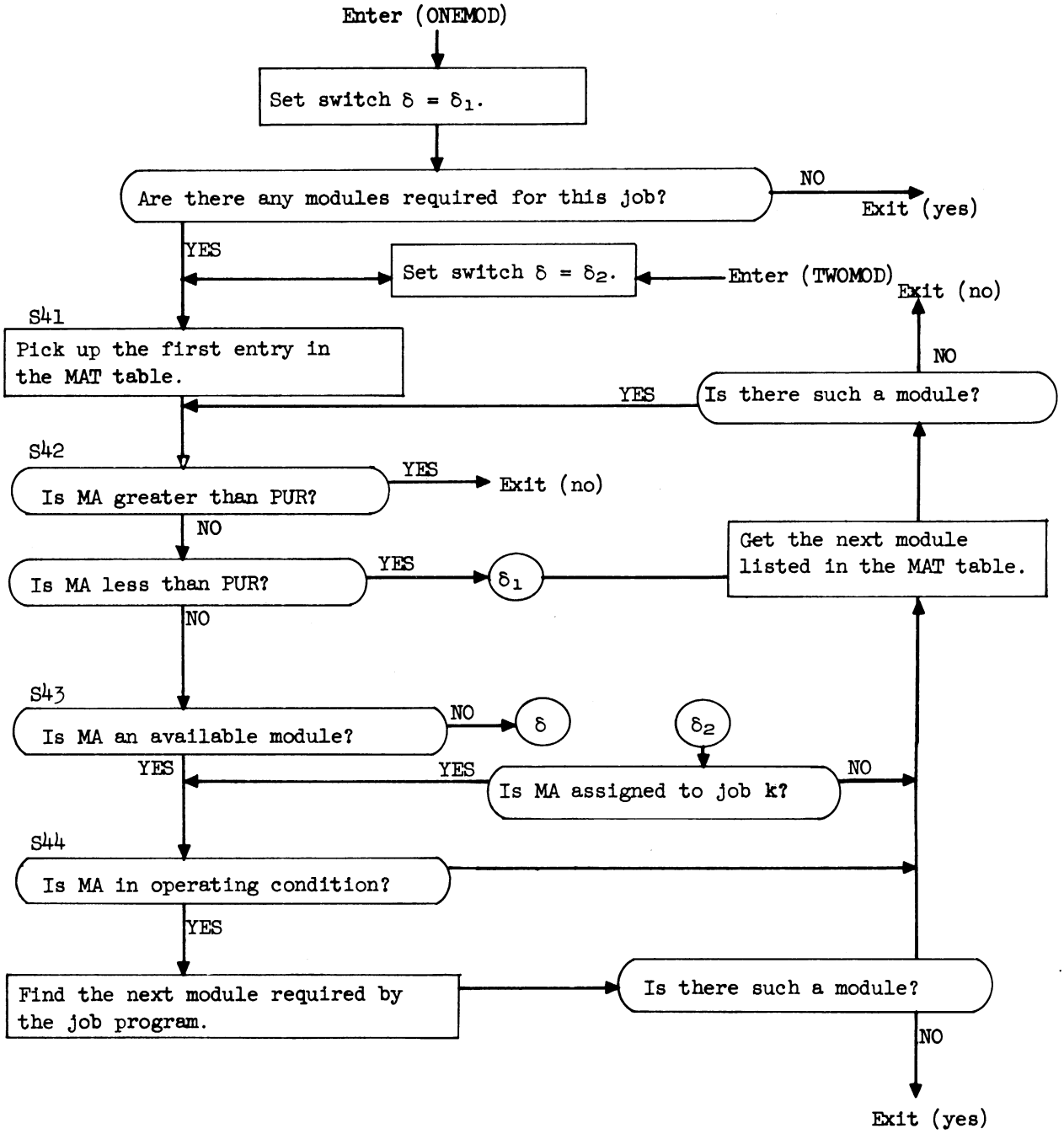

Figure 90. ONEMD subroutine.

TABLE XXXVI

SETMD SUBROUTINE

```
*COMPILE MAD,PUNCH OBJECT                                                    SETMD000
          EXTERNAL FUNCTION                                                          601
          ENTRY TO SETONE.                                                           602
          NORMAL MODE IS INTEGER
          EQUIVALENCE(INPUT(30),DIG),(INPUT(31),DMP),(INPUT(32)
         1 ,DJC),(INPUT(33),DMA),(INPUT(34),DMC),(INPUT(36),DPBQ)
         2 ,(INPUT(37),DPUR),(INPUT(38),DQ),(INPUT(48),MAEND),(IN
         3 PUT(51),PBQEND),(INPUT(53),QEND),(INPUT(54),CQEND),(
         4 MISC(0),LASTL),(MISC(1),LASTK),(MISC(2),COMPLT),(MISC
         5 (3),CXTIME),(MISC(4),ALPHAI),(MISC(5),TLX),(MISC(6),NEWP),(
         6 MISC(7),NEWL),(MISC(8),NEWR),(MISC(9),NEWJ),(MISC(10),NEWPR
         7 ),(MISC(11),Q),(MISC(12),JC),(MISC(13),QBUMP),(MISC(14),
         8 GAMMAI),(MISC(15),PHII),(MISC(16),PII),(MISC(17),K),
         9 (MISC(18),PUR1) ,(INPUT(47),JCEND)
          EQUIVALENCE (MISC(19),PBQU),(MISC(20),BACKLG),(MISC(21),STA
         1 NBY),(MISC(22),INPROG),(MISC(23),COMP),(MISC(24),ASTRSK)
          PROGRAM COMMON IGPLOG,IMPLOG,JCR,MAT,MCT,MISC,PERBUF,PURVUE
         1 ,QUEUE,SEQQ,TIMELG,INPUT,INFIN,TL,J,L,R,CQMAX,PBQMAX,QMAX,T
         2 IME,PUR,OLDTL,OLDTLX,JX,MP,TIME3,RUNNO,RUNNUM
          DIMENSION IGPLOG(195),IMPLOG(144),JCR(34),MAT(324),
         1 MCT(119),INPUT(74),PERBUF(159),PURVUE(367),QUEUE(249),
         2 SEQQ(111),TIMELG(439),MISC(74),TIME3(9)
          DELTAI=3                                                                   603
          TRANSFER TO S45                                                            604
          ENTRY TO UPSET.                                                            605
          DELTAI=4                                                                   606
S45       NEWK=QUEUE(Q-6)                                                            607
          QUEUE(Q)=255                                                               608
          MA=0                                                                       609
          TRANSFER TO DELTA(DELTAI)                                                  610
DELTA(4)  WHENEVER MAT(MA+3).E.K,MAT(MA+3)=-1                                        611
DELTA(3)  WHENEVER MAT(MA).L.PURVUE(PUR1)                                            612
S46       MA=MA+DMA                                                                  613
          WHENEVER MA .E.MAEND,FUNCTION RETURN
          TRANSFER TO DELTA(DELTAI)                                                  614
          END OF CONDITIONAL                                                         615
          WHENEVER MAT(MA+3).NE.-1.OR.MAT(MA+2).NE.0, TRANSFER TO S46                616
          MAT(MA+3)=NEWK                                                             617
          PUR1=PUR1+1                                                                618
          TRANSFER TO S46
          END OF FUNCTION                                                            620
```


