THE UNIVERSITY OF MICHIGAN

INDUSTRY PROGRAM OF THE COLLEGE OF ENGINEERING

A GPSS PRIMER

Thomas J. Schriber

Department of Statistics and Management Science
School of Business Administration
The University of Michigan

February, 1972

IP-841

Preface

For the past several months, work on a manuscript for a book tentatively entitled A GPSS Primer has been in progress. The manuscript is scheduled to be completed in another 6 weeks (March 15). After the standard procedure of editing, type-setting, drafting, and compositing, the book will then be published by John Wiley & Sons, Inc. It is estimated that this production process will take about 10 months beyond the receipt of the finished manuscript, making the publication date January/February, 1973.

In the meantime, to support need for a book of this sort in credit courses at The University of Michigan, and also in Michigan's regular one-week short course on GPSS ("Discrete Systems Simulation Using GPSS/360", July 10-14, 1972), about 80% of the manuscript is being printed in a limited quantity in the present format. Availability of this preliminary printing will hopefully make this already highly-teachable modeling language even more teachable and, from the student's point of view, will hopefully ease the learning process. Initial use of the manuscript will undoubtedly also turn up errors of various kinds which, in turn, will make it possible for the final product to be of higher quality.

Following the pattern set in my earlier attempts to write about GPSS, A GPSS Primer makes heavy use of fully-documented case studies. There are 18 such case studies in this preliminary printing. About 150 problems for practice have also been interspersed throughout the text at appropriate points, so students can test their understanding of the material as they proceed. After the book has been published, solutions to selected problems will very likely be available through Wiley & Sons.

The reader is expected to have had some experience in programming computers, using a language such as FORTRAN, BASIC, PL/I, or ALGOL. The reader is also expected to have had a first course in the pre-calculus probability, with some elementary statistics included. Building on this background, Chapter I provides a brief introduction to computer modeling of systems in which events occur at random. The essence of Chapter I is a FORTRAN model for a one-line, one-server queuing system. This preliminary printing includes only the one-line, one-server queuing model as "Chapter I". The model provides insight into the need to "pre-schedule" events in queuing systems. It serves as a springboard into Chapter II, where GPSS concepts are first introduced. The first GPSS Case Study is then a model for the one-line, one-server queuing system. This makes it possible for the reader to compare the step-by-step, "procedure-oriented" approach taken in FORTRAN with the more macro, "problem-oriented" approach taken in GPSS.

Immediately after the first GPSS Case Study, the Current and Future Events Chains are introduced. Based on the chain concepts, the "internal logic" of the GPSS Processor is discussed, and illustrated through a series of 3 numeric examples. Experience in teaching the language has shown that the chain concepts can be successfully introduced at such an early point. Although the numeric examples are tedious, they do "take the mystery out of GPSS" for the reader. From that point forward, as new GPSS concepts arise, they are interpreted, where appropriate, in terms of the chains. The hoped-for effect will be to produce an analyst who thoroughly understands the GPSS models he builds.

This preliminary printing includes four chapters on GPSS. Most instructors probably would not want to include more material than this in the GPSS portion of an introductory, 3-credit course in computer simulation. The final publication, however,

will contain an additional GPSS chapter dealing with such things as User Chains, Facility pre-emption, Assembly Sets, and discussion about how to build GPSS models which execute relatively efficiently. This additional chapter will be of interest to those who want to "go further" with the language, either in the classroom, or in an industrial setting.

No distribution of this preliminary printing is planned. There may be some individuals, however, who are actively working with GPSS and have need for an exposition of the language such as this. Such persons can get a copy of this printing from Ulrich's Books, Ann Arbor, 48104, at $10.95 postpaid. Any criticisms or suggestions for improvement based on study of this printing can be sent to me, and will be greatly appreciated.

Even now, it is not too early to thank James O. Henriksen, a software expert at The University of Michigan's Computing Center, for many informative conversations about the internal makeup of the GPSS Processor. These conversations have refined my understanding of GPSS considerably. Their effect should be noticeable in the manuscript.

Thomas J. Schriber

Ann Arbor
February 1, 1972

Contents

* See the comments on Chapter 1 in the Preface

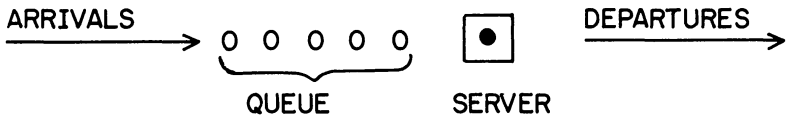
List of Case Studies

Chapter 1[1a]

FORTRAN Model for a One-Line, One-Server Queuing System

1.1 Statement of the Problem

Consider an isolated "system" consisting of a single person who performs some sort of service upon demand. The person might be the attendant at a toll booth on a turnpike, the clerk at the express checkout counter in a supermarket, the girl at a theatre ticket booth, the barber in a one-man barber shop, or the only clerk working in a tool crib. "Customers" come up to this "server" in an irregular pattern, wait their turn for service, are serviced on a first-come, first-served basis, and leave. This situation is illustrated in the following diagram.



QUEUEING SYSTEM

The configuration consisting of the server, the customer being served, and those waiting for service, is termed a queueing system.

The simple queueing system shown above is characterized by two random variables. The time that elapses between consecutive arrivals of customers to the system is a random variable. So is the time required for the server to perform a service. The distributions of these two random variables influence system properties of interest. Some of these system properties, which are themselves random variables, are:

(1) Length of the queue
(2) Time spent by each unit in the system
(3) Percent idle time of the server

System properties such as these are of special interest when economic considerations come into play. For example, if there are already too many waiting for a haircut, a potential customer may go elsewhere for service. In this case, "lost business" results for the barber. Or, if the "customers" are repairmen waiting to obtain tools at a tool crib, the "cost of waiting" vs. the "cost of providing service" must be considered. While they wait, repairmen are "idle" (and machines may be out-of-service), even though they are on the payroll. On the other hand, decreasing the waiting time of repairmen (perhaps by providing more than one tool crib clerk) has the implication of decreasing clerk utilization, i.e., increasing clerk "idle time." In such circumstances, a cost tradeoff clearly exists.[1b]

Build a computer model of a one-line, one-server queueing system. Assume that the interarrival time of customers is uniformly distributed over a range of values. The model should accept as data the average interarrival time, and a corresponding "percent variation," or "percent uncertainty." For example, interarrival time might be 10 minutes, plus or minus 20%; it would then be understood to be uniformly distributed over the interval from 8 to 12 minutes. The model should similarly accept as data the average service time, and a corresponding percent variation. When the

---

[1a] See the comments on Chapter 1 in the Preface.

[1b] For a detailed example of this situation, see Finite Mathematics with Business Applications, by John G. Kemeny, Arthur Schleifer, Jr., J. Laurie Snell, and Gerald Thompson (Prentice-Hall, Inc., 1962), pp. 207-210.

simulation starts, assume that no customers are in the system (the server is idle). The model run should cover a simulated time span inputted as data. Note that, in general, there will be customers in the system at the end of the simulation. Rather than making special arrangements to clear them from the system, simply "leave them there." When the indicated time has been spanned, the model should output:

(1) Duration of the simulation
(2) Server utilization (fraction of the time busy)
(3) Average number of customers in the system
(4) Average length of the waiting line
(5) The fraction of customers who had to wait for service

The model should then accommodate the user by permitting him to input new data (new distributions, or a new span of simulation, or both) or terminate the run. If new data are inputted, the initial system conditions specified above should be restored.

After the model has been built, run it with these data:

(1) Interarrival time is 10 minutes, with no variation
Service time is 10 minutes, with no variation
Duration of simulation is 480 minutes (one 8-hour day)

Note that, for these data, the situation is "deterministic." That is, by providing data with no variation allowed, there are no random elements in the system. In this case, the server will be "engaged" for 470 minutes. He waits 10 minutes for the first customer to arrive; subsequently, just as he finishes service on a customer, the next customer arrives. The server's utilization, then, should be 470/480, or 0.979. If his utilization does not equal 0.979, the model logic is invalid. Unfortunately, a model-measured utilization of 0.979 is no guarantee that the model logic is entirely valid. Use of the deterministic data, then, provides only a partial check on model validity in this case. In general, "deterministic runs" with simulation models provide one of the tools useful in model validation.

(2) Interarrival time is 10 minutes, with 40% variation
Service time is 9.5 minutes, with 40% variation
Simulate with time spans of 480, 960, 1440, 1920, 2400, 2880, and 3360 minutes

For these data, the expected server utilization at "steady state" is 0.95. A start-up condition in the model, however, is that the server will be idle for at least the first 6 minutes of the simulation. (The first customer cannot arrive at simulated time earlier than 6 with the data provided.) As the span of simulation increases, the "bias" in the utilization due to the start-up condition becomes less important. In the long run, model-measured utilization will approach a value of 0.95. For runs of finite duration, though, measured utilization will fluctuate randomly about the 0.95 figure. Trace this fluctuation by plotting the model-measured utilization vs. the simulation span for the data indicated above.

(3) Interarrival time is 10 minutes, with variations of 10, 20, 30, and 40%
Service time is 9.5 minutes, with variations of 10, 20, 30, and 40%
Duration of simulation is 48000 minutes

Run the model for all possible combinations of the variations indicated above. Use the results to produce a plot showing how the probability of having to wait for service depends upon the percent variation in the inputted random variables.

1.2 An Approach Leading to Solution

The task involved here is to move a system forward in simulated time. This means that the various events which can occur in the system must be identified, and their potential effect on the system's status must be articulated. The model must be "taught" how to allow these events to occur. Their occurrence must be faithful to the independent variables (in this case, interarrival times and service times) which govern system behavior.

It is convenient to divide all possible events into two categories: primary events; and secondary events. Primary events are those whose time of occurrence is pre-scheduled by sampling from distributions provided as data. Secondary events are those which occur at the same time as primary events, and as a direct consequence of the primary events. When primary and secondary events occur, it is usually then necessary to pre-schedule other primary events.

It has been said that primary events are pre-scheduled. Consider more closely what this means. Arrival of a customer is a primary event. When a customer arrives, his successor's time-of-arrival is pre-determined by sampling from the interarrival time distribution, then adding the sampled value to the current clock reading. It is this need to pre-schedule arrivals that makes them, by definition, primary events.

To illustrate pre-scheduling with a numeric example, suppose a customer arrives at simulated time 21. A sample is then drawn from the interarrival time distribution. The sampled value is, say, 9. Then the next customer is scheduled to arrive at future time 21 + 9, or 30. When he comes at time 30, the time of his successor's arrival will then have to be pre-scheduled. In short, a "boot-strapping" technique is used to arrange for customer arrivals.

In the above example, occurrence of a primary event called for pre-scheduling the next primary event of like kind. Now consider the other primary event in the system, completion of a service. Service completion is pre-scheduled at the time a customer goes into service. That is, when service is begun on a customer, the time of service completion is predetermined by sampling from the service time distribution, then adding the sampled value to the current clock reading. If a service begins at simulated time 52, and a service time of 11 is drawn, completion of that service is then scheduled to occur at future time 52 + 11, or 63.

Note that a service completion does not necessarily call for pre-scheduling the next service completion. Service completions can logically be pre-scheduled only when (1) a service completion has just occurred and there is another customer waiting to be put into service, or (2) a customer arrives at the system and finds the server idle.

A listing of primary events and their consequences in the one-line, one-server system is now possible.

Primary Event	Consequences (Secondary Events; and Pre-Scheduling)
Arrival of a Customer	(a) Pre-schedule time of successor's arrival (b) Is the server currently available? NO: Put customer in waiting line. YES: Put customer into service. This requires: (i) Updating the server's status, and (ii) Pre-scheduling the time of the service completion.
Completion of a Service	(a) Is there a customer waiting to go into service? NO: Update the server's status. YES: Put the customer into service. This requires: (i) Updating the length of the waiting line, and (ii) Pre-scheduling the time of the service completion.

With the exception of pre-scheduling, the above-indicated consequences of the primary events can be thought of as secondary events. For example, joining a waiting line is a secondary event.

Now that the basic logic for the computer model has been laid out, a series of final points will be taken up.

Model Startup

At model startup, the simulation clock is initially set to a reading of zero. There are no customers in the system, and the server is idle. The arrival of the first customer is pre-scheduled by sampling from the interarrival time distribution, and adding the sampled value to the current clock reading (which is 0). The logic of the system does not call for any additional pre-scheduling at simulated time 0. The next (and only) event scheduled to occur, then, is arrival of the first customer. The simulation clock is advanced to the time of this primary event, and its consequences are carried out.

There are now two events pre-scheduled for future occurrence: time of the second customer's arrival; and time of the first customer's service completion. The event times are tested to determine which of these events is the next to occur. The clock is then advanced to the early event time, and consequences of the event are carried out. The process is continued, event after event. The result is to move the system forward in simulated time.

Model Shutdown

Eventually, as simulated time elapses, the time of model shutdown is reached. Model shutdown can itself be thought of in the spirit of a primary event. Its time of occurrence is scheduled at model startup, when the desired span of simulation is inputted by the model. In general, then, there may be three events pre-scheduled for future occurrence. Whenever it is time to advance the simulation clock, all three events can be tested to determine which occurs next. Ultimately, the next event will be to shut off the model.

No Scheduled Service Completion

If the server is idle, no service completion is scheduled for future time. When testing to find the next event, then, only customer arrival and model shutdown should be considered. Rather than providing special test for this situation, it is more convenient to always use a single test which scans the event times for all three potentially pre-scheduled events (customer arrival; service completion; and model shutdown). But if, as hypothesized, no service is in progress, care must be taken to see that "service completion" is not the indicated next event. This can be done by setting service completion time to some arbitrarily large value. Any large value could be used. One convenient choice is to have the "large value" be shutdown time. When shutdown time is reached, the model goes directly into the shutdown phase anyway. Hence, no attempt is ever made to bring about a non-existent service completion.

Simultaneous Events

Consider what happens when two primary events are scheduled to occur at the same reading of the simulated clock. The clock is first advanced to the time in question. The consequences of one of the primary events are carried out. (The particular primary event selected depends on the way testing for the "next event" is conducted.

In the one-line, one-server system, it doesn't matter which of the two events is selected first.) As usual, the pre-scheduled events are then tested to find which is most imminent. By hypothesis, the "other" primary event is next to occur. The simulation clock is now "advanced" to the corresponding time. But the effect of the "advance" is null; the new time is identical to the previous clock reading, so simulated time hasn't really changed. Hence, both events are caused to occur at the given reading of the simulated clock.

Average Length of the Waiting Line

The length of the waiting line must be averaged with respect to time. This means that not only the line length, but also the duration of the length, must be considered. For example, consider the numeric situation tabulated below.

Length of Line, Customers	Clock Reading When Line Became That Length	Duration of Length	Product of Length and Its Duration
0	21	4	0
1	25	2	2
2	27	1	2
3	28	11	33
2	39	8	16
1	47	4	4
2	51	---	---
---	---		
Time Spanned (51-21):	30	Total:	57

Average Line Length = 57/30 = 1.90 customers

As the calculation indicates, the average line length for the data shown is 1.90 customers. The average length is not simply the sum of the entries in the first column, divided by the number of entries. That value (11/7 = 1.571+), which is meaningless, fails to take into account how long the line was at its various lengths.

Sampling from the Uniform Distributions

The last point concerns sampling from the uniformly distributed interarrival and service time distributions. Each draw is accomplished by adding to the smallest feasible value a random fraction of the range of the random variable. For example, suppose the service time is uniformly distributed between 8 and 12 minutes. Then, a particular service time can be formed by adding to 8 some random fraction of 12 - 8, or 4. The value returned by the function RAND[1c] can be used directly as the "random fraction" in effect. If a service time is being sampled in this case, and RAND returns a value of 0.3, then the service time is 8 + (0.3)(4), or 9.2 minutes.