C.C. 0041

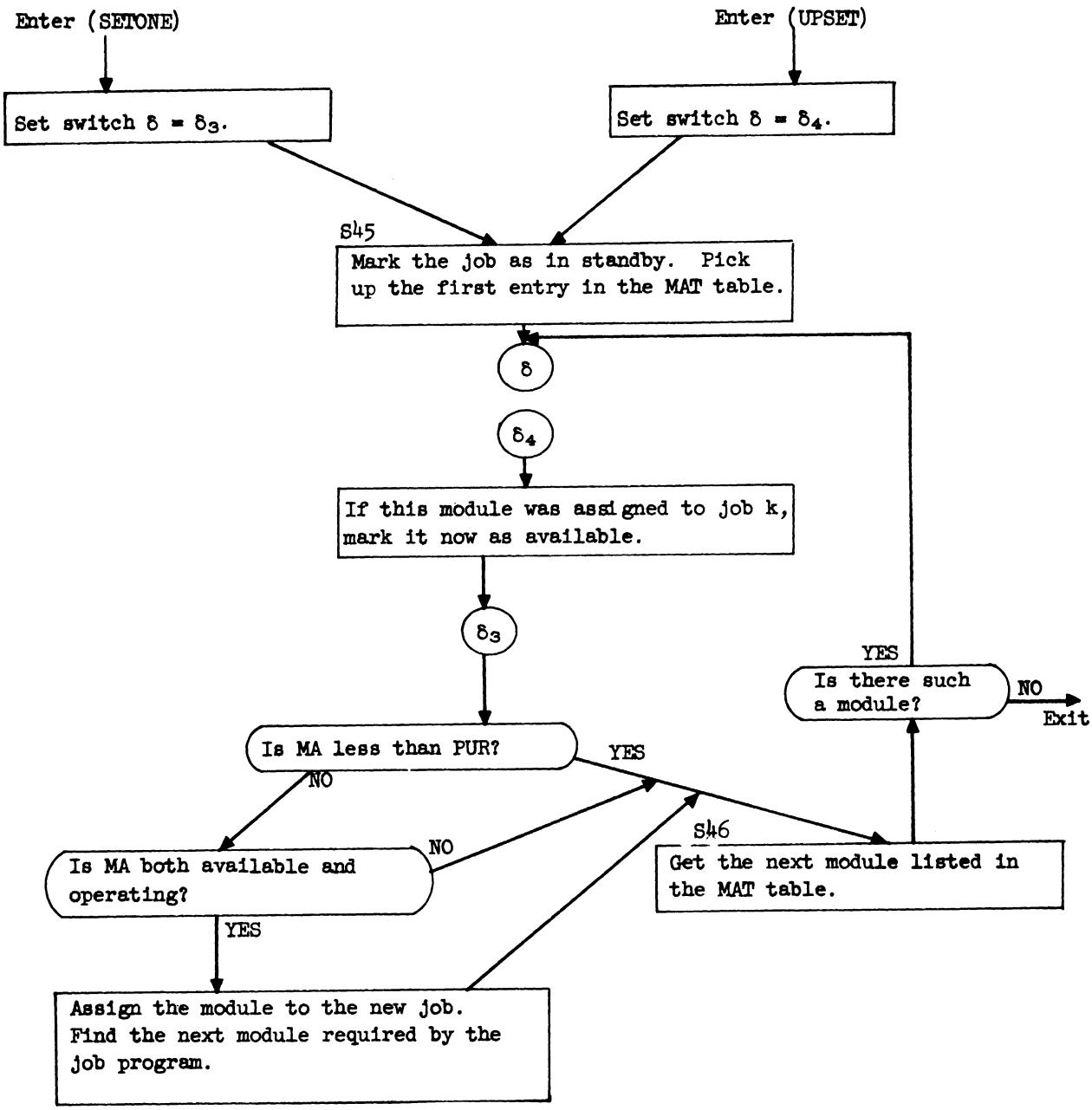

Figure 91. SETMD subroutine.

APPENDIX B

EXAMPLE OF A SIMULATION RUN

B.1. Introduction

This appendix describes a typical simulation run, assuming a polymorphic system with seven job computers. It serves as an example of the capability of the AN/FSQ-27 simulation, and of the considerations involved in setting up an experiment. This run includes simulation of shared module queueing as well as master computer activity.

B. 2. Description of Simulation Parameters

B.2.1. MASTER PROGRAM

The master program functions are already defined in the simulation program (Sections 3.3.8 and 3.3.9). Associated with the execution of each function, there is, in general, a subordinate module connection (minor joint) to be simulated. In the simulation, one is free to specify the subordinate module to be used and the time duration of the connection for each joint. Table XXXVII gives the modules and time durations which are specified for the master program on this run. Note that joints 8 and 23 are ones in which the MC or JC is disconnected and waiting for an alert; the time indicated for them has no significance except that it must be long in order that the CM does not go on to the next joint before receiving the alert.

Also, it should be pointed out that in this run no priorities are assigned to incoming jobs, and hence no interrupting occurs. As a result, certain joints listed in Table XXXVII, such as 9, 16, and 27, are never executed and their time durations has no effect upon results.

B.2.2. JOB PROGRAMS AND ARRIVALS

In this run, arrivals of requests for processing occur at random, which with an exponential distribution on the time between arrivals, given by

$$f(t)dt = \lambda\epsilon^{-\lambda t}\, dt \qquad (120$$

TABLE XXXVII

MASTER PROGRAM DESCRIPTION

Joint	Module Code	Function	Time (μsec)
1-PB	20	Get input job	1,000
2-MT	40	Get purview data	140,000
3-MD	10	Backlog entry	30,000
4-MD	10	Scan standby	55,000
5-PB	20	Output report	1,000
6-MD	10	Scan standby	260,000
7-CX	30	Alert	0
8-0	0	Wait	34,359,738,367
9-MD	10	Bump job	240,000
10-MD	10	Check JC status	1,000
11-MD	10	Respond to alert	2,000
12-CXM	50	Load memory	0
13-CX	30	Alert JC	0
14-MD	10	Alert response	240,000
15-CX	30	Remove alert	0
16-PB	20	JC interrupt report	1,000
17-MT	40	Load program	200,000
18-MD	10	Mark job in progress	1,000
19-0	0	Start job	0
20-0	0	End job	0
21-MD	10	Complete job	26,000
22-CX	30	Alert MC	0
23-0	0	Wait	34,359,738,367
24-CX	30	Remove alert	0
25-MD	10	Enter alert	1,000
26-CX	30	Alert MC of interrupt	0
27-MT	40	Dump job	40,000
28-PB	20	Input/output	1,000

with $\lambda = 5$ per minute.

Since one type of job request is considered, the statistical properties involved in representing job execution are the same for all requests. No priorities are assigned, and only a JC is required to accomplish processing of any job. Job execution is represented by 1000 minor joints for input/output (joint 28), each having the same duration and separated by a time interval of random duration for computation. Major joints have a distribution as shown in Figure 92.

It can be seen from Figure 92 that most of the input/output connections during job execution occur at intervals approximately equal to the service time for the shared module (the PB). Hence, there is considerable queueing for the shared module.

B.2.3. EQUIPMENT PARAMETERS

The simulation program allows arbitrary specification of the distribution on access time for drums, tapes, and the central exchange memory unit (Section A.1.2). (The time for loading the CXM is considered to be its access time in the simulation.) One may also specify the constant switching time at the central exchange.

In this run these parameters are all given the constant values listed in Table XXXVIII, which, along with Table XXXVII summarizes the description of the run.

TABLE XXXVIII

SUMMARY DESCRIPTION OF THE RUN

Modules	1 MC, 7 JC, 1 MD, 1 MT, 1 PB, 1 CX, 1 CXM
Mean arrival rate	5 per minute
Major joint duration	Distribution of Figure 92 Mean = .024 sec Variance = .0461 sec^2
Number of major joints	1000/job
Minor joint duration	9.58 msec
Initial conditions	All queues empty, MC on joint 1, JC's on joint 23
CX time	80 μsec
Tape access time	2.8 sec
Drum access time	8.5 msec
CX memory access time	2.0 msec

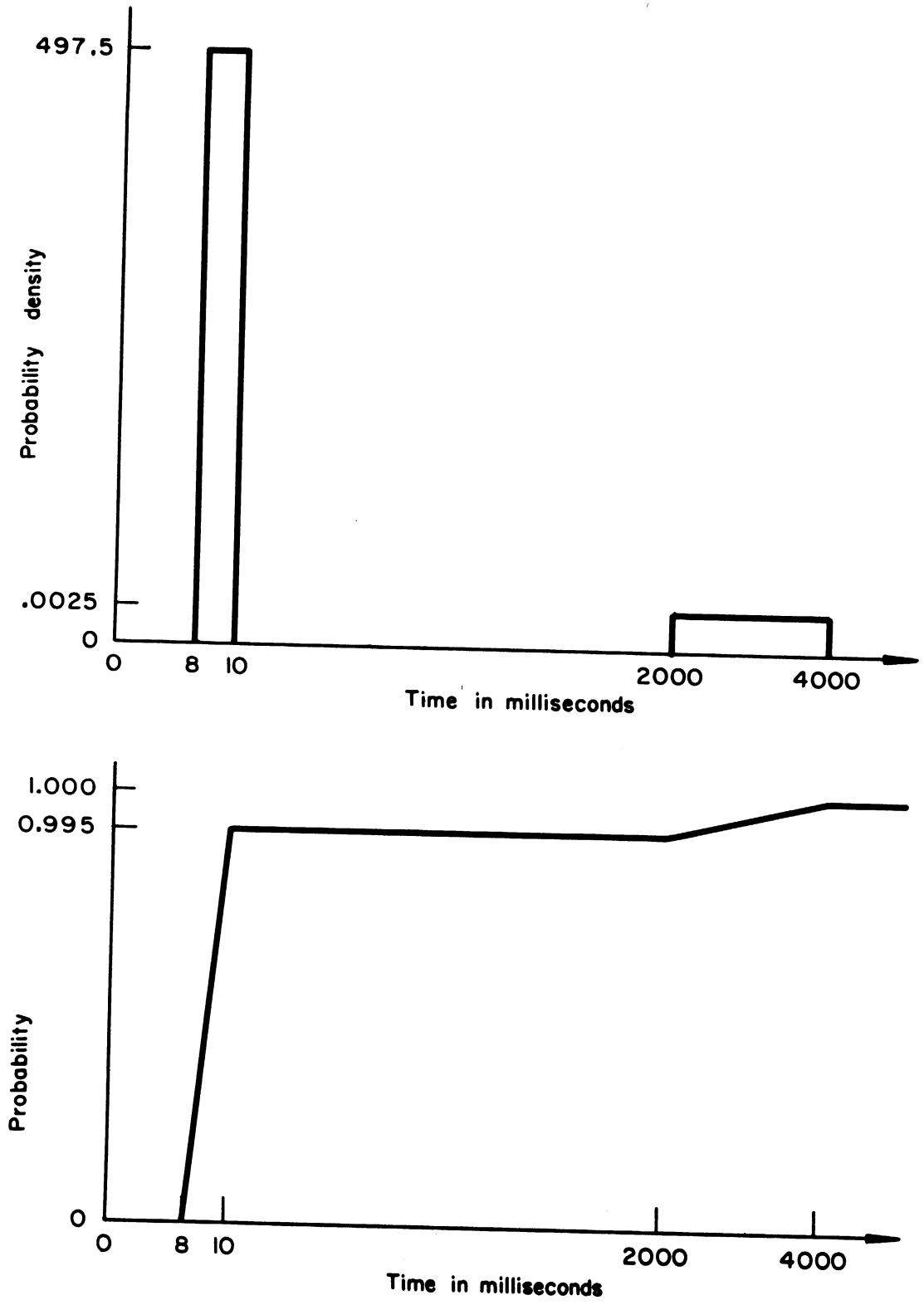

Figure 92. Probability distribution of major joint durations.

Note that the total time duration for each master program joint of Table XXXVII includes, in addition to the value listed there (the computation time for each joint), the CX switching time and appropriate access time from Table XXXVIII. For example, the total duration of the input/output connection of the PB is 1000+80+8500 = 9580 μsec.

B.3. Results of the Run

B.3.1. GENERAL

The run simulates 1287 seconds of system operation, with 116 problem request arrivals. From this number, the first 100 arrivals constitute the sample space on which results are based.

The IBM-704 computer time required for the run is 85 minutes. This result compares well with the predicted running time, whose calculation is summarized in Table XXXIX. It is clear that the majority of the running time results from the simulated competition for the shared module, i.e., attempted minor joints. Running time would be appreciably reduced if the number of joints per job program were smaller, and shared module queueing omitted (see Table XX, Section 5).

TABLE XXXIX

CALCULATION OF PREDICTED RUN TIME

Type of Joint	Est. Number in the Run	Approximate Simulation Time Per Joint	Running Time
Master program	25,600	20 msec	512 sec
Job program minor joints	115,000	10 msec	1150 sec
Job program major joints	115,000	9 msec	1035 sec
Attempted minor joints	287,500	12 msec	3450 sec
Total estimated running time = 6147 sec or 102.4 min.			

B.3.2. JOB ARRIVALS

The actual arrival time for the first 100 arrivals is compared in Figure 93 with the theoretical result for the exponential distribution, Eq. (120). This distribution appears reasonably accurate for such a small sample. Table XL compares the mean and variance of the arrival interval with the theoretically expected values.

TABLE XL

COMPARISON OF ARRIVAL TIME DISTRIBUTION

Parameter of Distribution	Sample Value	Theoretical Value
Mean	11.21 sec	12.0 sec
Variance	92.1 sec^2	144.0 sec^2

To demonstrate that there is no strong relationship between successive values of the arrival interval, a scatter diagram was made of the uniform random numbers which generate the intervals between job request arrivals. This is shown in Figure 94. Since the points are scattered rather evenly in the unit square, one concludes that successive arrival intervals are independent, as expected.

Note, however, that successive arrival intervals are not generated from successive numbers of the random number generator in the simulation program. This is true because the arrival history is generated as the simulation proceeds, and in the interval between successive arrivals a great many random numbers are taken to generate major joint durations. Figure 94 provides a check on the long-term correlation properties of the random number generator.

B.3.3. JOB EXECUTION TIME

Because the recorded output of the simulation does not include values of the major joint durations, it is not possible to check their distribution against Figure 92. However, job execution time, defined as the time interval between joints 19 and 20 of the master program, will consist of the sum of time durations for 1000 major joints, 1000 minor joints (joint 28), and all waiting time for the shared module. Excluding the waiting time, job execution time should therefore have an average value of 33.58 seconds, and a variance of 46.1 sec^2.