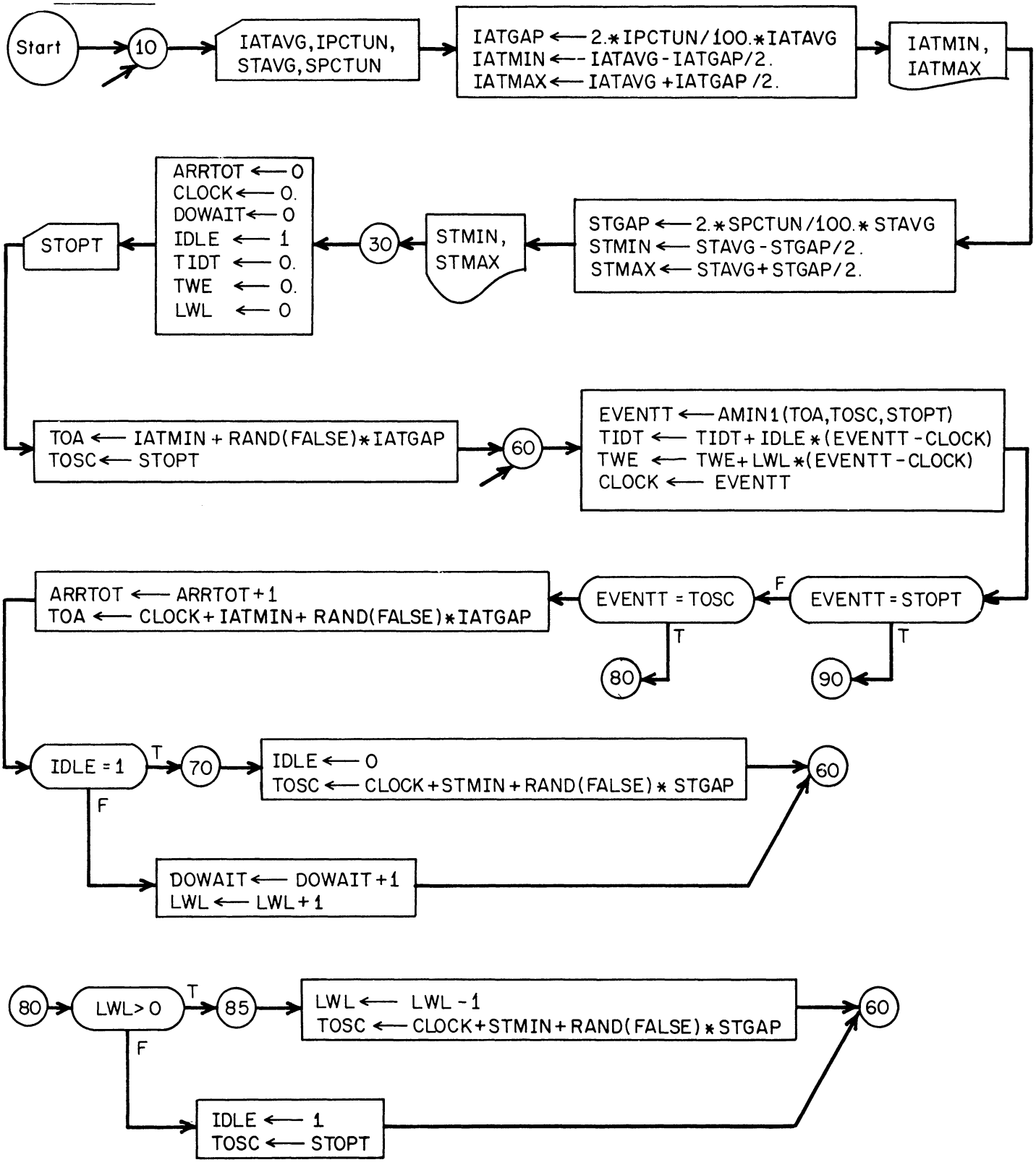
[1c] When called, the function RAND returns a value drawn at random from the population uniformly distributed on the open interval between 0.0 and 1.0. Such a "uniform random number generator" is usually available on most computer systems. The internal makeup of RAND is described in detail as Case Study 8 in FORTRAN Case Studies for Business Applications, by Thomas J. Schriber (John Wiley & Sons, Inc., Publishers; 1969).

1.3 Symbol Table

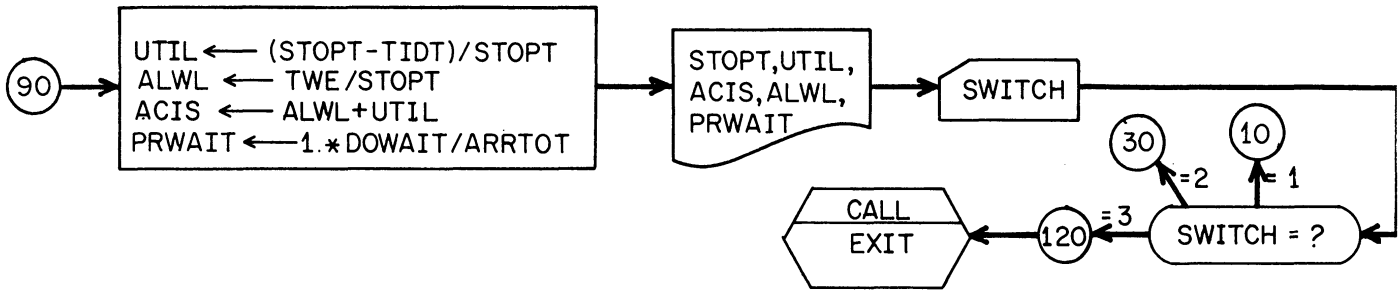
Program Variable	Definition	Mode	Subscript Class
ACIS	Average Customers in System; equal to average length of the waiting line, plus server utilization (i.e., average number of customers at the server)	Real	0
ALWL	Average Length of the Waiting Line	Real	0
AMIN1	An open FORTRAN function returning a copy of the smallest entry in its list of arguments; see any FORTRAN primer for further details	Real	
ARRTOT	Arrival Total; number of customers arriving at the system	Integer	0
CLOCK	Simulated clock	Real	0
DOWAIT	Number of customers who cannot go immediately into service upon arrival	Integer	0
EVENTT	Event Time; time of the next event scheduled to occur	Real	0
IATAVG	Interarrival Time Average; mean time between consecutive customer arrivals	Real	0
IATGAP	Interarrival Time Gap; range of values the interarrival time random variable can assume	Real	0
IATMAX	Interarrival Time Maximum; largest possible time between arrival of consecutive customers	Real	0
IATMIN	Interarrival Time Minimum; smallest possible time between arrival of consecutive customers	Real	0
IDLE	A variable recording the status of the server; when IDLE = 0, the server is engaged; when IDLE = 1, the server is idle	Integer	0
IPCTUN	Interarrival time Percent Uncertainty; percentage variation possible in mean interarrival time	Real	0
LWL	Length of the Waiting Line	Integer	0
PRWAIT	Probability of Waiting; the probability that a customer cannot immediately go into service upon arrival	Real	0
RAND	A Function returning a real value drawn from the population uniformly distributed over the open 0-1 interval; see Footnote 10	Real	
SPCTUN	Service time Percent Uncertainty; percentage variation in mean service time	Real	0
STAVG	Service Time Average; mean service time	Real	0
STGAP	Service Time Gap; range of values the service time random variable can assume	Real	0
STMAX	Service Time Maximum; largest possible value that the service time random variable can assume	Real	0
STMIN	Service Time Minimum; smallest possible value that the service time random variable can assume	Real	0
STOPT	Stop Time; simulated time at which the model is to be shut down	Real	0
SWITCH	A control digit provided as data at the end of the program cycle to signal what the user wants his next option to be	Integer	0
TIDT	Total Idle Time; Total time the server is idle	Real	0
TOA	Time of Arrival; time the next customer is scheduled to arrive at the system	Real	0

Program Variable	Definition	Mode	Subscript Class
TOSC	Time of Service Completion; time the customer now in service will be done with service; equal to STOPT in value if no customer is in service	Real	0
TWE	Total Waiting Experience; accumulated product of waiting line length and duration of length	Real	0
UTIL	Utilization of the server; fraction of the time that the server is engaged	Real	0

1.4 Flowchart(1d)



(1d) The flowcharting conventions used here are those of Elliott I. Organick in A FORTRAN IV Primer (Addison-Wesley; 1966). The USASI flowcharting standards will be used in the final version of the book.



1.5 Program Listing

```
            C     ***THIS PROGRAM SIMULATES A SINGLE-CHANNEL, SINGLE-SERVER QUEUING
            C        SYSTEM.  INTERARRIVAL TIMES OF DEMANDS FOR SERVICE ARE UNIFORMLY
            C        DISTRIBUTED.  SERVICE TIMES ARE ALSO UNIFORMLY DISTRIBUTED.
            C        SERVER UTILIZATION, AVERAGE LENGTH OF THE WAITING LINE, AVERAGE
            C        NUMBER OF CUSTOMERS IN THE SYSTEM, AND THE PROBABILITY THAT AN
            C        ARRIVAL WILL HAVE TO WAIT FOR SERVICE ARE ESTIMATED BY THE
            C        MODEL.  THE ESTIMATES ARE BASED ON A SIMULATION OF DURATION
            C        SPECIFIED AS DATA***
            C
            C     ***PROVIDE MODE INFORMATION***
0001                IMPLICIT REAL (A-Z)
0002                INTEGER ARRTOT,DOWAIT,IDLE,LWL,SWITCH
            C
            C     ***INPUT AVERAGE INTERARRIVAL TIME ('IATAVG'), THE PERCENT UNCERTAINTY
            C        IN THIS FIGURE ('IPCTUN'), AVERAGE SERVICE TIME ('STAVG'), AND
            C        THE PERCENT UNCERTAINTY IN THIS FIGURE ('SPCTUN')***
0003         10     READ(5,20) IATAVG,IPCTUN,STAVG,SPCTUN
0004         20     FORMAT(4F10.0)
            C
            C     ***COMPUTE WIDTH OF INTERARRIVAL TIME INTERVAL, AND THE MINIMUM AND
            C        MAXIMUM FEASIBLE VALUES OF INTERARRIVAL TIME.  OUTPUT THE RANGE
            C        OF FEASIBLE VALUES***
0005                IATGAP = 2.*IPCTUN/100.*IATAVG
0006                IATMIN = IATAVG - IATGAP/2.
0007                IATMAX = IATAVG + IATGAP/2.
0008                WRITE(6,25) IATMIN,IATMAX
0009         25     FORMAT('1INTERARRIVAL TIMES ARE UNIFORMLY DISTRIBUTED BETWEEN '
                   1 F6.2,' AND 'F6.2,' MINUTES.')
            C
            C     ***COMPUTE WIDTH OF THE SERVICE TIME INTERVAL, AND THE MINIMUM AND
            C        MAXIMUM FEASIBLE VALUES OF SERVICE TIME.  OUTPUT THE RANGE
            C        OF FEASIBLE VALUES***
0010                STGAP = 2.*SPCTUN/100.*STAVG
0011                STMIN = STAVG - STGAP/2.
0012                STMAX = STAVG + STGAP/2.
0013                WRITE(6,28) STMIN,STMAX
0014         28     FORMAT('0SERVICE TIMES ARE UNIFORMLY DISTRIBUTED BETWEEN '
                   1 F6.2,' AND 'F6.2,' MINUTES.')
            C
            C     ***INITIALIZE PERTINENT VALUES BEFORE STARTING THE SIMULATION***
0015         30     ARRTOT = 0
0016                CLOCK = 0.
0017                DOWAIT = 0
0018                IDLE = 1
0019                TIDT = 0.
0020                TWE = 0.
0021                LWL = 0
            C
            C     ***INPUT THE NUMBER OF SIMULATED TIME UNITS (MINUTES) FOR WHICH
            C        THE MODEL IS TO BE RUN***
0022                READ(5,50) STOPT
0023         50     FORMAT(F10.0)
            C
            C     ***COMPUTE THE TIME OF ARRIVAL TO THE SYSTEM OF THE FIRST
            C        SERVICE DEMAND***
0024                TOA = IATMIN + RAND(FALSE)*IATGAP
```

```
      C     ***BECAUSE THERE IS NOT NOW ANYONE "IN SERVICE", SET THE TIME
      C        OF NEXT SERVICE COMPLETION TO AN ARBITRARILY LARGE VALUE.  A
      C        CONVENIENT CHOICE OF AN "ARBITRARILY LARGE VALUE" IS THE TIME
      C        THE MODEL IS SCHEDULED TO SHUT OFF***
0025          TOSC = STOPT
      C
      C     ***SET TIME OF NEXT EVENT TO THE MINIMUM OF TIME OF ARRIVAL,
      C        TIME OF SERVICE COMPLETION, AND TIME OF STOPPING THE SIMULATION***
0026       60 EVENTT = AMIN1(TOA,TOSC,STOPT)
      C
      C     ***UPDATE TOTAL IDLE TIME OF THE SERVER AND TOTAL WAITING
      C        EXPERIENCE OF THOSE WAITING FOR SERVICE.  THEN ADVANCE THE CLOCK
      C        TO THE TIME OF THE NEXT EVENT***
0027          TIDT = TIDT + IDLE*(EVENTT-CLOCK)
0028          TWE = TWE + LWL*(EVENTT-CLOCK)
0029          CLOCK = EVENTT
      C
      C     ***GO TO 90 IF THE NEXT EVENT IS TO STOP THE SIMULATION;  OR GO
      C        TO 80 IF THE NEXT EVENT IS A SERVICE COMPLETION***
0030          IF (EVENTT .EQ. STOPT) GO TO 90
0031          IF (EVENTT .EQ. TOSC) GO TO 80
      C
      C     ***IF NEITHER TRANSFER WAS TAKEN, THE NEXT EVENT IS ARRIVAL OF
      C        A DEMAND FOR SERVICE.  UPDATE THE TOTAL ARRIVALS TO DATE,
      C        THEN PRE-SCHEDULE THE TIME WHEN THE ARRIVAL'S SUCCESSOR WILL ARRIVE***
0032          ARRTOT = ARRTOT + 1
0033          TOA = CLOCK + IATMIN + RAND(FALSE)*IATGAP
      C
      C     ***TRANSFER TO 70 IF THE SERVER IS IDLE;  OTHERWISE, UPDATE BOTH THE
      C        NUMBER OF THOSE WHO HAVE HAD TO WAIT FOR SERVICE, AND THE NUMBER
      C        OF THOSE WHO ARE NOW WAITING.  THEN RETURN TO DETERMINE
      C        THE NEXT EVENT***
0034          IF (IDLE .EQ. 1) GO TO 70
0035          DOWAIT = DOWAIT + 1
0036          LWL = LWL + 1
0037          GO TO 60
      C
      C     ***UPDATE THE STATUS OF THE SERVER, PRE-SCHEDULE THE TIME OF SERVICE
      C        COMPLETION, AND THEN RETURN TO DETERMINE THE NEXT EVENT***
0038       70 IDLE = 0
0039          TOSC = CLOCK + STMIN + RAND(FALSE)*STGAP
0040          GO TO 60
      C
      C     ***A SERVICE COMPLETION IS THE EVENT OCCURRING.  TRANSFER TO 85 IF
      C        NO ONE IS WAITING FOR THE SERVER WHO HAS JUST BECOME AVAILABLE***
0041       80 IF (LWL .GT. 0) GO TO 85
      C
      C     ***WHEN NO ONE IS WAITING, RECORD THAT THE SERVER IS IDLE, AND SET
      C        THE TIME OF NEXT SERVICE COMPLETION TO AN ARBITRARILY LARGE VALUE.
      C        THEN RETURN TO DETERMINE THE NEXT EVENT***
0042          IDLE = 1
0043          TOSC = STOPT
0044          GO TO 60
      C
      C     ***REMOVE A WAITING SERVICE DEMAND FROM THE WAITING LINE, PRE-SCHEDULE
      C        THE TIME OF SERVICE COMPLETION, THEN RETURN TO DETERMINE THE
      C        NEXT EVENT***
0045       85 LWL = LWL - 1
0046          TOSC = CLOCK + STMIN + RAND(FALSE)*STGAP
0047          GO TO 60
```

```
            C     ***THE END OF THE SIMULATION HAS BEEN REACHED.  COMPUTE PERTINENT
            C        STATISTICS, THEN OUTPUT THE SYSTEM PERFORMANCE INFORMATION***
0048         90   UTIL = (STOPT-TIDT)/STOPT
0049              ALWL = TWE/STOPT
0050              ACIS = ALWL + UTIL
0051              PRWAIT = 1.*DOWAIT/ARRTOT
0052              WRITE(6,100) STOPT,UTIL,ACIS,ALWL,PRWAIT
0053        100   FORMAT('4DURATION OF SIMULATION, MINUTES:' F16.2/
                  1       ' FRACTIONAL UTILIZATION OF SERVER:' F16.3/
                  2       ' AVERAGE NUMBER OF CUSTOMERS IN SYSTEM:' F11.3/
                  3       ' AVERAGE LENGTH OF THE WAITING LINE:' F14.3/
                  4       ' PROBABILITY OF HAVING TO WAIT FOR SERVICE:' F7.3)
            C
            C     ***INPUT A SWITCH DIGIT TO DETERMINE WHETHER TO (1) INPUT NEW
            C        INTERARRIVAL AND SERVICE TIME DISTRIBUTIONS, OR (2) CARRY OUT
            C        ANOTHER SIMULATION WITH THE CURRENT DISTRIBUTIONS, OR
            C        (3) TERMINATE EXECUTION***
0054              READ(5,110) SWITCH
0055        110   FORMAT(I1)
0056              GO TO (10,30,120), SWITCH
0057        120   CALL EXIT
0058              END
```

1.6 Data Format

Card Number	Card Columns	Content	Mode
1	1-10	IATAVG	Real
	11-20	IPCTUN	Real
	21-30	STAVG	Real
	31-40	SPCTUN	Real
2	1-10	STOPT	Real
3	1	SWITCH	Integer

1.7 Data Used (Data are shown only for "Data Situation 1", and the beginning of "Data Situation 2", as described in "Statement of the Problem"

```
10.       0.0       10.       0.0
480.
1
10.       40.       9.5       40.
2
480.
2
960.
2
1440.
```

1.8 Program Output

```
INTERARRIVAL TIMES ARE UNIFORMLY DISTRIBUTED BETWEEN  10.00 AND  10.00 MINUTES.

SERVICE TIMES ARE UNIFORMLY DISTRIBUTED BETWEEN  10.00 AND  10.00 MINUTES.


DURATION OF SIMULATION, MINUTES:          480.00
FRACTIONAL UTILIZATION OF SERVER:           0.979
AVERAGE NUMBER OF CUSTOMERS IN SYSTEM:      0.979
AVERAGE LENGTH OF THE WAITING LINE:         0.0
PROBABILITY OF HAVING TO WAIT FOR SERVICE:  0.0
```

Output for Data Situation 1 (See "Statement of the Problem")

Span of Simulation, Minutes	Server Utilization
480	0.832
960	0.916
1440	0.944
1920	0.955
2400	0.945
2880	0.959
3360	0.956

Tabulated Output for Data Situation 2 (See "Statement of the Problem")

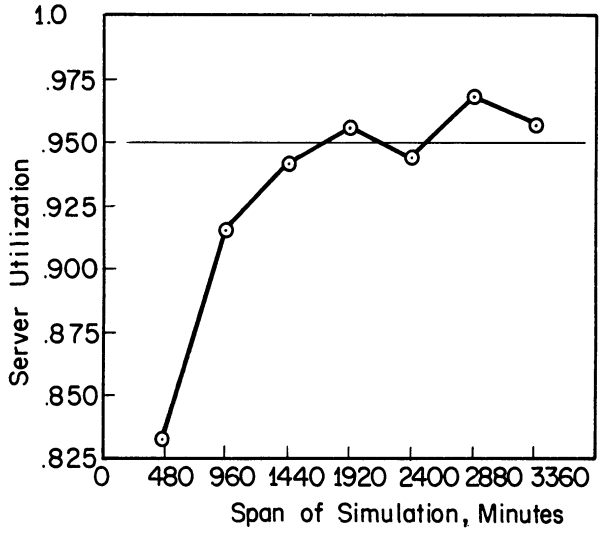


Figure B.1 Graph of Output for Data Situation 2

		Percent Variation Possible in Interarrival Time			
		10	20	30	40
Percent Variation Possible in Service Time	10	.385	.546	.666	.719
	20	.574	.625	.707	.756
	30	.668	.697	.726	.784
	40	.769	.762	.779	.760

Tabulated Output for Data Situation 3 (See "Statement of the Problem")

Table Entries are the Observed Probabilities of Having to Wait for Service;
Table Parameters: STAVG = 9.5; IATAVG = 10.0; STOPT = 48000.