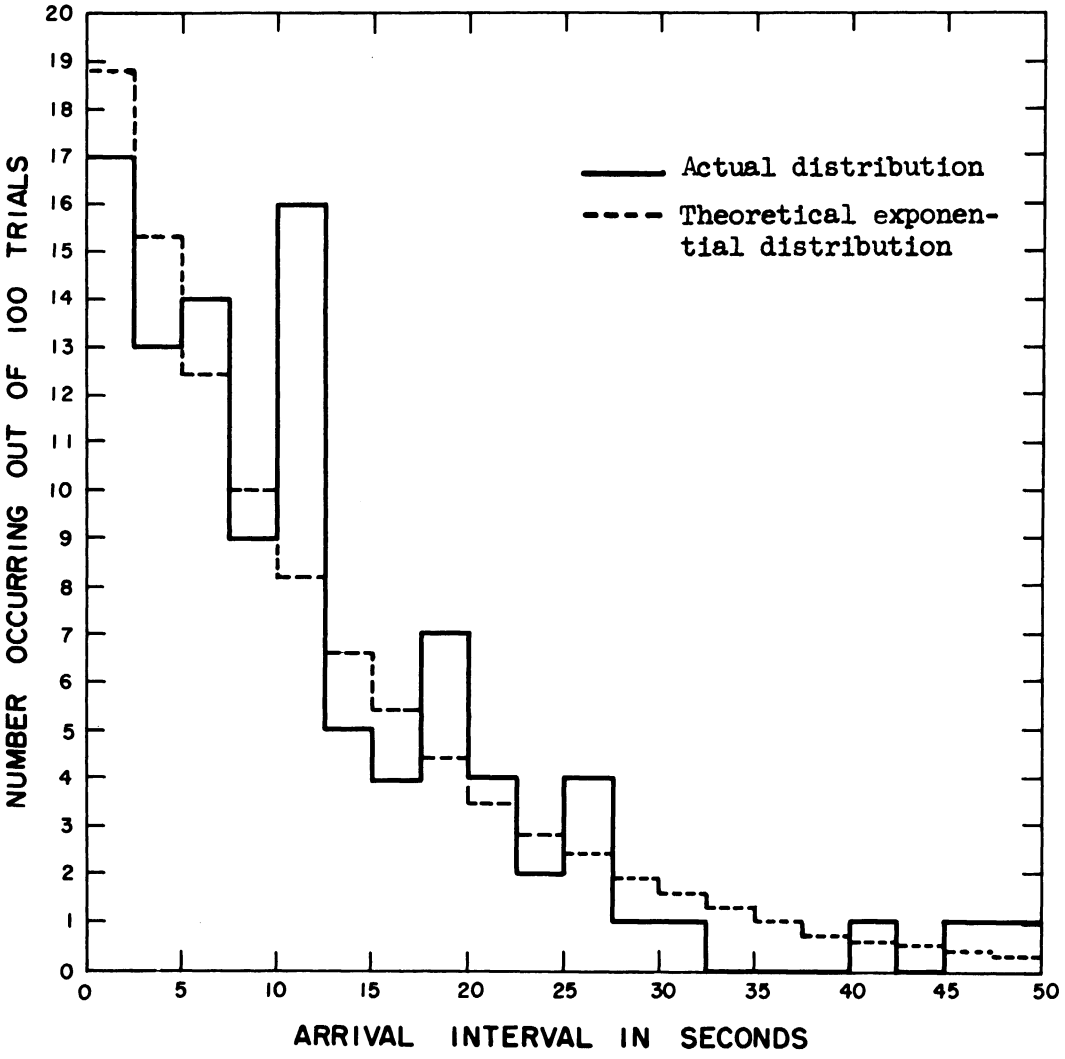

Figure 93. Comparison of distribution on arrival intervals.

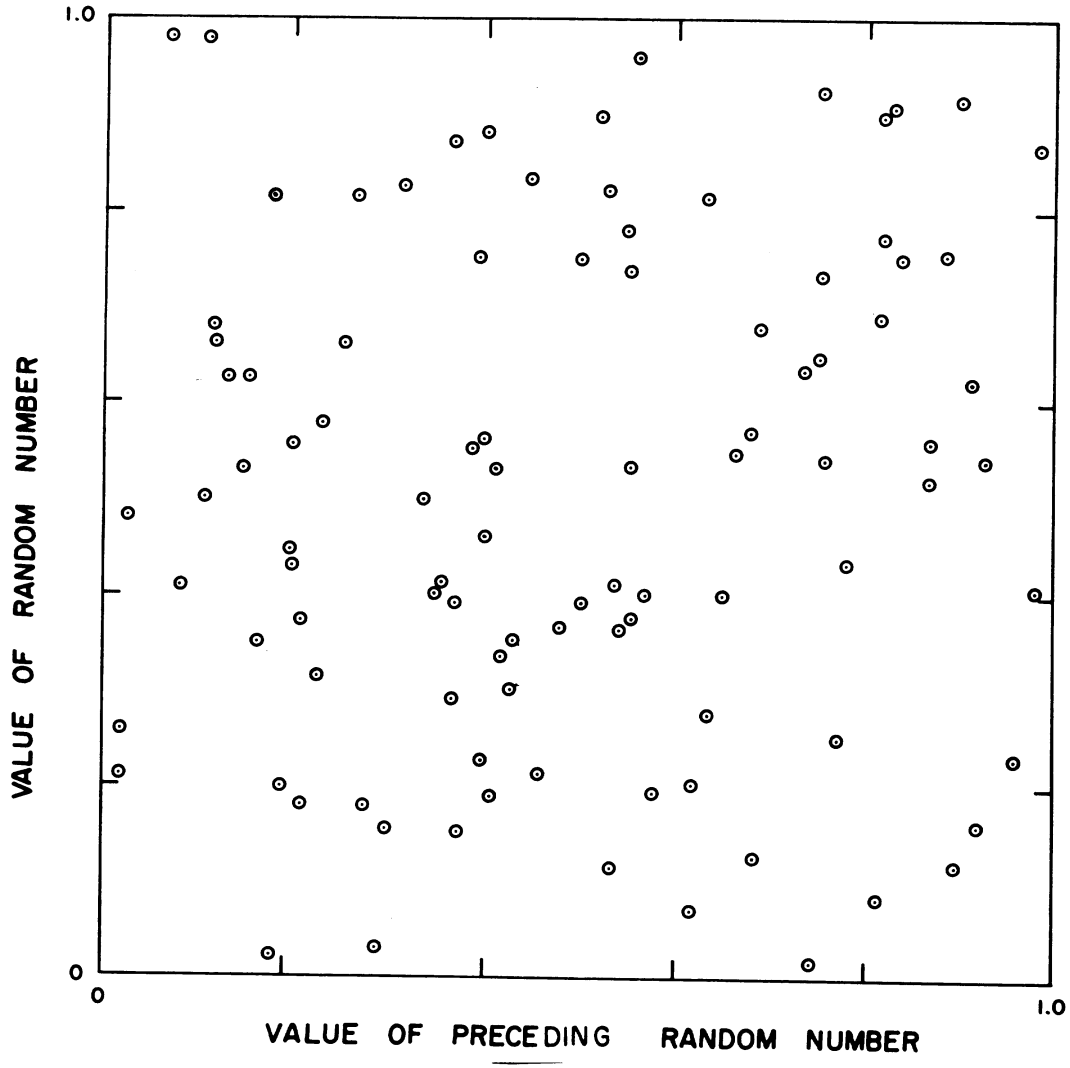

Figure 94. Scatter diagram of 100 uniform numbers generating the exponential arrival intervals.

The values obtained from the data are 53.13 seconds for the mean value, and 160.7 sec^2 for the variance. Assuming the major joint distribution, Figure 92, for the run, the average JC waiting time per request for the shared module is 19.6 msec, and the variance of the waiting time per request is 0.1146 sec^2.