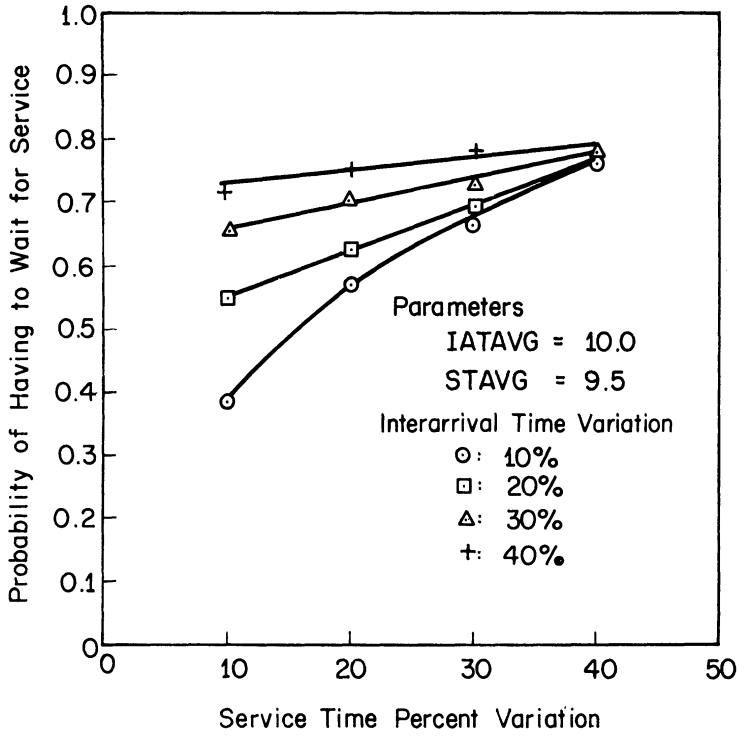


Figure B.2 Graph of Output for Data Situation 3

1.9 Observations

Note that the simulation clock (CLOCK) is a real variable. This means the equivalent of about 7 decimal digits are carried in the record of "time."(1e) For the runs made, the time unit used in expressing data is the minute. Hence, an arrival might be scheduled to occur when the clock reads, say, 54.34752 minutes. In many special-purpose simulation languages (such as GPSS), the "clock" is integer-valued. When this is the case, no fractions of the time unit used to express the data can be recorded by the clock. Of course, time can be as "finely divided" as is required for realism in the system being modeled, because the time unit used to express the data can be made as small as necessary.

Also note that the model is a "next event" simulator. That is, when the clock is changed, its reading is advanced directly to the time when the next event is scheduled to occur. Intermediate points in time, when nothing will happen in the system anyway, are jumped over. Next-event simulators are sometimes designated "variable time increment" simulators. This contrasts with approaches to simulation which use a fixed time increment. In such cases, when the clock is changed, it is always advanced by a fixed amount. Testing is then conducted on the system to determine if anything is to occur at the clock's new reading. For further discussion of the two approaches and references on their relative advantages and disadvantages, see Computer Simulation Techniques, by Thomas H. Naylor, Joseph L. Balintfy, Donald S. Burdick, and Kong Chu (John Wiley & Sons, Inc., 1966), pp. 126-127.

(1e) Based on the assumption that single precision is used with a System/360 computer.

Program Listing

Note how the time of service completion is set equal to the time of model shutdown on those occasions when the server is idle. See Statements 0025 and 0043 in the Program Listing.

Statement 0026 tests to determine the time of the next event in the system. The FORTRAN open function AMIN1 is used for this testing. The function scans the list of arguments to see which is smallest, then returns a copy of the smallest list entry as its value. The copy is saved by the model as EVENTT. At Statements 0027 and 0028, updating of TIDT and TWE is then accomplished before the clock is advanced to its new value. The updating requires use of the "time span" about to be eclipsed. This time span is simply EVENTT minus CLOCK. After the updating, CLOCK is moved forward to EVENTT. Then CLOCK is examined at Statements 0030 and 0031 to determine which event must be caused to occur.

Program Output

One of the disadvantages of simulation is that it produces only estimates of statistical properties, not their true values. This observation is reinforced in examining the program output. In Figure B.1, the fluctuation of server utilization about its known expected value of 0.95 is evident. In Figure B.2, rather than plotting fluctuations, point estimates of probability values are shown. If the true values of these probabilities were known and used, the plotted points and their curves would coincide. Due to error in the estimates, however, the smooth curves do not pass exactly through the plotted points.

Model Extension

Consider extending the model so that it represents a one-line, multiple-server queueing system. These considerations are involved in such an extension.

(a) Rather than recording a single "time of service completion," a vector of such values must, in general, be maintained. Each customer in service has a time of service completion associated with him. In scanning the future events list to find the next event, all of these pre-scheduled service completions must be taken into account.

(b) Rather than recording whether "the" server is idle or busy, a method must be adopted for recording how many servers are idle, and how many are busy.

(c) An assumption must be made as to whether the multiple servers have a common service time distribution or not. If all service times follow the same distribution, the task in (b) above is not difficult because there is no need to maintain individual identity for each server. Individual identities must be taken into account, however, if the service time distribution depends on the particular server in question.

Chapter 2

Basic GPSS Modeling Concepts

2.1 Some Preliminary Considerations

GPSS (General Purpose Simulation System) is both a language and a computer program. As a language, it has a well-defined vocabulary and grammar with which certain types of system models can be unambiguously described. As a computer program, it interprets a model described in the GPSS language, thereby making it possible to conduct experiments with the model on a computer. Without such interpretation, the computer would not be able to directly act out, or simulate, the system represented by the model. The computer program which performs this interpretation will be referred to as the "GPSS Processor," or simply as the "Processor."

There are several versions of GPSS. This is a result of historical developments dating back to the early 1960's. Some of the earlier versions were named GPSS, GPSS II, and GPSS III. The implementation now used most frequently is GPSS/360, so named because it can be used with computers in the International Business Machine's "System/360" family. Even for GPSS/360, a distinction is made between Version I and Version II. And, just over a year ago, a GPSS V was announced by IBM.

GPSS/360, Version I, is available without charge to users of IBM computer equipment. Version II, and GPSS V, can be rented for fees of approximately $20 and $55 per month, respectively. There is upward compatability among Versions I and II, and GPSS V. Furthermore, Version II and GPSS V are not major extensions of Version I. As a consequence, only GPSS/360, Version I, will be covered in this book. The several additional features in the newer implementations are easily acquired by the person familiar with Version I. A brief comparison of Version I with the later implementations appears in Appendix F.

Computer manufacturers other than IBM have, in some instances, taken steps to make GPSS available to users of their hardware. For example, a form of GPSS II can be used on certain General Electric computers. Control Data Corporation offers GPSS for its 6000 series computers. And, a form of GPSS/360 has been implemented on the Univac 1108 by University Computing Corporation. For these various versions, corresponding literature is available from the companies which undertook the implementations. IBM literature which is especially pertinent to the GPSS/360, Version I, covered in this book is (1) GPSS/360: Introductory User's Manual [Form Number GH20-0204], and (2) GPSS/360: User's Manual [Form Number GH20-0311]. Appendix A contains a listing of additional IBM literature dealing with GPSS. Such literature can usually be obtained through IBM sales offices throughout the country.

2.2 Approach to Model-Building in GPSS

A GPSS model of a system may be expressed either as a Block Diagram, or as the punchcard equivalent of a Block Diagram. The model builder usually begins his work by constructing a Block Diagram of the system he intends to simulate. The punchcard version of the Block Diagram is then prepared and presented to the computer for implementation. Model conceptualization most often takes place at the Block Diagram level. After a Block Diagram has been developed, the process of producing its equivalent as a deck of punched cards is then straightforward and mechanical.

A Block Diagram is a collection of characteristically-shaped figures (Blocks) connected by directed line segments. There is a set of some 44 Blocks which GPSS makes

available to the model builder. The shapes of these Blocks are pre-defined. The distinction among shapes eases the process of becoming familiar with a model by studying its Block Diagram. The various shapes have no significance, as such, in the punchcard version of the model.

Models are built by selecting certain Blocks from the available set and arranging them in a diagram so that, at the time of model implementation, they (that is, their images) interact meaningfully with one another. The logical requirements of the system being modeled dictate which Blocks are used in constructing the model. When the model is implemented, it is the interaction among the Blocks which is analogous to (simulates) the interaction of elements in the real system being modeled.

The silhouette of a typical GPSS Block Diagram is shown in Figure 2.1. Ten different Blocks can be identified in the figure. Several of them appear more than one time. Detailed information that would typically appear in the Blocks has been deliberately deleted. The intent here is to provide a first glimpse of a Block Diagram for the sake of perspective.

The properties of a basic subset of Blocks will be taken up in the rest of this chapter. The subset is chosen to make it possible to build complete, although relatively simple, GPSS models of systems. In succeeding chapters of the book, additional Blocks will be studied, and their use will be illustrated in context. As the model-builder's "Block vocabulary" grows, he can build models of increasingly complicated systems. When the entire set of Blocks has been mastered, rather sophisticated models can be built in GPSS with relative ease.

2.3 Transactions: Dynamic Entities in GPSS Models

Lines with arrowheads were used in the Chapter 1 flowcharts to represent a time ordered series of steps to be followed in performing procedures. Expressing this differently, it can be said that "control moves from box to box" (or, from instruction to instruction) in a flowchart of the Chapter 1 variety. The directed line segments in the Figure 2.1 Block Diagram also suggest movement, or flow. In GPSS, however, the concept of "control moving from Block to Block" is not entirely valid and must be discarded. The directed line segments in a GPSS Block Diagram represent paths of flow along which units of traffic move. Each unit of traffic is termed a Transaction. Transactions, then, are dynamic (i.e., moving) entities in a GPSS model. Their movement from Block to Block takes place as execution of a GPSS model proceeds.
Block Diagram represent paths of flow along which units of traffic move. Each unit of traffic is termed a Transaction. Transactions, then, are dynamic (i.e., moving) entities in a GPSS model. Their movement from Block to Block takes place as execution of a GPSS model proceeds.

When a simulation first begins, no Transactions exist in a GPSS model. As the simulation proceeds, Transactions enter the model at certain times, according to the logical requirements of the system being modeled. Similarly, Transactions leave the model at certain times during the course of a simulation. In general, then, there are many Transactions in a model. Nevertheless, only one of these Transactions moves forward at a time.

After it is set into motion, a Transaction moves from Block to Block along whichever path it is in. Each Block can be thought of as a point at which a subroutine can be called. When a Transaction "enters" a Block, the corresponding subroutine is executed. This forward motion is continued until any one of three possible circumstances arises:

(1) The Transaction moves into a Block whose purpose is to hold it there for a prescribed length of time.

(2) The Transaction moves into a Block whose purpose is to remove it from the model.

(3) The Transaction attempts to move into the next Block in its path, but that Block refuses to let it enter. In this case, the Transaction is held at the Block preceding the one refusing entry. Later, it will repeat its

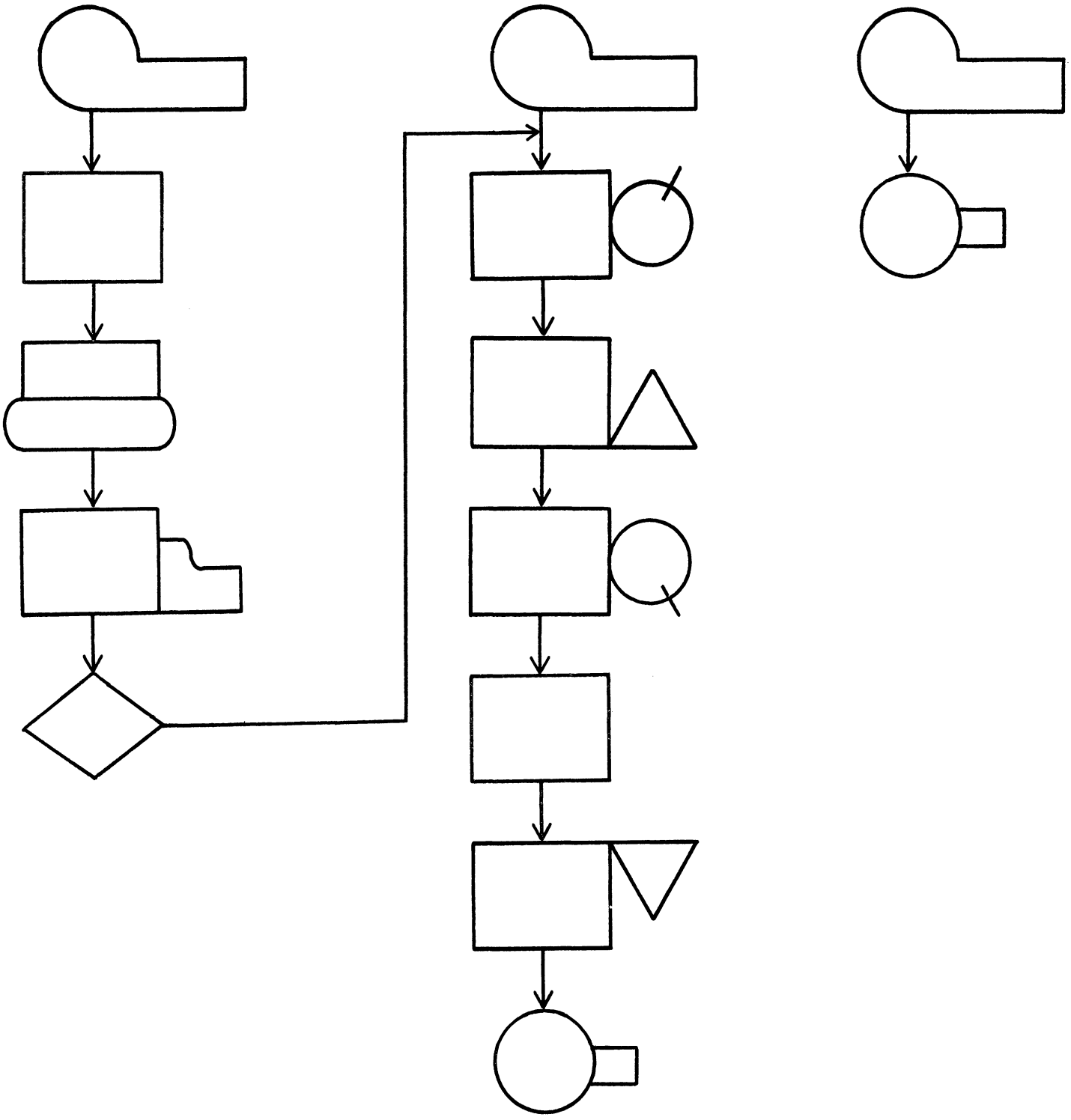


Figure 2.1 The Silhouette of a Typical GPSS Block Diagram

attempt to gain entry to the next Block. As model conditions change, one of the subsequent attempts will, in general, be successful. The Transaction can then continue moving along its path of flow.

After a Transaction comes to rest for one of the three reasons explained, the forward motion of another Transaction in the model is initiated.[2a] Eventually it, too, comes to rest, and the forward motion of yet another Transaction begins. It is in this fashion that execution of a GPSS model proceeds. Recall that each successful entry of a Transaction into a Block results in a call on a subroutine. Model execution, then, consists of a series of subroutine calls which result from movement of Transactions.

Given this much information, certain inferences can now be drawn from Figure 2.1. For example, two Blocks in the figure are "sinks" in the sense that the path of flow is not extended beyong them. These two Blocks, identical in shape, must have the purpose of removing from the model Transactions which flow into them. Similarly, there are two Blocks in the figure which are "sources" in the sense that no path of flow leads into them. These Blocks must represent points at which Transactions enter a model. Finally, it can be noted that Figure 2.1 consists of two separate, free-standing segments. In general, "a" GPSS model can consist of many free-standing segments. As a model is executed, "activity" occurs in the segment containing the currently-moving Transaction. When it comes to rest, the next Transaction set into motion may happen to be in a different model segment, resulting in a switch of the "action" to that segment. The concept of free-standing Block Diagram segments, then, is entirely valid in GPSS.

Until now, the discussion of Transactions has been completely abstract. An attempt has been made to answer the question, "What is a Transaction?", but there has been nothing said to answer the parallel question, "What meaning do Transactions have?" The "meaning" of Transactions is determined by the model builder. This is done by establishing an analogy, or correspondence, between Transactions and elements of the real-world system being modeled. These analogies are never "declared" to the GPSS Processor. They exist only in the mind of the analyst who builds the model. Of course, each model must be built in a way consistent with the analogies the analyst has in mind.

Some examples of possible analogies between Transactions and elements of real systems appear in Table 2.1. In a model of a barber shop, for example, a Transaction might represent a customer. In the real system, the customer arrives at the shop, joins a waiting line, waits until his turn comes, then uses the services of the barber and leaves. The customer can clearly be thought of as a "unit of traffic" moving through the barber shop. In a GPSS model of the shop, various Blocks must be used to represent such occurrences as a customer's arrival (entry of a Transaction), a customer's use of the barber, a customer's departure (removal of a Transaction), and so on. Movement of a Transaction from Block to Block in the model is then analogous to movement of a customer from stage to stage in the barber shop.

These analogies between Transactions and real-system elements will be brought into sharper focus as properties of the GPSS language are unfolded.

[2a] The order in which Transactions are moved will be explained in detail later.

System	System Element Represented by a Transaction
Supermarket	Shopper
Highway	Car
Maintenance Shop	Part
Inventory Control	Demand
Barber Shop	Customer

Table 2.1 Examples of Possible Interpretations for Transactions

2.4 The Simulation Clock

Time passes as various events occur in real systems. A customer arrives at a barber shop. Later, his turn comes and the barber begins to cut his hair. Still later, the haircut is finished and the customer leaves. If such events are to be represented in a simulation model, they must occur against a background of simulated time. As a consequence, the GPSS Processor automatically maintains a clock to record "what time it is."

When a simulation begins, the Processor first pre-schedules Transaction arrivals to the extent possible. The simulated clock is then set to the earliest time that a Transaction is to enter the model. This Transaction (and others, if any, which are to enter at the same time) is then brought into the model. Then, it (or they, one by one) is moved through as many Blocks as possible. Eventually, there is nothing else which is to occur in the system at this first clock reading. The GPSS Processor then advances the clock to the time when the next event, or series of events, is scheduled to take place. These events, as represented by Transaction movement through Blocks, are then caused to occur. When there are no more Transaction movements left to perform at this second clock reading, the clock is again advanced, and so on. It is in this manner that the passage of time is simulated.

A numeric example will help explain the process just described in general terms. In a one-man barber shop, suppose the first few events on a given day are as shown in Table 2.2. As modeled in GPSS, the Processor would maintain the clock as follows. The shop is opened at the start of the simulation. Nothing is to occur until time 22.[2b] The clock is therefore set to 22, and Customer 1 arrives (a Transaction enters the model). Finding he does not have to wait, the customer immediately gets into the barber's chair (the Transaction flows forward, Block by Block, until it moves into a Block which deliberately holds it to simulate hair-cutting time). No further occurrences are to take place until time 29. The clock is consequently advanced to 29, and Customer 2 arrives (another Transaction enters the model). This customer has to wait for the barber (the Transaction is denied permission to move into the Block which simulates the act of engaging the barber). Nothing else is to happen at time 29. The Processor then moves the clock to 33, causes Customer 3 to arrive (yet another Transaction enters the model), and so on.

It should be clear by now how intimately the reading of the simulated clock is related to the sequence of events which can, in general, occur in a simulation model. One of the advantages of GPSS is that it automatically updates the simulation clock as required by the logic described in a model. There is no need for the analyst to explicitly arrange for this clock maintenance, as was the case in the Chapter 1 example of a one-man barber shop. The precise manner in which the GPSS Processor updates the clock will be explained in due course.

[2b] In the one-man barber shop model in Chapter 1, the concept of pre-scheduling events was explained. It is this pre-scheduling which makes it possible to "know" at what future time an event will occur. The precise way in which the GPSS Processor pre-schedules future events will be explained later.

Sequence in Which the Event Occurs	The Event Itself	Real Time of Occurrence	Simulated Time of Occurrence
1	The barber opens the shop	8:00 a.m.	0
2	Customer 1 arrives and goes into service	8:22 a.m.	22
3	Customer 2 arrives	8:29 a.m.	29
4	Customer 3 arrives	8:33 a.m.	33
5	Customer 1 is finished	8:47 a.m.	47
6	Customer 2 goes into service	8:47 a.m.	47
7	Customer 4 arrives	9:07 a.m.	67

Table 2.2 A Possible Sequence of Events in a One-Man Barber Shop

There are several important features of GPSS and the GPSS clock which will now be stated, point by point.

(1) The GPSS clock registers only integer values. This means that events can only occur at "whole" time values in GPSS models.

(2) The unit of time which the clock registers is determined by the analyst. However, the time unit chosen is never "declared" to the Processor. It is expressed implicitly in terms of the time data built into the model. If all time data are expressed in minutes, then the minute is the implicit unit of time. Or, if all time data are in milliseconds, then the unit of time is the millisecond. The analyst is responsible for deciding the smallest time unit required to realistically reflect real-system events in his model. He must then take care to express all his time data in terms of this smallest unit.

In the Table 2.2 barber shop example, the implicit time unit is 1 minute. This means it would not be possible for a customer to arrive at, say, the 47th second after 8:51 a.m. If it is necessary to have arrivals occur "to the nearest second" in the model, then the implicit time unit cannot be larger than 1 second.

(3) GPSS is a "next event" simulator. That is, after a model has been fully updated at a given point in simulated time, the clock is advanced to the nearest time at which one or more next events are scheduled to occur. Potential clock readings are "jumped over" when no events are to take place at those times. This means that execution time requirements are independent of the implicit time unit chosen by the analyst.

Finally, care should be taken to distinguish between "simulated time" and "real time." When the simulation clock is advanced to a next reading, that reading "remains constant" while the model is updated. Nevertheless, real time passes as the updating occurs. It may require hours of real time to move models of some systems (e.g., computer systems) through only minutes of simulated time. On the other hand, experiments equivalent to weeks, months, or even years of simulated time can often be conducted in only seconds of real time on a computer. This ability to "compress time" is one of the potential advantages of experimenting with systems by simulating them on a computer.

2.5 General Details Associated with Blocks in a GPSS Block Diagram

The Figure 2.1 Block Diagram silhouette is repeated in Figure 2.2, where the general details associated with each of the Blocks have been filled in. Each Block carries with it information falling into three categories.

(1) Location

Each Block occupies a specific Location in a Block Diagram. Strictly speaking, Locations are designated numerically. The first Block in a model occupies Location 1, the second Block occupies Location 2, and so on. Fortunately, it is not necessary for the model builder to provide Location Numbers. When the GPSS Processor inputs the punched-card version

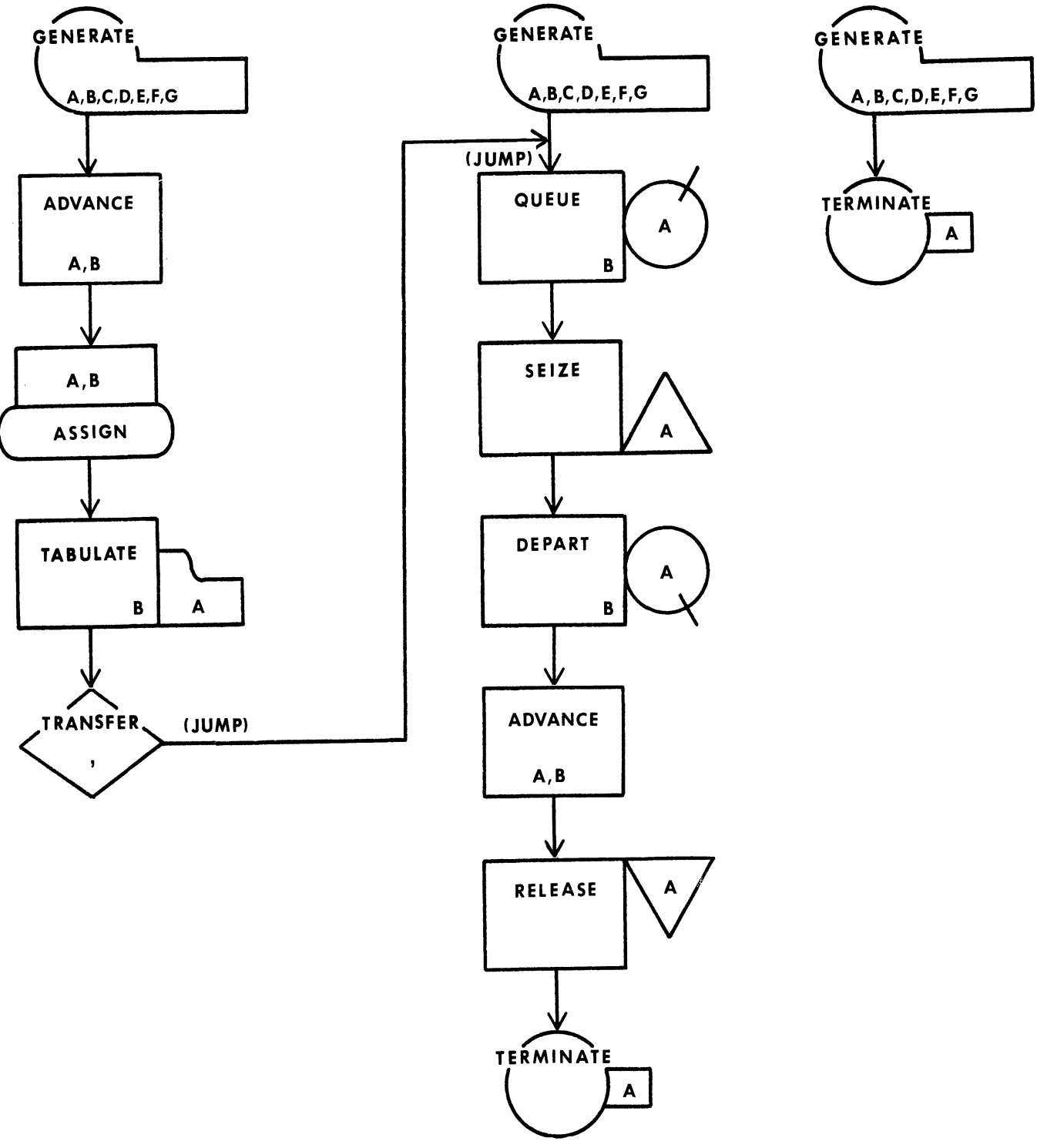


Figure 2.2 A Repetition of Figure 1.1, with General Block Details Shown

of a model, it assigns the Location Numbers in the order in which the cards for the various Blocks have been placed in the card deck.

It frequently happens that the analyst wants to know which Locations are occupied by certain Blocks in a model. This information may be required so the Blocks in question can be referenced from one or more other Blocks. When this need arises, it would be tedious to "count Blocks" to predict the Location Numbers ultimately to be assigned by the GPSS Processor. Rather than counting, the analyst has the option of providing symbolic Location Names for the Blocks of interest. When such symbolic names are used, the Processor later supplies absolute Location Numbers in their place.

Symbolic names are composed of from 3 to 5 alphanumeric characters, with the restriction that the first 3 be alphabetic. Examples of valid and invalid symbolic location names are shown in Table 2.3

Valid	Invalid
BYPAS	BY25
BLOK1	2AND4
OUT	NO
JOE23	A2B
FLO2M	P

Table 2.3 Examples of Valid and Invalid Names for Block Locations

Normally, the analyst only assigns a symbolic name to the Location occupied by a Block when it is logically necessary to refer to that Block from other parts of the model. In Figure 2.2, for example, the Location occupied by the "QUEUE" Block has been symbolically named JUMP. The Location Name is written just above the QUEUE Block, and is enclosed in parentheses. (The parentheses are not part of the name.) Note how the Location JUMP is referenced at another Block in the model (i.e., at the "TRANSFER" Block). The name JUMP is supplied at the TRANSFER Block to indicate where each Transaction entering the TRANSFER Block is to go next.

Of course, all Location Names in a model must be unique. Otherwise, ambiguities would arise.

(2) Operation

The "Operation" of a Block is a "verb" suggestive of the task the Block accomplishes. Each Block type is characterized by its own, pre-defined verb. The operations GENERATE, QUEUE, SEIZE, DEPART, ADVANCE, RELEASE, TERMINATE, ASSIGN, TABULATE, AND TRANSFER all appear in Figure 2.2. When the punchcard version of a Block Diagram is prepared, each Block Operation must be provided in its entirety. No abbreviations are permitted.

(3) Operands

The various Blocks have Operands associated with them. A Block's Operands provide the specific information on which the Block's action is based. The Operands may be conveniently thought of as the arguments used in calls on subroutines.

The number of Operands each Block has depends on the type of Block. No Block uses more than 7 Operands. Most use only 1 or 2. The Operands are represented in general as A, B, C, D, E, F, and G. The Operands are shown only in this general fashion in Figure 2.2.

For some Blocks, certain Operands must always be specified, whereas others are optional. In some cases, when optional Operands are not explicitly provided, the Processor assumes "default values" to be in effect.

2.6 Bringing Transactions into a Model: The GENERATE Block

The primary method of bringing Transactions into a model is to use one or more GENERATE Blocks. The GENERATE Block can be thought of as a "door" through which Transactions enter a model. It is usually the analyst's intention to have Transactions come into a model at different points in time. The time between two consecutive arrivals is termed "inter-arrival time." The inter-arrival time concept discussed in the Chapter 1 queuing system case is applicable here. In fact, the approach used in GPSS to arrange for Transaction arrivals is identical to the one taken in arranging for customer arrivals in the Chapter 1 case. That is, when a Transaction enters a model through a GENERATE Block, the Processor pre-determines its successor's time of arrival by sampling from an inter-arrival time distribution, then adding the sampled value to the current clock reading. When that future time is reached, another Transaction is brought into the model through that GENERATE Block, and so on.

Recall that, in the Chapter 1 queuing model, it was necessary for the analyst to supply the logic necessary to support this pre-scheduling procedure. In GPSS, the Processor conducts the various required steps automatically, as part of the operation of the GENERATE Block. The analyst is consequently relieved of a number of otherwise burdensome details.

In fact, almost all the analyst does in using a GENERATE Block is supply the specifications for the inter-arrival-time distribution. The required information is expressed through the Block's A and B Operands. In GPSS, all possible inter-arrival time distributions are divided into two categories:

(1) Uniformly distributed inter-arrival times.

(2) All other inter-arrival distributions.

In short, a "special case" is made of what is perhaps the simplest of all non-trivial distributions, the uniform. To express more complicated (and realistic) inter-arrival time distributions in GPSS, the analyst must resort to Function definition. The way Functions are defined and used at GENERATE Blocks will be taken up in Chapter 3. In this chapter, only uniformly-distributed inter-arrival times will be considered.

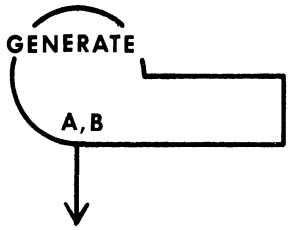
The GENERATE Block, with its A and B Operands in their usual positions, is shown in Figure 2.3. As indicated, the A Operand specifies the average time between consecutive arrivals of Transactions at the GENERATE Block. The B Operand provides the half-width of the range over which the inter-arrival time is understood to be uniformly distributed. When the Operands are supplied as constants, they must be integers.[2c] In fact, there is only one Block in the entire GPSS Block vocabulary for which a decimal point can be included as part of an Operand. In all other circumstances, it is an error to include decimals when expressing Block Operands.

Figure 2.4 provides a specific example of the GENERATE Block. The A and B Operands are 5 and 3, respectively. The inter-arrival time, then, is uniformly distributed over the range 5±3, i.e., over the integers 2, 3, 4, 5, 6, 7, and 8. Recall that the GPSS clock only registers integer values. For this reason, Transactions can only be brought

---

[2c] The possibility of using Operands other than constants will be considered beginning in Chapter 3.

into a model at integer-valued points in time. This explains why 5±3 describes the closed interval of integers from 2 to 8, rather than the continuum of all values between 2 and 8. In this example, then, inter-arrival time can take on any one of 7 different values. Because the values are uniformly distributed, each occurs with a relative frequency of 1/7th.