Since JC waiting time is not useful in advancing job processing, the shared-module queueing reduces the effective use of JC job processing time. In this run,

$$\text{avg. use of JC job processing time} = \frac{(33.58)}{33.58 + 19.6} \times 100 = 63\%$$

where 33.58 sec is the job execution time exclusive of waiting time, and 19.6 sec is the average total JC waiting time per job. The output of the simulation does not at present provide information for calculation of shared module utilization factor.

Table XLI summarizes the results for job execution time. Figure 95 compares the distribution of job execution times with the theoretical normal distribution having the same mean and variance. Since job execution time is the sum of a large number of random variables, it is expected (according to the Central Limit theorem) to have a normal distribution. The comparison indicates that the sample is not large enough to assure a good fit, but the discrepancy does not seem unreasonable for this small sample.

TABLE XLI

RESULTS FOR JOB EXECUTION TIME

Sample Source	Mean Execution Time (sec)	Variance of Execution Time (sec^2)
First 100 jobs	53.13	160.7
First 20	48.02	159.0
Second 20	61.68	78.9
Third 20	48.20	123.6
Fourth 20	54.90	124.2
Fifth 20	52.92	213.2
First 40	54.85	163.8
Middle 40	51.54	---
Last 40	53.90	165.53

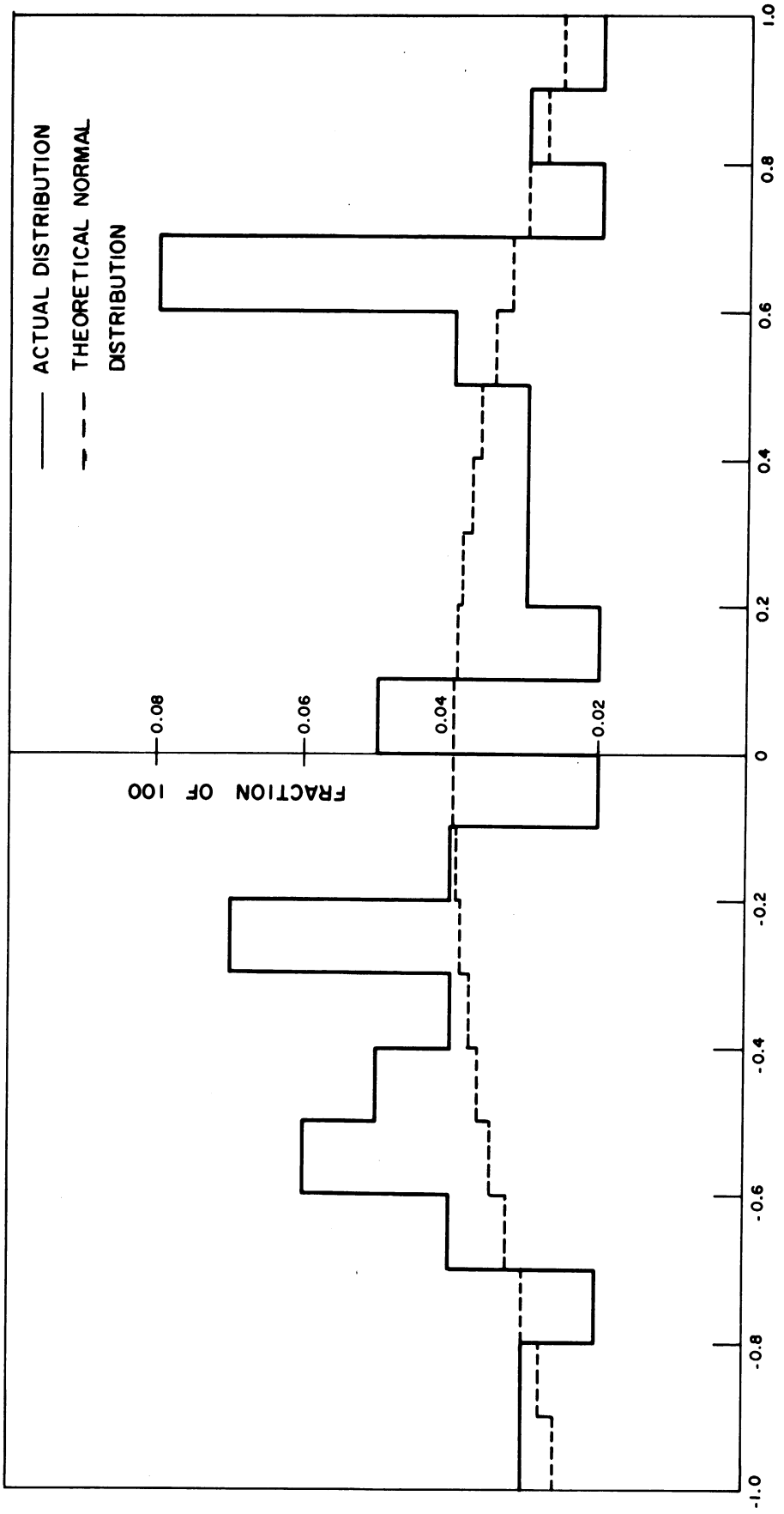

Figure 95. Comparison of distribution of job execution time with normal distribution.

B.3.4. THROUGHPUT TIME

Throughput time for a job is the sum of the job execution time, master program processing time, and total waiting time in the system queues. The value for average throughput time obtained from the data is 66.4 seconds. Hence, master program processing time and waiting time on the average amount to 13.3 seconds.

The minimum value for MP processing time is the sum of the total durations for MP joints 1, 2, 3, 1, 4, 12, 13, 17, 18, 21, 4, and 5, or about 6.2 seconds. Hence, average waiting time in the system queue is about 7 seconds.

B.3.5. AVERAGE NUMBER OF JOBS IN THE SYSTEM, AND STATE PROBABILITIES

Calculations were made of the average number of jobs in the system, average number of jobs in progress, and the probabilities of a given number of jobs in the system in the sample run.

The average number of jobs waiting in the system queues is the difference of the total number in the system and the number in progress. These figures are shown in Tables XLII and XLIII.

TABLE XLII

RESULTS FOR THE NUMBER OF JOBS IN PROGRESS AND IN THE SYSTEM

Quantity	Mean	Variance
Number in the system	5.79	5.38
Number in progress	5.3	
Number in the queues	0.4	

TABLE XLIII

STATE PROBABILITIES OBTAINED FROM DATA

n	P_n
0	0
1	.0346
2	.0587
3	.1135
4	.0859
5	.1577
6	.1291
7	.1437
8	.1558
9	.0850
10	.0375

At no time are there more than 10 jobs in the system simultaneously, nor is the system ever completely empty. It is therefore evident that the state probabilities are not very accurate.

B.4. Conclusions

The length of run taken in this example is sufficiently long to yield accurate results in measurements of mean values, but not in measurements of state probabilities or distributions. This is consistent with what has been shown in Section 5 regarding statistical accuracy of results.

The basic aspect of system operation represented in this run involves the interaction of multiple queue processes, namely, the queueing of jobs at the system input and the queueing of computers actively engaged in job processing for use of a shared subordinate module. This is a more complex problem to treat analytically than those considered in Section 4. Recall that in Section 4 a discussion was given of queueing for a shared module when seven job computers were continuously busy processing job requests. That discussion is not applicable to this run since, here, all job computers are not continuously busy because of the random nature of arrivals at the system input. Furthermore, the number of JC's that are busy depends on the shared-module queueing process, since waiting time for use of the shared module affects the service-time distribution for the job.