Operand	Significance	Default Value
A	Average Inter-Arrival Time	Zero
B	Half-Width of Range Over Which Inter-Arrival Time is Uniformly Distributed	Zero

Figure 2.3 The GENERATE Block and its A and B Operands

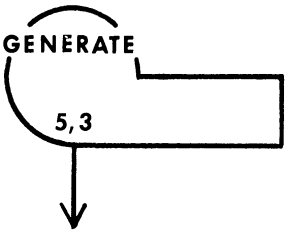


Figure 2.4 A GENERATE Block with Specific A and B Operand Values

To illustrate pre-scheduling, suppose a Transaction arrives at the Figure 2.4 GENERATE Block at simulated time 15. After this Transaction has moved to the next Block in the model, the GPSS Processor draws a sample from the 5±3 inter-arrival time distribution. The sampled value is, say, 7. The Processor then schedules arrival of the next Transaction at that GENERATE Block at future time 15 + 7, or 22. When that Transaction appears at the GENERATE Block and moves to the next Block, the time of its successor's arrival will then have to be pre-scheduled. In short, a "boot-strapping" technique is used to arrange for Transaction arrivals. Again, note the similarity between the approach used here, and that used to pre-schedule customer arrivals in the Chapter 1 one-man barber shop example.

As suggested in Figure 2.3, values for the A and/or B Operands do not have to be provided explicitly at a GENERATE Block. When no values are specified, "default" values of zero are assumed by the Processor. Figure 2.5 shows an example in which the default option has been taken with the B Operand. The A Operand is 10. Because a value

of zero is assumed for the B Operand, the inter-arrival times are uniformly distributed over the integers 10±0. That is, inter-arrival times are always exactly 10. This is an example, then, of deterministic (i.e., non-random) inter-creation times.

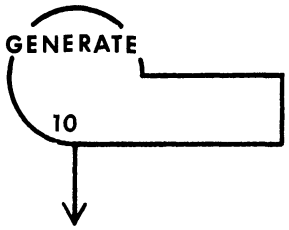


Figure 2.5 A GENERATE Block with No Explicit B Operand

Three additional GENERATE Block Operands will now be introduced and discussed. They are shown in Table 2.4. The C Operand, if used, supplies an Offset Interval. The Offset Interval is a time. In particular, it is the time when the first Transaction is to arrive at the GENERATE Block. After this first arrival, all subsequent arrivals follow the inter-arrival time distribution provided through the A and B Operands. Put differently, the C Operand can be used by the analyst to force the first Transaction arrival to occur at whatever particular time he specifies. After that, times of arrival are generally random, per the information provided at the A and B Operands. When the C Operand is not used, all arrivals at that GENERATE Block take place at times governed by the A and B Operands.

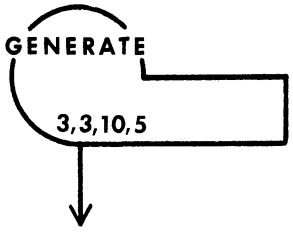
Operand	Significance	Default Value
C	Offset Interval	No Offset in Effect
D	Limit Count	Infinity
E	Priority Level	Zero

Table 2.4 Significance of GENERATE Block C, D, and E Operands

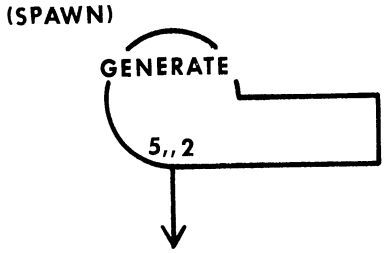
The D Operand places a limit on the total number of Transactions which can enter the model through a given GENERATE Block during a simulation. When that many arrivals have occurred, the GENERATE Block becomes inactive. If no Limit Count is specified, the GENERATE Block remains active throughout a simulation.

The E Operand states the Priority Level, or Priority Class, of each Transaction entering the model through a given GENERATE Block. For reasons to be made clear later, it is convenient to make distinctions among Transactions with respect to their relative processing priority. In total, 128 different Priority Levels are possible in a model. These Priority Levels are designated numerically as 0, 1, 2, 3, 4, ... , 125, 126, and 127. The higher the number, the higher the priority. For example, a Priority Level of 4 is higher than Priority Levels of 0, 1, 2, and 3. The lowest priority possible, then, is 0. As noted in Tabel 2.4, the default value of a GENERATE Block's E Operand is 0. As a result, all Transactions entering a model through such a Block will be in the lowest Priority Class possible.

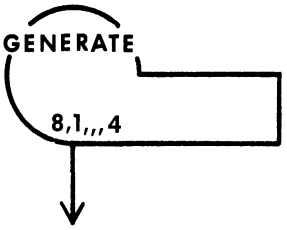
Figures 2.6a, b, and c show examples of GENERATE Blocks in which values have been explicitly provided for the C, D, and E Operands in some cases. In Figure 2.6a, the first arrival is to take place at time 10. After that, inter-arrival times vary at random over the range 3±3, i.e., from 0 to 6, inclusive. However, only 5 Transactions are to arrive in total through that GENERATE Block.



(a) A GENERATE Block with Operands A through D Specified



(b) A GENERATE Block with Operands A and C Specified



(c) A GENERATE Block with Operands A, B, and E Specified

Figure 2.6 Additional GENERATE Block Examples

In Figure 2.6b, the first arrival is to take place at time 2. After that, arrivals are to occur every 5 time units. That is, arrivals will occur at times 2, 7, 12 17, 22, 27, and so on. Note the two consecutive commas which appear after the A Operands. Because nothing is entered between the two commas, there is no B Operand. The default

value of zero is consequently in effect. Note that the Block Location has been given the symbolic name SPAWN.

In Figure 2.6c, all arrivals follow the 8±1 inter-arrival time distribution. No Offset Interval is used. Arrivals will continue at the Block throughout the simulation, because no Limit Count has been provided. Each Transaction generated there will have a Priority Level of 4, as specified by the E Operand. Note the three consecutive commas between the B and E Operands. These have the effect of forcing default values for the C and D Operands.

2.7 Exercises

2.7.1 Show a GENERATE Block at which Transactions will arrive every 7±2 time units, throughout the course of a simulation. Assuming that the third arrival occurs at time 21, list the possible times at which the fourth arrival might take place. What is the probability that the fourth arrival takes place at time 30? What Priority Level will be assigned to Transactions arriving at the GENERATE Block?

2.7.2 Show a GENERATE Block at which Transactions will arrive:

(a) Every 6 time units.

(b) Every 6 time units, except that the first is to arrive at time 15.

(c) Every 6 time units, but only until 10 have arrived in total.

2.7.3 Show a GENERATE Block at which Transactions will arrive every 15±5 time units:

(a) With a Priority Level of 0.

(b) With a Priority Level of 9.

2.7.4 Inter-arrival times at a particular GENERATE Block are to be uniformly distributed over the integers:

(a) 4, 5, 6, 7, and 8. Show a GENERATE Block which will have the desired effect.

(b) 4, 5, 6, 7, 8, and 9. Can you show a GENERATE Block to accomplish this? [When Functions are discussed in Chapter 3, a method will be shown for solving this problem.]

2.7.5 The GENERATE Block shown in Figure P2.7.5 will eventually result in an error condition when the model in which it is used is run on the computer. Can you tell why?

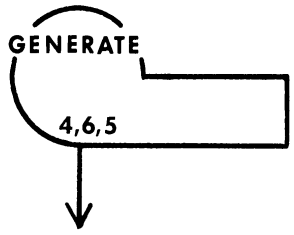


Figure P2.7.5

2.7.6 Why is the Figure P2.7.6 GENERATE Block invalid?

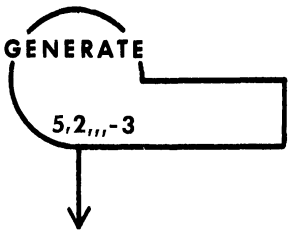


Figure P2.7.6

2.7.7 Why is the Figure P2.7.7 GENERATE Block invalid?

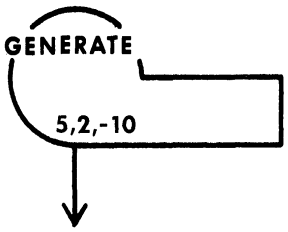


Figure P2.7.7

2.7.8 Figure P2.7.8(a) is valid, whereas Figure P2.7.8(b) is invalid. Explain why.

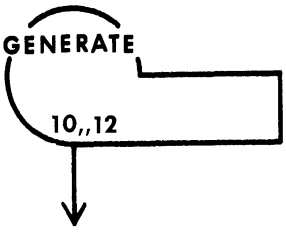


Figure P2.7.8(a)

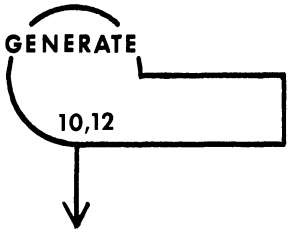


Figure P2.7.8(b)

2.7.9 Assume that only the GENERATE Blocks shown in Figure P2.7.9 are used in a particular GPSS model, and that at time 20 no Transactions have yet been removed from the model. How many Transactions with Priority Level 0 are in the model at time 20? With Priority Level 7? With Priority Level 13?

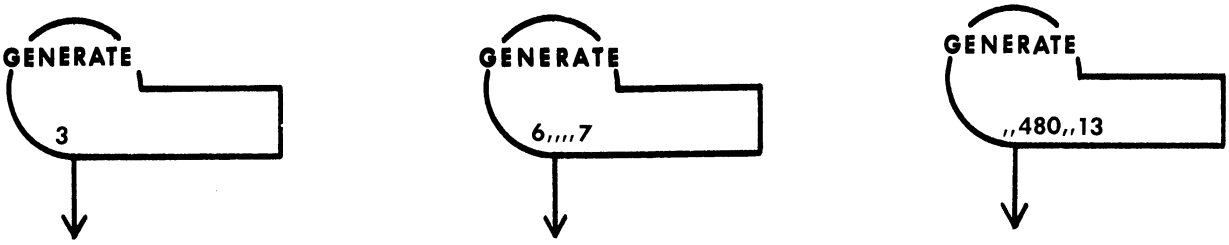


Figure P2.7.9

2.7.10 (a) Suppose it is an analyst's intention to have the implicit time unit in a model be 1 minute. At a particular point in the model, Transactions are to be introduced with inter-arrival times uniformly distributed between 3 and 6 minutes. The analyst provides the GENERATE Block shown in Figure P2.7.10a to produce this effect.

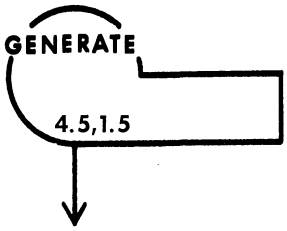


Figure P2.7.10a

Why is his work in error?

(b) Realizing his error, the analyst decides to make the implicit time unit be 0.1 minutes. He modifies the GENERATE Block Operands, with the result shown in Figure P2.7.10b.

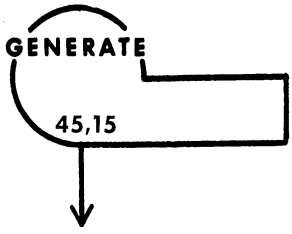


Figure P2.7.10b

How many different values can the inter-arrival time random variable take on in this case?

(c) Still later, the analyst decides that he must work with a smaller implicit time unit. He chooses 1 second as the unit. Show the appearance of the Figure P2.7.10a GENERATE Block as modified to correspond to this smaller time unit. Now how many different inter-arrival times can be realized at the GENERATE Block?

2.7.11 Transactions are to arrive at a GENERATE Block every 0.6±0.2 days. State what the A and B Operands should be for the GENERATE Block if the implicit time unit is to be:

(a) 0.1 days

(b) 1/5 day

(c) 0.3 hours

(d) 0.1 hours

2.8 Punchcards Corresponding to Blocks in GPSS Block Diagrams

The three types of information carried by Blocks have already been described in Section 2.5. Corresponding to this information, there are three fields laid out on the punchcard used for Block representation. The card columns making up each field are shown in Table 2.5.

Card Columns	Block Information
2-6	Location
8-18	Operation
19-71	Operands

Table 2.5 Punchcard Fields in Which Block Information is Entered

The symbolic Location Name (if any) of a Block must be punched in consecutive columns anywhere within the field consisting of columns 2 through 6. The Block's Operation is entered in consecutive columns in the field beginning with card column 8. The Operands must be punched in order in a field beginning with column 19. They must be entered in consecutive card columns and be separated from one another by commas, without any intervening blanks.

A coding sheet showing punchcard images for the GENERATE Block examples in Figures 2.6a, b, and c appears in Figure 2.7.[2d] Note that "explanatory comments" have been entered in Figure 2.7, beginning (arbitrarily) in column 31. The same comments could also be entered on the punchcards themselves. This is because the first blank column encountered in the Operands field causes the GPSS Processor to terminate its scan of that punchcard. Explanatory comments can consequently be included toward the end of each punchcard for purposes of model documentation.

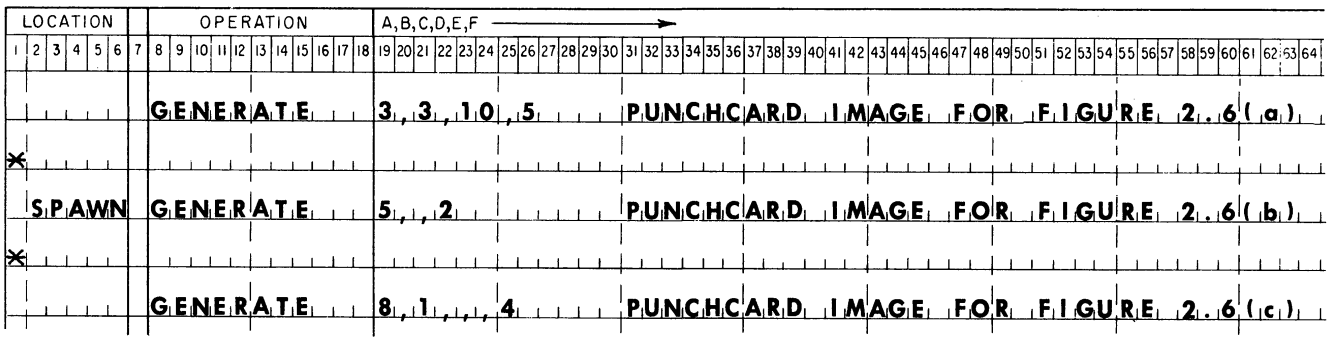


Figure 2.7 Punchcard Images for the Blocks in Figures 2.6a, b, and c

[2d] Pads of GPSS coding sheets are available through IBM. The Form Number for these pads is GX20-1701.

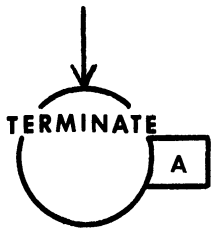
Note that column 1 is not part of the Location field. When an asterisk is entered in column 1, the Processor ignores the entire card. This provides further possibilities for documenting models with lengthy comments, and/or leaving space between distinct model segments. In this spirit, "blank cards" have been inserted between the three examples in Figure 2.7.

The first and third examples in Figure 2.7 have nothing entered in the Location field. This is consistent with the fact that, in Figures 2.6a and c, the GENERATE Blocks have not been given Location names. The Figure 2.6b Block has been tagged with the symbolic Location Name SPAWN. That name is therefore entered in columns 2 through 6 in the second Figure 2.7 example.

2.9 Removing Transactions from a Model: The TERMINATE Block

Transactions are removed from a model whenever they flow into a TERMINATE Block. TERMINATE Blocks always accept Transactions which seek to move into them. There may be any number of TERMINATE Blocks in a model.

The TERMINATE Block, with its A Operand in its usual position, is shown in Figure 2.8. As indicated, the A Operand is a Termination Counter decrement. That is, it is the amount by which a special counter, called the Termination Counter, is to be decremented each time a Transaction moves into the TERMINATE Block. When the analyst chooses not to provide a TERMINATE Block A Operand, a default value of zero is in effect. Movement of Transactions into such TERMINATE Blocks then does not decrease the value of the Termination Counter.



Operand	Significance	Default Value
A	Termination Counter Decrement	Zero

Figure 2.8 The TERMINATE Block and Its A Operand

What is the Termination Counter? It is a computer memory location in which a positive integer value is stored at the time a simulation run is begun. As the simulation proceeds, Transactions flow into TERMINATE Blocks from time to time, resulting in decrementation of this counter. As soon as the counter has been decremented to zero (or less), the simulation stops.

Note carefully that, although there may be many TERMINATE Blocks in a model, there is only one Termination Counter. It is this Termination Counter which will be decremented whenever a Transaction flow into any TERMINATE Block in a model.

As already indicated, the Termination Counter is supplied with its initial value at the time a simulation begins. The GPSS Processor starts the simulation when it

encounters a START Card in the punchcard version of a model. It uses the A Operand on the START Card as the initial value for the Termination Counter.

The format for the START Card is displayed in Figure 2.9. As shown, the word START is punched in the Operation field on the card. The A Operand, as usual, is entered beginning in column 19. Figure 2.9 also shows a specific example of a START Card in which the A Operand has a value of 1.

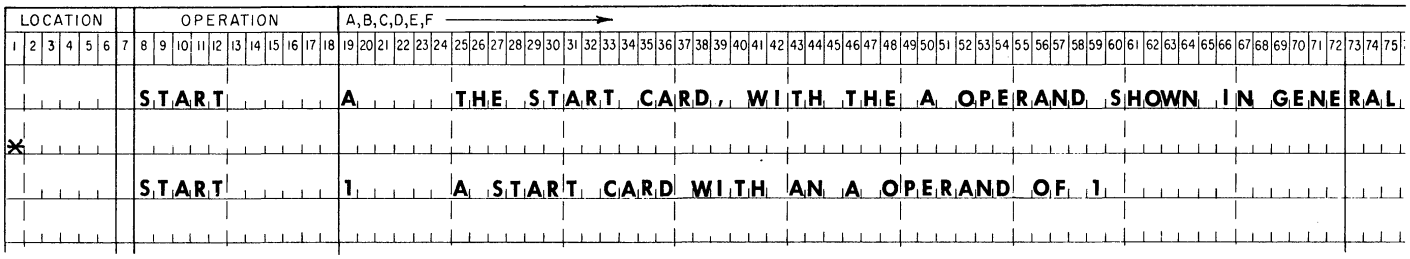


Figure 2.9 Format for the START Card

Now consider an example in which the TERMINATE Block and START Card are used in harmony to control the duration of a simulation run. Suppose a model builder has chosen 1 minute as the implicit time unit in a model. He wants to run the model through 8 hours of simulated time, and then have it shut off. This is the approach he might use:

(1) He includes in the model the two-Block segment shown in Figure 2.10.

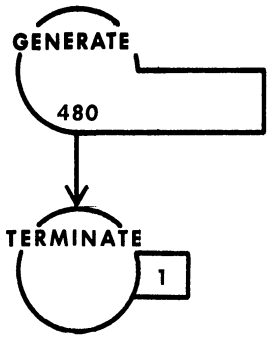


Figure 2.10 A Two-Block Segment Which Shuts Off a Run at Simulated Time 480

(2) At all other TERMINATE Blocks in the model, he defaults on the A Operand. This means that terminations occuring at those Blocks during the simulation do not cause the Termination Counter to be decremented.

(3) He punches 1 as the A Operand on the START Card.

The A Operand of 1 on the START Card causes the Processor to give the Termination Counter an initial value of 1 when the simulation is started. As the simulation proceeds, Transaction terminations that occur from time to time at other TERMINATE Blocks in the model have no effect on the Termination Counter. Then, at simulated time 480, a Transaction finally appears at the Figure 2.10 GENERATE Block and flows to the next Block. There, the

Transaction is removed from the model. Due to the TERMINATE Block's A Operand of 1, 1 is subtracted from the Termination Counter in the process. But this has the effect of decreasing the counter's value from 1 to 0. The Processor therefore shuts off the simulation.

This may seem like a strange way to implement run control in a model. Nevertheless, it is the only way to control the duration of a run in a GPSS model.

Now suppose that, to accomplish the same objective stated in the preceding example, the analyst uses this approach:

(1) He includes in the model the two-Block segment shown in Figure 2.11.

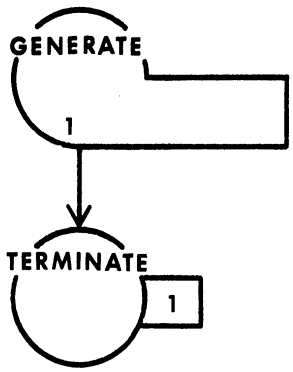


Figure 2.11 An Alternative Two-Block Segment Which Shuts Off a Run at Simulated Time 480

(2) At all other TERMINATE Blocks in the model, he defaults on the A Operand.

(3) He punches 480 as the A Operand on the START Card.

Note that, at the Figure 2.11 GENERATE Block, the inter-arrival time is 1. That is, Transactions arrive there at times 1, 2, 3, 4, ... , 478, 479, and 480. Upon arrival, each of these Transactions flows to the next Block, where it is removed from the model and the Termination Counter is decreased by 1. When the 480th Transaction flows into the Figure 2.11 TERMINATE Block, then, the Termination Counter has already been decremented to 1. The 480th Transaction causes it to be decreased from 1 to 0, and the simulation stops.

The approach shown in the first example is much better than that just shown. The reason is simple. Although both approaches are logically sound, the latter requires 480 executions of the GENERATE and TERMINATE Blocks in Figure 2.11, whereas the former only requires 1 execution of each Block in Figure 2.10. Each Block execution consumes computer time.[2e] The latter approach, then, is about 480 times more expensive than the former.[2f]

[2e] On the System/360, Model 67, each Block execution requires about 1 millisecond of computer time. The actual time varies, of course, depending on the type of Block in question. The 1 millisecond figure is nonetheless a conveneient rule of thumb.

[2f] Actually, the factor of 480 might be on the low side, depending on conditions in other parts of the model.

As a final point, recall from Section 2.6 that with one rare exception, when constants are supplied in GPSS as Operands, they must be integers. This general statement holds, of course, for the TERMINATE Block and START Card A Operands, as well as in the vast majority of other instances. This fact should be fixed firmly in mind.

2.10 Exercises

For each of these exercises, assume that all TERMINATE Blocks other than the ones shown have default values of zero for their A Operands.

2.10.1 The two-Block segment in Figure P2.10.1 is used as a "timer" in a GPSS model. The START Card has 8 entered as the A Operand. At what simulated time will the run shut off?

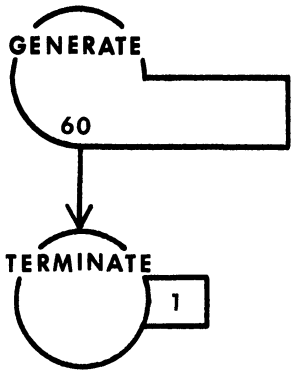
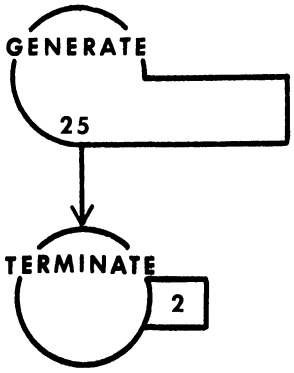


Figure P2.10.1

2.10.2 Suppose that the A Operand in the Figure P2.10.1 GENERATE Block is changed to 70, and that all other conditions are left the same. At what simulated time will the run shut off?

2.10.3 Suppose that the A Operand in the START Card in Problem 2.10.1 is changed to 3, and that all other conditions are left the same. At what time will the GPSS Processor shut off the model? What will be the final value of the Termination Counter in this case?

2.10.4 A modeler decides to use two, two-Block segments to control a simulation run. The two segments are shown in Figure P2.10.4. If he uses a START Card with an A Operand of 25, when does the simulation shut off? Is the final value of the Termination Counter 0, or -1? Explain your answer.



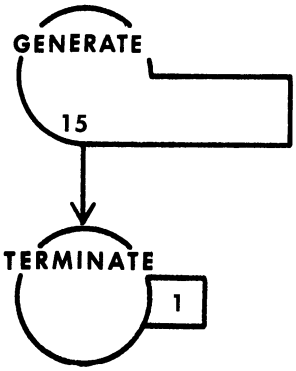


Figure P2.10.4

2.11 Entities to Simulate Single Servers: Facilities

Consider the concept of entities whose purpose is to perform "service upon demand." Such entities might either be people, or things. For example, these people provide service on demand.

(a) a barber
(b) a gas station attendant
(c) a repairman
(d) an insurance agent
(e) a carpenter

In a similar sense, these things are designed to provide service on demand.

(a) a card punch
(b) a pencil
(c) a parking space
(d) an opera glass
(e) a crane

Whether people or things, entities such as those listed above will be referred to as "servers." As understood here, servers are characterized by two properties of interest.

(1) Each server can only respond to one demand for service at a time. If a new service demand arises when the server is already providing service, then the new demand must either (a) wait its turn for the server, or (b) go elsewhere.[2g]

(2) When a server has been engaged, time elapses while the service demanded is performed. This time is termed service time.

In GPSS, the term "Facility" is a synonym for "server." Just as there can be many servers at different points in a system, there can be many Facilities in a GPSS model. Names are given to Facilities, making it possible to distinguish among them. The names

---

[2g] There is a third possibility. If the new demand is "important enough," it can interrupt the server at the expense of the earlier demand. This interrupt capability can be modeled in GPSS. It will be discussed in Chapter 6.

are supplied by the model builder, rather than being pre-defined. Names can be either numeric, or symbolic.

When Facilities are named numerically, positive whole numbers must be used. The largest number which is valid equals the maximum number of different Facilities allowable in a model.(2h)

When Facilities are named symbolically, the same set of rules that applies for naming Block Locations must be followed. As previously discussed, symbolic names are composed of from 3 to 5 alphanumeric characters, with the restriction that the first 3 be alphabetic. Examples of valid and invalid names for Facilities, both numeric and symbolic, are shown in Table 2.6.

Valid	Invalid
CRANE	IT
26	26KEY
CPU	OS
SURVR	-5
1	94528

Table 2.6 Examples of Valid and Invalid Names for Facilities

It was pointed out in Section 2.3 that the abstract concept of "Transaction" has meaning, from the analyst's point of view, in terms of the analogies he draws between Transactions and elements in the real system being modeled. The same observation holds for Facilities. A modeler might choose, for example, to let Facility 9 represent a repairman in a maintenance system. Or, the Facility named CPU might be chosen to represent a central processing unit in a computer system, and so on. The process of drawing analogies between abstract concepts in GPSS, and their real-system equivalents, is an inherent part of modeling in the GPSS language.

2.12 Engaging and Disengaging Facilities: The SEIZE Block and the RELEASE Block

Suppose we want to use a server. In doing so, we go through this series of steps.

(1) We wait our turn, if necessary. Of course, waiting takes place over an interval of time.

(2) When our turn comes, we engage the server. It might also be said that we "capture," or "seize," the server. The event "seize the server" occurs at a point in time.

(3) We hold the server in a state of capture while the service demanded is performed. The service is performed over an interval of time.

(4) When the demanded service has been performed, we disengage the server. It might also be said that we "release" him. The event "release the server" occurs at a point in time.

As would be expected, this same series of steps is followed when simulating the use of a server in GPSS. The GPSS implementation of steps (2) and (4) will now be considered. The means of providing step (3), and the possibility of gathering statistics on the step (1) waiting process, are then taken up in the following sections.

In GPSS, the entities which place demands on Facilities for service are Transactions. It is the nature of Transactions that they tend to flow forward in a model,

---

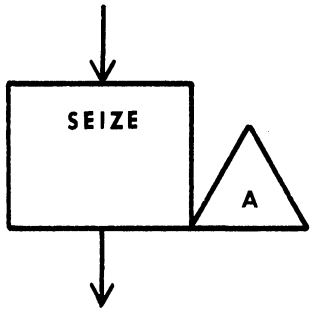
(2h) The maximum number of Facilities allowed in a model depends on the amount of computer storage available. For example, in GPSS/360, the normal quantity of Facilities with 64K (i.e., 64,000) bytes of memory is 35; with 128K bytes, 150; and with 256K bytes, 300. Hence, in a model run with 64K bytes of memory, no Facility number can normally exceed 35. Appendix B shows the normally available quantities of the various GPSS entities.

Block by Block. Suppose that, as its next activity, a Transaction is to seize a Facility (i.e., capture a server). The Transaction accomplishes this objective by moving (or *attempting* to move) into a particular Block associated with the Facility of interest. The Block has these features:

(1) If the Facility is already in use, the Transaction is denied entry to the Block, i.e., it is not permitted to seize the Facility at this time, but must wait its turn. This "denial of permission" to enter a Block brings the moving Transaction temporarily to rest, as was described in Section 2.3.

(2) If the Facility is not in use, the Transaction is permitted to enter the Block. Movement of a Transaction into a Block causes the underlying Block subroutine to be executed. A consequence of the subroutine execution is to change the status of the Facility from "not in use" to "in use."

The Block which has these properties is the SEIZE Block. This Block and its A Operand are shown in Figure 2.12.



Operand	Significance	Default Result
A	The name (numeric or symbolic) of the Facility to be seized	Error(2i)

Figure 2.12 The SEIZE Block and Its A Operand

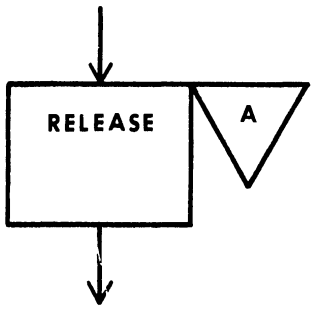
Just as movement of a Transaction into a SEIZE Block simulates capturing a server, movement of the same Transaction into another particular Block simulates releasing the server. The purpose of this other Block, of course, is to change the status of the previously-captured Facility from "in use" to "not in use." The Block which has this purpose is the RELEASE Block. The Block and its A Operand are shown in Figure 2.13.

The RELEASE Block never denies entry to a Transaction seeking to enter it. On the other hand, it would clearly be illogical for a Transaction to attempt to release a Facility not already in use. If such an attempt is made, the GPSS Processor outputs an error message and terminates execution of the model. Furthermore, if the Transaction seeking to release the Facility is not the very same one which has it "in use," an error message is outputted and execution of the model stops.

It is not necessary to declare the existence of particular Facilities to the GPSS Processor before referring to them at SEIZE Blocks. The very fact that they are referenced at SEIZE Blocks forces the Processor to recognize their existence. By the same token, recall that when Transactions were discussed, nothing was said about declaring their existence to the Processor before attempting to generate them. In short, the Processor has been

(2i) Appendix C contains a complete list of GPSS error messages.

taught to automatically provide Transactions and Facilities (and other GPSS entities not yet discussed) as required by the logic in a model. Whereas Transactions lead a transient exitence, however, all Facilities referenced in a model exist throughout the course of a simulation run.



Operand	Significance	Default Result
A	The name (numeric or symbolic) of the Facility to be released	Error

Figure 2.13 The RELEASE Block and Its A Operand

In concluding this section, note that the events "engage a server" and "disengage a server" are complementary. The latter event reverses the effect produced by the former. The SEIZE and RELEASE Blocks, then, can be thought of as a complementary pair. Many other GPSS Blocks exist in "complementary pairs." One member of the pair always has the effect of "undoing," or "reversing," the effect of the other pair member. In most cases, the "characteristic shapes" of complementary Blocks are mirror images of each other with respect to a horizontal axis. This makes the shapes easier to remember. Figure 2.14 shows the SEIZE and RELEASE Blocks in a mirror-image perspective.

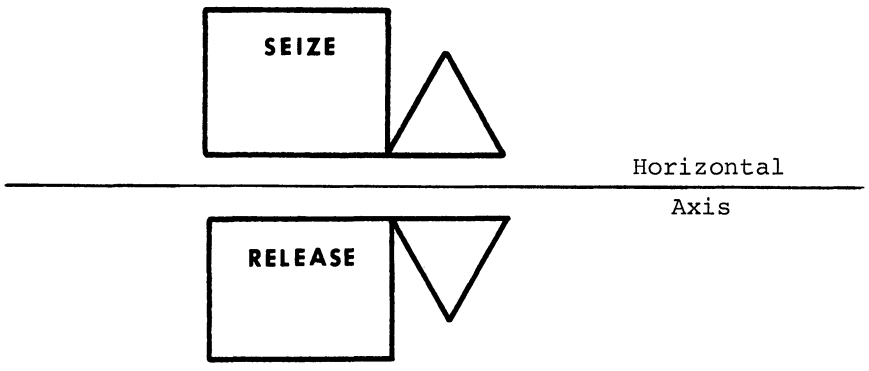


Figure 2.14 The SEIZE and RELEASE Blocks as Mirror Images

2.13 Providing for the Passage of Time: The ADVANCE Block

Assume that a Transaction has just moved into a SEIZE Block, thereby capturing a Facility. After executing the Block subroutine, the Processor immediately attempts to move the Transaction into the next sequential Block. There are very few restrictions on what this next Block might be. For example, it could be another SEIZE Block, referencing another Facility. This would make sense logically if it were necessary, say, to engage both a repairman and a particular tool before a certain type of service could be performed. If the repairman and the tool were simulated with a pair of Facilities, the Transaction would have to simultaneously hold both in a state of capture before its demand for service could be met.

Usually, though, a Transaction captures a Facility with the objective of immediately receiving service from it. As noted in Section 2.12, time passes while the service is performed. While this time is passing, the Transaction should cease its forward motion in the model. Only after the service time has elapsed should it move on into a RELEASE Block to disengage the server.

The ADVANCE Block is provided in GPSS to accomplish the task of freezing a Transaction's motion for a prescribed length of time. The "prescribed length of time" is usually a random variable. This is consistent with the experience that service time usually varies from one service to the next.

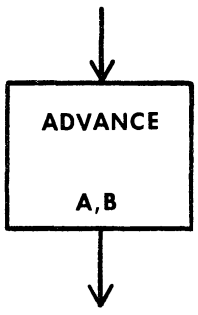
The information required to describe the applicable service time distribution is expressed through the ADVANCE Block's A and B Operands. The various possible service time distributions are divided into two categories:

(1) Uniformly distributed service times.

(2) All other service time distributions.

In short, as with the GENERATE Block, a "special case" is made of the uniform distribution. Expressing more complicated distributions requires the use of GPSS Functions. Because Function definition is deferred to the next chapter, only uniformly distributed service times will be considered for now.

Figure 2.15 shows the ADVANCE Block with its A and B Operands. As indicated, the A Operand supplies the average time that Transactions entering the Block are held there. The B Operand provides the half-width of the range over which the holding times are understood to be uniformly distributed.