This run is therefore an example of simulation of a complex system for which not even a crude analytical model is presently available. Because of the length of run, only steady-state mean value measures of performance are accurate. Among those that may be calculated are mean number of jobs in the system and mean waiting time for use of the shared module. From the latter it is possible to calculate average use of available JC time, but the present simulation output does not provide information on utilization factor for the shared module.

The simulation program runs rather slowly in a situation such as the one described in this appendix. However, runs indicate that increased speed may be obtained when system activity is simplified, e.g., by omitting competition for the shared module.

APPENDIX C

AUXILIARY PROGRAMS

This appendix describes briefly several programs written for the IBM-704 computer to perform numerical calculations of the solutions of queueing problems. The programs were written in the MAD (Michigan Algorithm Decoder) language, using subroutines available on the library tape at The University of Michigan Computing Center. A listing is given of the MAD statements, and a description of input card format and printed output format is provided.

C.1. Transient Solutions for the Exponential Model

This program solves the time-dependent equations of detailed balance for the multiple-channel system with exponential arrival- and service-time distributions. The numerical solution of these equations is accomplished by the Runge-Kutta fourth-order method, available as a library subroutine.

The following equivalents exist between the notation used in the program and the notation used in writing the equations in Section 4:

$$Y(J) \equiv P_{J-1}$$

$$F(J) \equiv \dot{P}_{J-1}$$

The quantity X is equivalent to time, or normalized time, depending on how the equations are written (see Section 4.2). In the program, the equations are written in the form

$$F(J) = A(J,1) \cdot Y(J-1) + A(J,2) \cdot Y(J) + A(J,3) \cdot Y(J+1) \quad (121)$$

where:

A(J,1), A(J,2), A(J,3) are the coefficients determined from the transition probabilities.

For example, the equation

$$\dot{P}_7 = 7\rho P_6 - (7\rho+7)P_7 + 7P_8$$

which is scaled to the service time for a single channel, would appear in the program as

$$F(8) = 7\rho \cdot Y(7) - (7\rho+7) \cdot Y(8) + 7 \cdot Y(0)$$

with

$$A(8,1) = 7\rho$$

$$A(8,2) = -(7\rho+7)$$

$$A(8,3) = 7$$

The input data to the program are as follows:

N = integer number of equations to be solved (1 + maximum allowed state)

X = initial value of time from which calculation is to begin

XFIN = final value of time for which calculations are to be made

H = step size in X for the numerical calculation

A(1,1)...A(N,3) = values of the coefficient.

The card format for the input data is described in Table XLIV. Each line in the table corresponds to one card. All quantities to be specified are decimal numbers, except N, which is an integer whose maximum allowed value is 65. The value of N must be justified to the right of its field when punched on the card. Decimal numbers may be located anywhere in their proper fields, provided the decimal point is punched. If the decimal point is not specified, the program will presume its location to be between the 9^{th} and 10^{th} digits from the right side of the field. As indicated in Table XLIV, initial conditions are punched 4 to a card until N of them have been specified. Coefficients are punched 3 to a card, giving a total of N coefficient cards.

The printed output of the program includes the values of P_0, P_1,...P_{N-1} and $\dot{P}_{N-1}$ for every value of X. The maximum state of the system (N-1) is printed at the beginning of the output for each solution, along with the coefficients, for convenience of checking and identification. Table XLV describes the format of the output.

The time required to perform a complete solution depends, of course, on the number of equations to be solved and the desired number of solution points. It has been observed that the program computes 72 points per minute in the solution of 21 equations, which indicates its speed.

The MAD statements which constitute the program are given in Table XLVI.

TABLE XLIV

MAKEUP OF INPUT DECK FOR ONE SOLUTION

Card Column Number			
1 — 18	19 — 36	37 — 53	54 — 72
N	X	XFIN	H
Y(1)	Y(2)	Y(3)	Y(4)
Y(5)	Y(6)	Y(7)	Y(8)
.	.	.	.
.	.	.	.
.	.	.	.
A(1,1)	A(1,2)	A(1,3)	
A(2,1)	A(2,2)	A(2,3)	
.	.	.	
.	.	.	
.	.	.	
A(N,1)	A(N,2)	A(N,3)	

TABLE XLV

PRINTED OUTPUT FORMAT

Highest state is [N-1], coefficients are					
	[A(1,1)]		[A(1,2)]		[A(1,3)]
	.		.		.
	.		.		.
	.		.		.
	[A(N,1)]		[A(N,2)]		[A(N,3)]
Results					
[x]	$[P_0]$	$[P_1]$	$[P_2]$	$[P_3]$	$[P_4]$
	$[P_5]$	.	.	.	.
	.	.	$[P_{N-2}]$	$[P_{N-1}]$	$[\dot{P}_{N-1}]$
[x + H]	$[P_0]$	$[P_1]$	$[P_2]$	$[P_3]$	$[P_4]$
	$[P_5]$	.	.	.	.
	.	.	$[P_{N-2}]$	$[P_{N-1}]$	$[\dot{P}_{N-1}]$
	.	.	.	.	.
	.	.	.	.	.
[xfin]	$[P_0]$	$[P_1]$	$[P_2]$	$[P_3]$	$[P_4]$
	$[P_5]$	.	.	.	.
	.	.	$[P_{N-2}]$	$[P_{N-1}]$	$[\dot{P}_{N-1}]$

TABLE XLVI

$\dot{P}] = [U]\ P]$ PROGRAM

```
          INTEGER N,J
          DIMENSION Y(66), Q(66), A(198, D(1))    ,F(66)
          VECTOR VALUES NOTE = $ S28,23HPRESENT VALUE OF MEAN = F18.3,1
         15H ,MEAN SQUARE = F18.3*$
          VECTOR VALUES INTIAL = $ I18,3F18.9*$
          VECTOR VALUES INCOND = $ 4F18.9*$
          VECTOR VALUES COEFF = $ 3F18.9 *$
          VECTOR VALUES D(1) = 2,1,3
          VECTOR VALUES RMRKA = $17H1HIGHEST STATE IS I3,19H. COEFFICIE
         1NTS ARE //*$
          VECTOR VALUES HEAD = $ 8H4RESULTS   //*$
          VECTOR VALUES RMRKB = $ (S33,3F18.6)*$
          VECTOR VALUES RMRKC = $ 1H0,F18.3,5E18.5/(S19,5E18.5)*$
          EXECUTE SETRKD. (N,Y(1),F(1),Q,X,H)
START     READ FORMAT INTIAL, N,X,XFIN,H
          READ FORMAT INCOND, Y(1)...Y(N)
          READ FORMAT COEFF, A(1,1)...A(N,3)
          PRINT FORMAT RMRKA, N-1
          PRINT FORMAT RMRKB, A(1,1)...A(N,3)
          PRINT FORMAT HEAD
          PRINT FORMAT RMRKC, X, Y(1)...Y(N)
RKCOMP    S = RKDEQ. (0)
          WHENEVER S.E. (2.0), TRANSFER TO NEXTPT
          THROUGH LOOP, FOR J= 1,1,J.G.N
LOOP      F(J) = A(J,1) * Y(J-1) + A(J,2) * Y(J)+ A(J,3) * Y(J+1)
          TRANSFER TO RKCOMP
NEXTPT    PRINT FORMAT RMRKC,X,Y(1)...Y(N),F(N)
          WHENEVER X.GE.XFIN,TRANSFER TO START
          TRANSFER TO RKCOMP
          END OF PROGRAM
```

C.C. 0030

C.2. First-Passage Time Distribution

The program which computes transient solutions for the exponential model will also compute the distribution on first-passage time to a given state. The necessary modification of the equations of detailed balance has been discussed in Section 4.5. The output of the program gives both the probability-density function of the first-passage time

$$p(t_{o,n} = x) = P_{N-1}(x)$$

and the probability-distribution function

$$P(t_{o,n} \leqslant x) = \dot{P}_{N-1}(x)$$

An addition to the program has been made to provide a calculation of the mean and mean square of the first-passage time via the triangular rule. Otherwise the description given in C.1 applies to this program.