Operand	Significance	Default Value
A	Average service time	Zero
B	Half-width of range over which service time is uniformly distributed; called	Zero

Figure 2.15 The ADVANCE Block and Its A and B Operands

Figure 2.16 shows an advance Block with A and B Operands of 30 and 5, respectively For each Transaction flowing into that Block, the range of possible holding times varies over the integers from 25 to 35, inclusive. Suppose a Transaction moves into the Block at time 134 and, executing the Block subroutine, the Processor draws a sample of 31 from the 30±5 distribution. The Transaction will then be delayed at the Block until future time 134 + 31, or 165. At that time, the Processor will attempt to move it into whatever is the next sequential model Block.

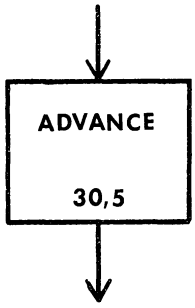


Figure 2.16 An ADVANCE Block with Specific A and B Operand Values

The ADVANCE Block never refuses entry to a Transaction. Any number of Transactions can be held there simultaneously. Whenever a Transaction moves into such a Block, the underlying subroutine is executed again and a customized holding time is computed. The newcomer, then, is in no way influenced by the presence of other Transactions in the Block.

The classical SEIZE-ADVANCE-RELEASE pattern is shown in Figure 2.17. A Transaction moving down the indicated path eventually will capture Facility 5, hold it for 12±12 time units, and then release it. After the Transaction enters the RELEASE Block and that subroutine has been executed, the Processor will attempt to move it into the next Block in the model. Meantime, with execution of the RELEASE subroutine completed, the next Transaction intending to use Facility 5 will be able to capture it.

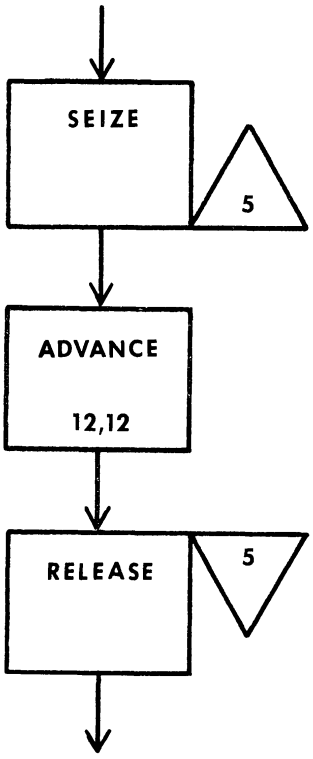


Figure 2.17 An Example of the SEIZE-ADVANCE-RELEASE Sequence

In the Figure 2.17 example, a holding time of zero will be computed for about 4% of the Transactions which enter the ADVANCE Block. Rather than holding the Transaction at the ADVANCE Block when this happens, the Processor immediately moves it to the next Block. There, it releases the Facility it had "just" captured. There is nothing invalid here. If the analyst includes such possibilities in models he constructs, it is up to him to interpret their significance in the context of the system he is modeling.

Do not form the conclusion from the Figure 2.17 example that ADVANCE Blocks can only be placed after SEIZE Blocks in a GPSS model, or that RELEASE Blocks must follow them. ADVANCE Blocks can actually be put anyplace in a model. The Processor will not object. Of course, the choice of their locations should be defensible on logical grounds.

2.14 When Waiting Occurs: Where and with What Privileges Transactions Wait

Consider a situation in which a Transaction moves down the path shown in Figure 2.17. When it arrives at the Figure 2.17 model segment, suppose that Facility 5 is in a state of capture. The Transaction is denied permission, then, to move into the SEIZE Block. Two questions arise:

(1) While the Transaction "waits" for the Facility, where does it wait?

(2) When the Facility again becomes available and two or more Transactions are waiting for it, which Transaction will be permitted to capture it next?

The answer to the first question is that the Transaction waits at the Block preceding the SEIZE. Does this mean that the effect of the Transaction's having entered that preceding Block has not yet taken place? Not at all. Remember that, when a Transaction moves "through" a Block, the sequence of events is (a) the Block subroutine is executed as soon as the Transaction moves into the Block, then (b) the Processor attempts to move the Transaction into the next Block. It is entirely possible for a Transaction to come to rest in a Block after it has triggered execution of that Block's subroutine. This type of "coming to rest" is, in a true sense, involuntary; the Transaction would prefer to keep moving forward in the model, if system conditions permitted. Contrast this with a Transaction "voluntarily" coming to rest in an ADVANCE Block.

The answer to the second question touches on the concept of queue discipline. "Queue" is a synonym for "waiting line." The queue discipline exercised by a server is the rule he follows in determining who to service next, given that two or more demands for service await him. In our normal activities, we usually encounter "first-come, first-served" queue discipline. The person who has been waiting the longest for the server captures him next. This is the queue discipline which is implemented by the GPSS Processor as the "default" case.

Actually, the default queue discipline implemented in GPSS is slightly more sophisticated than "first-come, first-served." It is first-come, first-served, within Priority Class. It was pointed out in Section 2.6 that each Transaction has a particular priority associated with it. The Priority Level of waiting Transactions is automatically taken into account by the Processor when selecting the one which is to seize a Facility next. In a case study later in this chapter, it is shown how priority distinctions can be put to use in models, and made clear why the GPSS default queue discipline is first-come, first-served, within Priority Class.

2.15 Gathering Statistics When Waiting Occurs: The QUEUE Block and the DEPART Block

It is frequently of interest to gather information about the behavior of a waiting line. Answers to questions such as these might provide valuable insight for a decision-maker.

(a) How many entries were there to the (potential) waiting line?

(b) How many of these entries were actually forced to wait, in contrast with being able to capture the server immediately?

(c) What was the maximum number waiting at any one time?

(d) What was the average number waiting?

(e) Of those who had to wait, how much time did they spend in the queue on average?

At his option, the analyst can make use of a complementary Block pair in GPSS to produce answers to these questions.

It is important to realize that gathering such statistics is optional. Some elaboration is worthwhile here. Figure 2.18 is intended to suggest a "system" in which involuntary waiting might potentially occur at up to 6 different locations. The model builder might want to maintain statistics on the waiting process at each of the 6 locations, or at none of them. More likely than not, however, he will want to study the waiting process at some of the locations, but not all. He might judge it to be important to gather statistics at locations 2, 4, 5, and 6, but not locations 1 and 3. Note that, if statistics for some locations are not to be used later anyway, it is better not to collect them. Collecting them results in additional overhead for the Processor, leading to increased execution times and larger run charges.

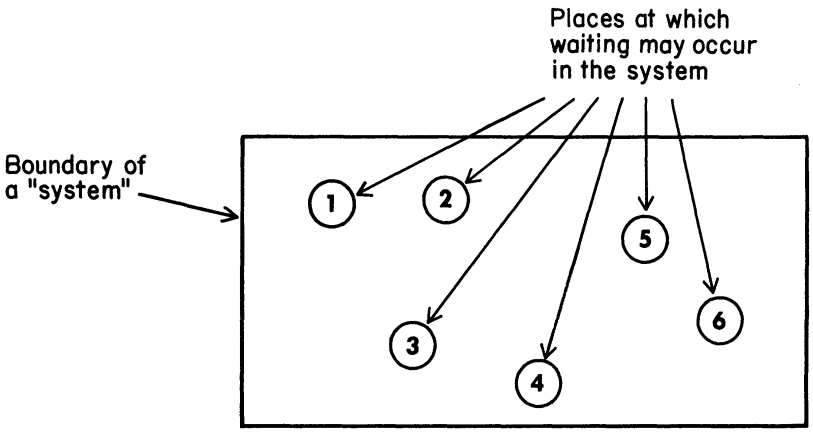


Figure 2.18 Six Locations at Which Involuntary Waiting Might Occur in a Particular System

In this book, waiting lines for which statistics are being maintained are termed explicit Queues, or simply Queues. Those for which no data are being gathered are termed implicit queues.

To distinguish among different explicit Queues in a model, names are supplied for them by the model builder. The naming conventions are the same as for Facilities. Names can either be numeric or symbolic. If numeric, they must be positive whole integers. The largest number which is valid equals the maximum number of explicit Queues allowable in a model.[2j] If symbolic, they are composed of from 3 to 5 alphanumeric characters, with the first 3 being alphabetic.

[2j] As with Facilities, the maximum number of explicit Queues allowed in a model depends on the amount of computer storage available. The normal quantities of explicit Queues are 70, 150, and 300, for 64K, 128K, and 256K bytes of memory, respectively.

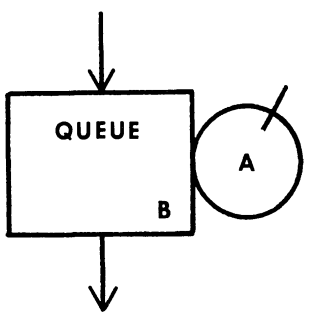
The role of the complementary Block pair used to request that statistics be gathered at a point of potential waiting is easily pointed out in terms of our own experience with respect to waiting lines. Three steps can be identified.

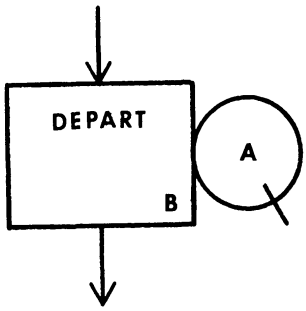
(1) We join the waiting line. That is, we "queue up." Queueing up is an event which occurs at a point in time.

(2) We wait our turn. The waiting occurs over an interval of time.

(3) We depart the waiting line. The departure is an event which occurs at a point in time.

The QUEUE Block, the first member in the pair, simulates the event "join the waiting line." The DEPART Block, the other member, simulates the event "depart the waiting line." The two Blocks, and their A and B Operands, are shown in Figure 2.19. The A Operand is the name of the Queue involved. The B Operand is the number of units by which the record of Queue-content is to be changed.





Operand	Significance	Default Value or Result
A	Name (numeric or symbolic) of the Queue to be joined or departed	Error
B	Number of units by which the recorded content of the Queue is to be modified	1

Figure 2.19 The QUEUE and DEPART Blocks, with Their A and B Operands

The flexibility offered by the B Operand in the QUEUE-DEPART Block pair is puzzling at first. After all, it is a Transaction which moves into a QUEUE Block, thereby triggering the event "a waiting line is joined." Since a Transaction causes the event, it seems logical that the Queue involved should have its recorded content increased exactly by 1. Why have it increased by, say, 2 or more? An example helps provide an answer. Suppose a Transaction represents a train. A proposed track system is being studied. At one point in the system, one or more trains are to wait on an auxiliary track until a switch is thrown. The maximum number of train cars which ever simultaneously have to wait on that auxiliary track is to be measured, so that the required length of track can be determined. When a train pulling several cars arrives at that strip of track, it increases the car-content of the waiting line by more than 1. In GPSS, this situation could be modeled by having a Transaction flow into a QUEUE Block, increasing the content of the waiting line by more than 1. It is occasionally convenient, then, to have the flexibility provided by the B Operand at the QUEUE-DEPART Blocks, even if the default value of 1 will most frequently be used.

Referring again to Figure 2.17, it is clear that involuntary waiting might occur in conjunction with the use of Facility 5. Suppose that statistics are to be collected for this waiting process. An explicit Queue is introduced into the model segment by sandwiching the SEIZE Block between the QUEUE-DEPART Block pair, as shown in Figure 2.20. The name 5 has

been given to the Queue. This choice is arbitrary. There is no reason why the Queue and Facility would have to be named identically. It may be convenient for the analyst to choose identical names for a waiting line and the server ahead of whom that line forms. But, to the GPSS Processor, it of course does not matter.

Suppose that a Transaction comes down the Figure 2.20 path and Facility 5 is available. The Transaction enters the QUEUE Block, that subroutine is executed, and the waiting line content is increased by 1. The Processor then attempts to move the Transaction into the SEIZE Block. The attempt is successful. The SEIZE subroutine is executed, and the status of Facility 5 is changed from "not in use" to "in use". The Transaction then flows into the DEPART Block. That subroutine is performed, and the waiting line content is decreased by 1. Next, the Transaction flows into the ADVANCE Block. Assuming a non-zero holding time is computed, the Transaction temporarily comes to rest. All of this has happened at a particular reading of the GPSS clock. This means that, although the Transaction joined Queue 5, it did no waiting there. In GPSS, this phenomenon is termed a "zero entry". Because the SEIZE Block is sandwiched between the QUEUE-DEPART pair, all Transactions capturing the Facility move through the QUEUE Block, even if the Facility is available when they first attempt the capture.

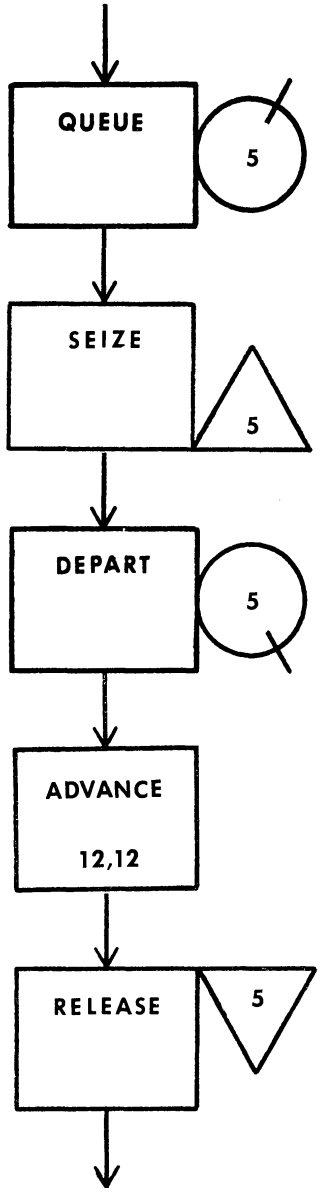


Figure 2.20 An Explicit Queue Provided for the SEIZE-ADVANCE-RELEASE Sequence.

In the Queue statistics it maintains, the Processor distinguishes between "zero entries" and "entries that actually wait". The nature of these statistics will be examined by means of a computerized example in the next section.

A Transaction contributes to the "content" of an explicit Queue when it moves into the corresponding QUEUE Block. That contribution continues, independent of where the Transaction is, until it eventually (if ever) moves into a corresponding DEPART Block. It is a misconception to think to think that a Transaction is not "in the waiting line" unless it is "in the QUEUE Block".

In Figure 2.20, it is true that when a Transaction is in the Queue, it is in the QUEUE Block; and, when it is no longer in the QUEUE Block, it is no longer in the Queue. But this is true in that example only because moving from the QUEUE Block leads directly to movement into the DEPART Block at the same simulated time. This need not necessarily be the case. In fact, the Block Diagram segment shown in Figure 2 .21 is perfectly acceptable to the Processor. A Transaction arrives at the GENERATE Block every 5 time units, joins Queue 7, and is then immediately removed from the model. Although perhaps nonsensical in terms of logic, the example shows that Transactions can "join a Queue," and then be removed from the model without ever "departing that Queue."

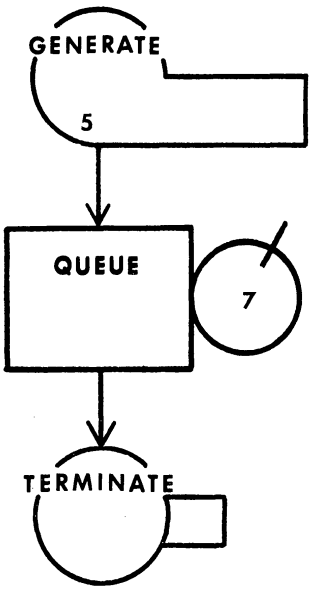


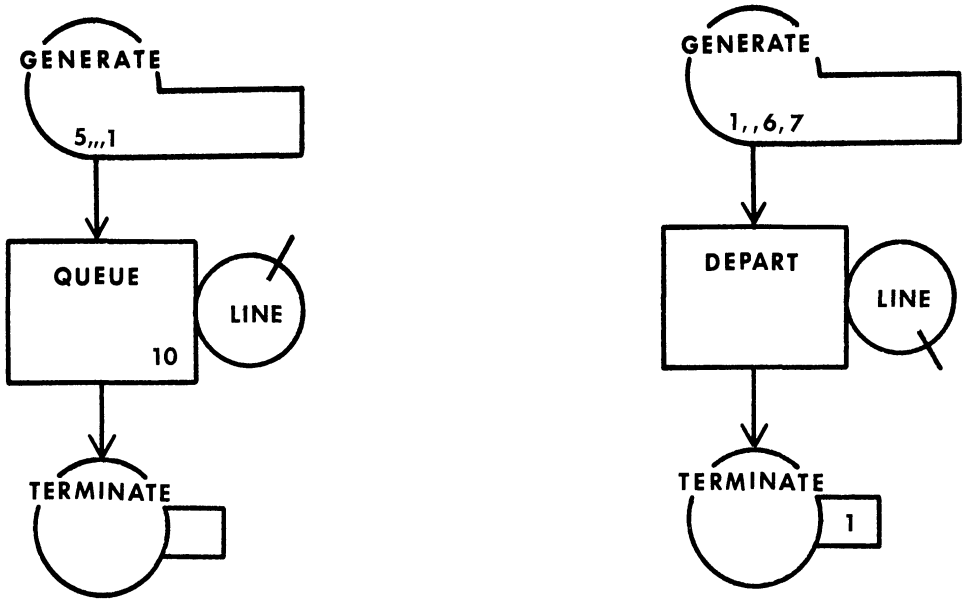
Figure 2.21 Removal of a Transaction from a Model While It Is Still "in a Queue"

In GPSS, then, residence in a Queue is a matter of records, and is not the same as being "physically present in a line." It is easy to find equivalent real situations. For example, a shopper "takes a number" when she enters a meat market. When that number is called, it will be her turn to be served. If the shopper judges that her number won't come up for awhile, she might go next door to a bakery, wait her turn for service there, be served, and then return to the meat market to continue waiting there. As a matter of record, she has been in the meat market all the time, without always being physically present in the store.