C.3. Computation of Roots

A program has also been written to compute the roots of the characteristic equation describing the transient solution in the exponential model (see Section 4.3.1.3). The program makes use of a library routine for computing the roots of a polynomial, which is assumed to have the form

$$a_o x^N + a_1 x^{N-1} + \ldots + a_{N-1} x + a_N \qquad (122)$$

The input to the program is as follows:

A(0) = real part of a_o

A(1) = imaginary part of a_o

A(2) = real part of a_1

.
.
.

A(2N+1) = imaginary part of a_N

The card format for the input is described in Table XLVII. The value of N for each solution is given on the first card of the deck; the largest allowed value is 49. The real and imaginary parts of each coefficient are punched on the same card. These are decimal numbers and may be located anywhere in the proper field if the decimal point is also punched. Otherwise, the program will assume the decimal point to be between the 9^{th} and 10^{th} digits from the left of the field.

The output of the program is a listing of the real and imaginary parts of the roots. In general, roots of multiplicity greater than 2 cannot be found.

TABLE XLVII

INPUT FORMAT FOR ROOT COMPUTATION

Card Column		
1 3	18 19 36	80
N		
A(0)	A(1)	
A(2)	A(3)	
.	.	
.	.	
.	.	
.	.	
A(2N)	A(2N+1)	

The MAD statements are listed in Table XLVIII.

C.4. Transient Solution for Erlang 2 Service Time

The program that computes the transient state probability solutions with Erlang Type Two service-time distribution is more complex than that for the exponential distribution. The state of the system is described by two subscripts: n, the total number in the system, and s, the number of units in the last phase.[22] However, the Runge-Kutta routine allows only

[22]The definition of s is somewhat different here from that in Section 4.2.

TABLE XLVIII

ROOT COMPUTATION PROGRAM

```
          INTEGER N
          DIMENSION A(100),R(100)
          VECTOR VALUES INPUT = $ I3/(2F18.9)*$
          VECTOR VALUES DATA=$4H1N =I5,33H ,  COEFFICIENTS A(0) TO A(N)
         1ARE  /S11,4HREAL,S14,4HIMAG /(1H ,2E18.6)*$
          VECTOR VALUES RMRK = $ 6HOROOTS//*$
          VECTOR VALUES OUTPUT = $ 1H ,2E18.6*$
          VECTOR VALUES ERR1 = $8H ERROR 2*$
          VECTOR VALUES ERR2 = $8H ERROR 3*$
          VECTOR VALUES ERR3 = $8H ERROR 4*$
START     READ FORMAT INPUT, N, A(0)...A(2*N+1)
          PRINT FORMAT DATA, N, A(0)...A(2*N+1)
          PRINT FORMAT RMRK
          M = ZER2. (N,A,R)
          WHENEVER M.E. 1.0
          PRINT FORMAT OUTPUT, R(0)...R(2*N-1)
          TRANSFER TO START
          OR WHENEVER M.E. 2.0
          PRINT FORMAT ERR1
          TRANSFER TO START
          OR WHENEVER M.E. 3.0
          PRINT FORMAT ERR2
          TRANSFER TO START
          OTHERWISE
          PRINT FORMAT ERR3
          TRANSFER TO START
          END OF CONDITIONAL
          END OF PROGRAM
```

C.C. 0028

single subscripts. It was therefore necessary to write the equations of detailed balance in the program so that the state probabilities P(n,s) could be referred to with single subscripts.

The equations of detailed balance have the form:

$$\dot{P}(n,s) = a_{ns}P(n-1,s) + b_{ns}P(n,s-1) + c_{ns}P(n,s) + d_{ns}P(n+1,s+1) \qquad (123)$$

Implementing these equations in the program involved setting

$$P(n,s) = Y(I),$$

then

$$P(n-1,s) = Y(I-n) \qquad n \leqslant M$$

$$P(n-1,s) = Y(I-M-1) \qquad n > M$$

$$P(n,s-1) = Y(I-1)$$

$$P(n+1,s+1) = Y(I+n+2) \qquad n \leqslant M$$

$$P(n+1,s+1) = Y(I+M+2) \qquad n > M$$

and the equations were therefore written as

$$F(I) = A(I,1)Y(I-J) + A(I,2)Y(I-1) + A(I,3)Y(I) + A(I,4)Y(I+L) \qquad (124)$$

where:

$$J = n \qquad \text{for } n \leqslant M$$

$$J = M+1 \qquad \text{for } n > M$$

$$L = n+2 \qquad \text{for } n \leqslant M$$

$$L = M+2 \qquad \text{for } n > M$$

and

$$A(I,1) = a_{ns}$$

$$A(I,2) = b_{ns}$$

$A(I,3) = c_{ns}$

$A(I,4) = d_{ns}$

are the coefficients determined by the transition probabilities.

In the program, the following quantities are used:

N = total number of equations to be solved

NMAX = maximum allowed state of the system

M = number of service channels

K = n, total number in the system.

I, J, and L are used in the program as described above. The number equations to be solved

$$N = \frac{M(M+1)}{2} + (NMAX-M)(M+1)$$

The format of the program input is described in Table XLIX. The limit

TABLE XLIX

INPUT FORMAT FOR TRANSIENT SOLUTIONS WITH ERLANG 2 SERVICE TIME

Card Column				
1 6	7 12	13 18	19 36	37 54	55 72
N	M	NMAX	X	XFIN	H
Y(0)			Y(1)	Y(2)	Y(3)
Y(4)			.	.	.
.			.	.	.
.			.	.	.
.			.	.	.
.			.	.	.
A(0,1)			A(0,2)	A(0,3)	A(0,4)
A(1,1)			A(1,2)	A(1,3)	A(1,4)
A(N-1,1)			A(N-1,2)	A(N-1,3)	A(N-1,4)

TABLE L

$\dot{P}] = [U]\ P]$ (ERLANG 2) PROGRAM

```
          INTEGER I,J,K,L,END,DEL,M,NMAX,N
          DIMENSION Y(140),Q(140),F(140),P(20)
          DIMENSION A(560,D(1))
          VECTOR VALUES D(1) = 2,4,4
          VECTOR VALUES INTIAL = $ 3I6,3F18.9*$
          VECTOR VALUES INCOND = $ 4F18.9*$
          VECTOR VALUES NOTEA = $13H1SOLUTION OF I3,8H EQS OF I3,40H CH
         1ANNEL QUEUEING SYSTEM WITH MAX STATE I3///S36,12HCOEFFICIENTS
         1*$
          VECTOR VALUES NOTEB = $ S5,4E18.5*$
          VECTOR VALUES DATA = $1H0,F18.3,5E18.5/(S19,5E18.5)*$
          VECTOR VALUES HEAD = $8H4RESULTS*$
          EXECUTE SETRKD. (N,Y,F,Q,X,H)
START     READ FORMAT INTIAL , N,NMAX,M,X,XFIN,H
          READ FORMAT INCOND, Y(0)...Y(N-1)
          THROUGH STOP1,FOR I =0,1,I.G.N  -1
STOP1     READ FORMAT INCOND, A(I,1)...A(I,4)
          PRINT FORMAT NOTEA,N,M,NMAX
          THROUGH STOP2,FOR I=0,1,I.G.N-1
STOP2     PRINT FORMAT NOTEB, A(I,1)...A(I,4)
          PRINT FORMAT HEAD
          S=2.0
          TRANSFER TO FIRST
RKCOMP    S=RKDEQ.(0)
FIRST     END=0
          THROUGH LOOP,FOR K=0,1,K.G.NMAX
          P(K)=0.
          WHENEVER K.G.M
          J=M+1
          DEL=END
          END = END + M + 1
          OTHERWISE
          DEL=END
          END=END+K+1
          J=K
          L=K+2
          END OF CONDITIONAL
          THROUGH LOOP,FOR I=DEL,1,I.E.END
          WHENEVER S.E.2.0, TRANSFER TO NEXTPT
          F(I)=A(I,1)*Y(I-J)+A(I,2)*Y(I-1)+A(I,3)*Y(I)+A(I,4)*Y(I+L)
          TRANSFER TO LOOP
NEXTPT    P(K) = P(K) + Y(I)
LOOP      CONTINUE
          WHENEVER S.E.2.0,TRANSFER TO PROB
          TRANSFER TO RKCOMP
PROB      PRINT FORMAT DATA,X,P(0)...P(NMAX)
          WHENEVER X.GE.XFIN,TRANSFER TO START
          TRANSFER TO RKCOMP
          END OF PROGRAM
```

where

$$J = n \qquad \text{for } n \leqslant M$$

$$J = M+1 \qquad \text{for } n > M$$

$$JP = n \qquad \text{for } n \leqslant M$$

$$JP = M \qquad \text{for } n > M$$

and the equations are therefore written

$$\begin{aligned} F(I) = {} & A(I,1)Y(I-J-1) + A(I,2)Y(I-J) \\ & + A(I,3)Y(I) + A(I,4)Y(I+JP) \\ & + A(I,5)Y(I+JP+1) + A(I,6)Y(I+JP+2) \end{aligned} \tag{126}$$

with A(I,1) corresponding to a_{ns}, A(I,2) to b_{ns}, etc. The definitions of N, NMAX, M, X, XFIN, and H are the same as in C.4.

The input format is described in Table LI and the output format is the same as in C.3. MAD statements are listed in Table LII.

TABLE LI

INPUT FORMAT FOR TRANSIENT SOLUTIONS FOR HYPER-EXPONENTIAL SERVICE TIME

Card Column					
1 – 6	7 – 12	13 – 18	19 – 36	37 – 54	55 – 72
N	NMAX	M	X	XFIN	H

Card Column					
1 – 12	13 – 24	25 – 36	37 – 48	49 – 60	61 – 72
Y(0)	Y(1)	Y(2)	Y(3)	Y(4)	Y(5)
Y(6)	.	.	.	.	.
	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
A(0,1)	A(0,2)	A(0,3)	A(0,4)	A(0,5)	A(0,6)
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
A(N-1,1)	A(N-1,2)	A(N-1,3)	A(N-1,4)	A(N-1,5)	A(N-1,6)

TABLE LII

$\dot{P}] = [U]\ P]$ (HYPER-EXPONENTIAL) PROGRAM

```
          INTEGER I,J,K,JP,END,DEL,M,NMAX,N
          DIMENSION Y(140),Q(140),F(140),P(20)
          DIMENSION A(840,D(1))
          VECTOR VALUES D(1) = 2,6,6
          VECTOR VALUES INTIAL = $ 3I6,3F18.9*$
          VECTOR VALUES INCOND = $6F12.9*$
          VECTOR VALUES NOTEA = $13H1SOLUTION OF I3,8H EQS OF I3,40H CH
         1ANNEL QUEUEING SYSTEM WITH MAX STATE I3///S36,12HCOEFFICIENTS
         1*$
          VECTOR VALUES NOTEB = $S5,6E18.5*$
          VECTOR VALUES DATA = $1H0,F18.3,5E18.5/(S19,5E18.5)*$
          VECTOR VALUES HEAD = $8H4RESULTS*$
          EXECUTE SETRKD. (N,Y,F,Q,X,H)
START     READ FORMAT INTIAL , N,NMAX,M,X,XFIN,H
          READ FORMAT INCOND, Y(0)...Y(N-1)
          THROUGH STOP1,FOR I =0,1,I.G.N  -1
STOP1     READ FORMAT INCOND, A(I,1)...A(I,6)
          PRINT FORMAT NOTEA,N,M,NMAX
          THROUGH STOP2,FOR I=0,1,I.G.N-1
STOP2     PRINT FORMAT NOTEB, A(I,1)...A(I,6)
          PRINT FORMAT HEAD
          S=2.0
          TRANSFER TO FIRST
RKCOMP    S=RKDEQ.(0)
FIRST     END=0
          THROUGH LOOP,FOR K=0,1,K.G.NMAX
          P(K)=0.
          WHENEVER K.G.M
          J=M+1
          DEL=END
          END = END + M + 1
          JP=M
          OTHERWISE
          DEL=END
          END=END+K+1
          J=K
          JP=K
          END OF CONDITIONAL
          THROUGH LOOP,FOR I=DEL,1,I.E.END
          WHENEVER S.E.2.0, TRANSFER TO NEXTPT
          F(I)=A(I,1)*Y(I-J-1)+A(I,2)*Y(I-J)+A(I,3)*Y(I)+A(I,4)*Y(I+JP)
         1+A(I,5)*Y(I+JP+1)+A(I,6)*Y(I+JP+2)
          TRANSFER TO LOOP
NEXTPT    P(K) = P(K) + Y(I)
LOOP      CONTINUE
          WHENEVER S.E.2.0,TRANSFER TO PROB
          TRANSFER TO RKCOMP
PROB      PRINT FORMAT DATA,X,P(0)...P(NMAX)
          WHENEVER X.GE.XFIN,TRANSFER TO START
          TRANSFER TO RKCOMP
          END OF PROGRAM
```

DISTRIBUTION LIST

(One copy unless otherwise noted)

Rome Air Development Center
Air Research and Development Command
United States Air Force
Griffiss Air Force Base, Alabama
Attn: Project Engineer
Attn: RAAP
Attn: RAALD
Attn: ROZMSTT
Attn: RAIS, Mr. Malloy
(For Flight Lt. Tanner)

Rome Air Development Center
Air Research and Development Command
United States Air Force
Griffiss Air Force Base, Alabama
Attn: RAOL, Captain Norton

AU
Maxwell Air Force Base, Alabama
Attn: AUL

ASD
Wright-Patterson Air Force Base
Ohio
Attn: ASAPRD

Chief
Naval Research Laboratory
Washington 25, D.C.
Attn: Code 2021

Air Force Field Representative
Naval Research Laboratory
Washington 25, D.C.
Attn: Code 1010

Commanding Officer
USASRDL
Fort Monmouth, New Jersey
Attn: SIGRA/SL-ADT

Chief, Bureau of Ships
Main Navy Building
Washington 25, D.C.
Attn: Code 312

Office of the Chief Signal Officer
Department of the Army
Washington 25, D.C.
Attn: SIGRD

AFPR
Lockland Branch
General Electric Company
P. O. Box 91
Cincinnati 15, Ohio

Chief
Air Force Section
Maag, Germany

P. O. Box 810
A.P.O. 80
New York, New York

AFSC
Andrews Air Force Base
Washington 25, D.C.
Attn: SCSE

ASTIA 10
Arlington Hall Station
Arlington 12, Virginia
Attn: TIPCA