Earlier, it was indicated that a Transaction cannot release a Facility unless it is in use, and unless that Transaction is the user of record. There is no such restriction with Queues. If the record of a Queue's content is 0, however, and a Transaction enters a DEPART Block referencing that Queue, an error message is outputted and execution stops. That is, Queue content cannot be negative. But it is possible for a Transaction to depart a Queue which it has never joined. If this happens, the Processor outputs an "Execution Warning Message," but the simulation continues.

Figure 2.22 is a contrived example of a two-segment Block Diagram which is perfectly acceptable to the GPSS Processor. At time 5, a Transaction arrives in segment (a), increases the contents of the Queue LINE from 0 to 10, and then leaves the model. With the Limit Count of 1, that segment is then no longer active. At time 6 in segment (b), and every time unit thereafter, a Transaction enters the model, decreases the content of the

Queue LINE by 1, and then leaves the model. This continues until 7 Transactions have moved through segment (b). If it is assumed that the model consists only of these two segments, and that the A Operand on the START Card is 7, the model now shuts off. The recorded content of the Queue LINE at shutoff is 3.



Segment (a) Segment (b)

Figure 2.22 An Example in Which Transactions Depart a Queue They Never Joined

In the process of building GPSS models for complex systems, the analyst will frequently find these "surprising" language flexibilities to be of considerable convenience.

2.16 Documentation Standard for Case Studies

In this book, frequent use is made of case studies to illustrate pertinent features of GPSS. A consistent documentation pattern is followed in presenting these cases. These are the components of the documentation:

(1) Statement of the Problem

This component provides a sufficiently detailed description of the problem so that a GPSS model for the system described can be built and run.

(2) Approach Taken in Building the Model

This section attempts to explain how the task of interpreting the problem in the context of GPSS was approached. The intention is to explain the rationale behind the particular approach used. For some of the easier problems, the "approach" is almost trivial. For more difficult problems, this may not be true.

(3) Table of Definitions

The importance of using analogies in building GPSS models is clear. Good model documentation includes a description of the analogies chosen by the analyst. The Table of Definitions is a listing of the various GPSS entities used in the model, with a brief explanation of their interpretation as elements in the system being modeled. The implicit time unit chosen by the analyst appears at the head of this table. Then the interpretation given to Transactions appears. After that, alphabetic order is followed for the other entities and their interpretations.

(4) The Block Diagram

In a sense, the Block Diagram is the model. An unadorned Block Diagram can be difficult for someone other than the model builder to follow. For this reason, annotations are placed adjacent to the Blocks in the diagram. Annotations are brief comments which indicate what the Block simulates, or helps to simulate, in the system being modeled.

(5) Extended Program Listing

As the GPSS Processor reads in the punchcard version of a model, these three steps are performed (among others).

(a) A Location Number is assigned to each Block in the model.

(b) A card number is assigned to each punchcard in the deck.

(c) For each card, this information is copied to the printer.

(i) Block Number (if the card is a Block image).

(ii) Information punched in the Location, Operation, and Operands fields in the card, including comments, if any.

(iii) Card number (number of the position this card occupies in the deck).

The result is a Processor-produced listing of the original program. Actually, because numeric Block Locations and Card Numbers are included in this listing, but are not entered on the original cards themselves, this listing is better described as an "extended program listing." The extended listing will always be included in model documentation. At first, such a listing is useful because it implicitly answers the sometimes-asked question, "where do the cards go?" Later, the listings are of less interest. They are provided, though, for the sake of completeness.

(6) Program Output

The printout produced when the simulation is run is shown to display what the analyst receives for his efforts, and often to serve as the basis for discussion. Frequently, only "selected" output is shown. This is done to save space, and to emphasize those portions of the output which are of most interest. In some cases, rather than showing direct printout, summaries are presented.

(7) Discussion

The discussion may involve model logic, model implementation, and program output.

Model Logic

When appropriate, features in the Block Diagram are discussed to relate them to facets of the problem itself, or to the particular approach taken for interpretation of the problem in a GPSS context.

Model Implementation

The punchcard version of a model contains cards corresponding to Blocks, but it also contains cards supplying other information. For example, the START Card discussed in Section 2.9 must be included as part of the model implementation. There is no "Block" in a Block Diagram which corresponds to the START Card. Similarly, a variety of other possibilities exists for including information in the card deck which does not appear directly in the Block Diagram. When the occasion demands, this information will be singled out in the extended source listing and discussed.

Program Output

The program output may occasionally be evaluated in terms of the sense in which it provides an "answer" to the original problem. The primary purpose of the case studies, though, is to illustrate model-building in GPSS, not to develop numeric answers to problems. For this reason, discussion of program output is not particularly emphasized.

2.17 Case Study 2A: A One-Line, One-Server Queuing System

(1) Statement of the Problem

The inter-arrival time of the customers at a one-chair barber shop is uniformly distributed over the range 18±6 minutes. Service time for haircuts is 16±4 minutes, uniformly distributed. Customers coming to the shop get their hair cut, first-come, first-served, then leave. Model the shop in GPSS, making provisions to collect data on the waiting line. Then run the model through 8 hours of simulated time. Interpret the output produced by the model in the context of the barber shop.

(2) Approach Taken in Building the Model

This model is easily constructed as a single sequence of Blocks, excepting the run-control component. The order in which the Blocks appear corresponds to the sequence of stages through which customers move in the real system. Customers arrive; if necessary, they wait their turn; then they engage the barber, get their hair cut, release the barber, and leave. Except for the GENERATE and TERMINATE Blocks, this sequence has already been displayed and discussed in Figure 2.20.

To control the duration of the run, a two-Block timer segment can be used. In Figure 2.10, a segment accomplishing the objective required here is presented and discussed. That segment will be used for this model.

(3) Table of Definitions

Time unit: 1 Minute

GPSS Entity	Interpretation
Transactions	
Model Segment 1	Customers
Model Segment 2	A timer
Facilities	
JOE	The barber
Queues	
JOEQ	The Queue used to gather statistics on the waiting experience of customers

Table 2A.1 Table of Definitions for Case Study 2A

(4) Block Diagram

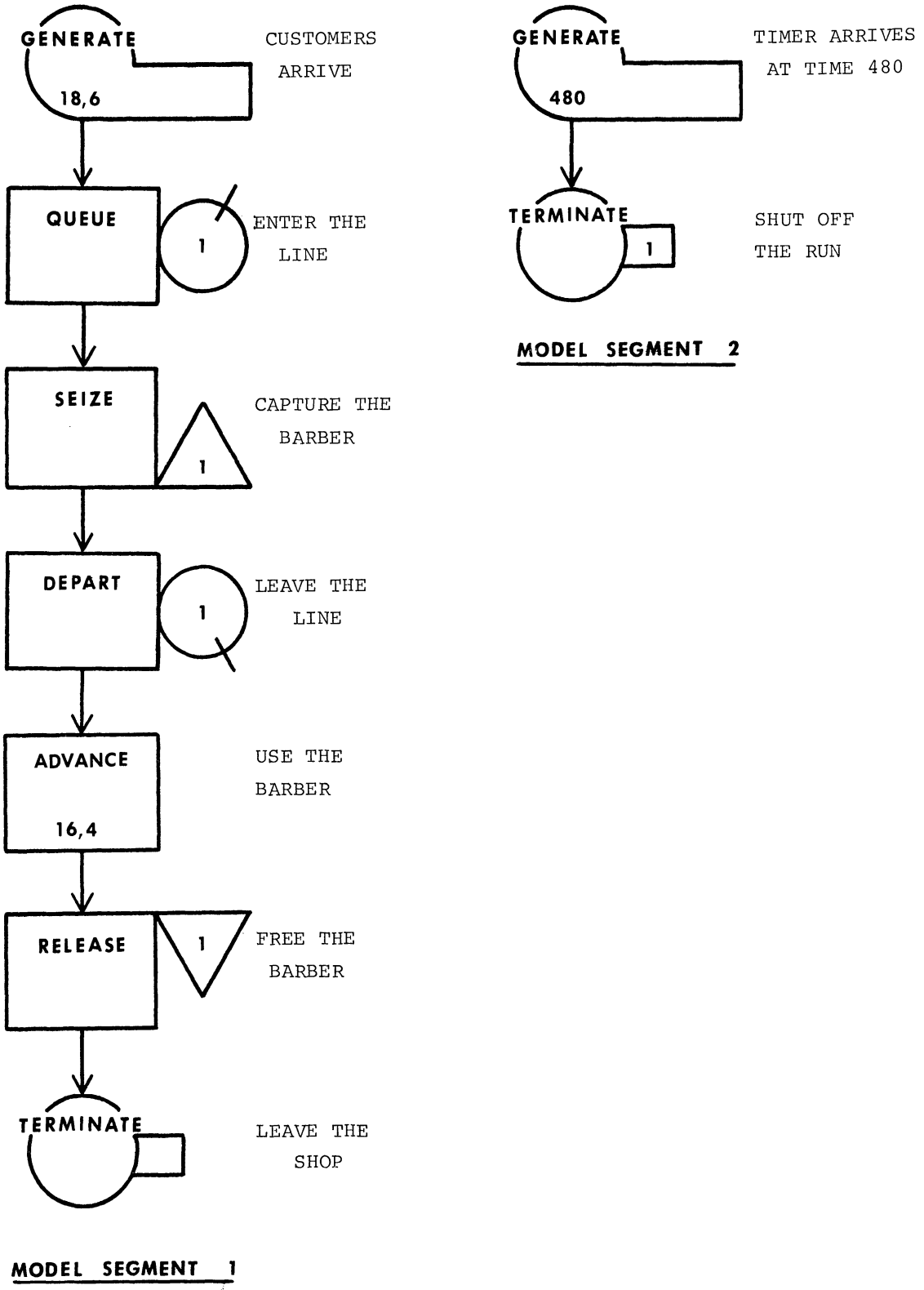


Figure 2A.1 Block Diagram for Case Study 2A

(5) Extended Program Listing

LOCATION		OPERATION	A,B,C,D,E,F →
		SIMULATE	
*			
*		MODEL SEGMENT 1	
*			
		GENERATE	18,6 CUSTOMERS ARRIVE
		QUEUE	JOEQ ENTER THE LINE
		SEIZE	JOE CAPTURE THE BARBER
		DEPART	JOEQ LEAVE THE LINE
		ADVANCE	16,4 USE THE BARBER
		RELEASE	JOE FREE THE BARBER
		TERMINATE	LEAVE THE SHOP
*			
*		MODEL SEGMENT 2	
*			
		GENERATE	480 TIMER ARRIVES AT TIME 480
		TERMINATE	1 SHUT OFF THE RUN
*			
*		CONTROL CARDS	
*			
		START	1 START THE RUN
		END	RETURN CONTROL TO OPERATING SYSTEM

(a) Coding Sheet for Punchcard Version of the Model

```
BLOCK                                                              CARD
NUMBER   *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS        NUMBER
                SIMULATE                                             1
         *                                                           2
         *      MODEL SEGMENT 1                                      3
         *                                                           4
1                GENERATE   18,6          CUSTOMERS ARRIVE          5
2                QUEUE      JOEQ          ENTER THE LINE            6
3                SEIZE      JOE           CAPTURE THE BARBER        7
4                DEPART     JOEQ          LEAVE THE LINE            8
5                ADVANCE    16,4          USE THE BARBER            9
6                RELEASE    JOE           FREE THE BARBER          10
7                TERMINATE                 LEAVE THE SHOP           11
         *                                                          12
         *      MODEL SEGMENT 2                                     13
         *                                                          14
8                GENERATE   480           TIMER ARRIVES AT TIME 480 15
9                TERMINATE  1             SHUT OFF THE RUN         16
         *                                                          17
         *      CONTROL CARDS                                       18
         *                                                          19
                START      1             START THE RUN            20
                END        RETURN CONTROL TO OPERATING SYSTEM       21
```

(b) Extended Program Listing Produced for Model in (a)

Figure 2A.2 Model as Submitted, and Corresponding Extended Program Listing

(6) Program Output

```
1      GENERATE    18     6
2      QUEUE       1
3      SEIZE       1
4      DEPART      1
5      ADVANCE     16     4
6      RELEASE     1
7      TERMINATE
8      GENERATE    480
9      TERMINATE   1
       START       1
```

(a) Assembled Model

```
FACILITY SYMBOLS AND CORRESPONDING NUMBERS


                1    JOE
```

(b) Symbol Dictionary for Facilities

```
QUEUE SYMBOLS AND CORRESPONDING NUMBERS


                1    JOEQ
```

(c) Symbol Dictionary for Queues

```
RELATIVE CLOCK          480  ABSOLUTE CLOCK          480
BLOCK COUNTS
BLOCK CURRENT    TOTAL    BLOCK CURRENT    TOTAL    BLOCK CURRENT    TOTAL
    1      0       27
    2      1       27
    3      0       26
    4      0       26
    5      1       26
    6      0       25
    7      0       25
    8      0        1
    9      0        1
```

(d) Clock Values and Block Counts

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
JOE	.860	26	15.884	3	

(e) Facility Statistics

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS
JOEQ	1	.160	27	12	44.4	2.851	5.133

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(f) Queue Statistics

TABLE NUMBER	CURRENT CONTENTS
	1

Figure 2A.3 Selected Program Output for Case Study 2A

(7) Discussion

Model Logic

In the model presented here, no provision is made for "removing customers from the barber shop" when the simulation shuts off at time 480. If the barber were to be true to the model, he would simply have to "walk out of the shop" at the end of his 8-hour day. Conversely, if the model were to be true to the barber, it would simulate locking the door after 8 hours, but would not shut off until all customers already in the shop at that time had been serviced. It will eventually be seen how this latter approach can be implemented in GPSS.

Model Implementation

The coding sheet from which the punchcard version of the model was prepared is shown in Figure 2A.2(a). The corresponding extended program listing produced by the Processor appears in Figure 2A.2(b). Notice how the Processor has augmented the original information in producing the extended program listing. The "extensions" consist of the "Block Number" and "Card Number" columns appearing at the extreme left and right, respectively, in Figure 2A.2(b). Inspection of the Block Number column shows that Block Numbers have been assigned, in sequence, to each punchcard representing a Block image. In the Card Number column, note that each card in the deck has been assigned a sequence number.

"Comments" have been used liberally to document the model. Cards 2, 3, 4, 12, 13, 14, 17, 18, and 19 in Figure 2A.2(b) are comments cards which set off the model segments, and the Control Card segment. An asterisk (*) has been entered in column 1 on each of these cards. The Block-image punchcards also carry comments in the Operands field. These comments are identical to the annotations written next to the corresponding Blocks in the Figure 2A.1 Block Diagram.

Card 1 in Figure 2A.2(b) is the SIMULATE Card. If the analyst is submitting a deck to have a run made, this card usually must be the first one the Processor encounters when it inputs the deck. The card consists of the single word SIMULATE, punched in the Operation field. If the SIMULATE card is absent, the Processor checks the deck for violations of the language rules, but makes no run with the model.

As stated in Section 2.9, the Processor starts the simulation when it finds a START Card in the model. A START Card has been placed, then, at the end of the model (Card 20). A 1 has been entered as the A Operand on the START Card.

After a run shuts off, the computer session is not necessarily finished. Many additional options remain open to the analyst. Whether or not these options are exercised, the analyst eventually reaches the point at which all instructions for the run have been included in the deck. At this point, he puts in an END Card. This card instructs the Processor to return control to the operating system. The END Card appears after the START Card in Figure 2A.2(b). It consists of the word END, punched in the Operation field.

The order of the cards within a model segment is critical, but the relative ordering of model segments within the card deck is not. For example, the timer segment could have been placed ahead of the major segment in Figure 2A.2 without having any effect on the model. If this had been done, the extended program listing would appear as shown in Figure 2A.4.

```
BLOCK                                                                CARD
NUMBER   *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS          NUMBER
                SIMULATE                                                1
         *                                                              2
         *      MODEL SEGMENT 2                                         3
         **                                                             4
1                GENERATE   480          TIMER ARRIVES AT TIME 480     5
2                TERMINATE  1            SHUT OFF THE RUN              6
         *                                                              7
         *      MODEL SEGMENT 1                                         8
         *                                                              9
3                GENERATE   18,6         CUSTOMERS ARRIVE              10
4                QUEUE      JOEQ         ENTER THE LINE                11
5                SEIZE      JOE          CAPTURE THE BARBER            12
6                DEPART     JOEQ         LEAVE THE LINE                13
7                ADVANCE    16,4         USE THE BARBER                14
8                RELEASE    JOE          FREE THE BARBER               15
9                TERMINATE                LEAVE THE SHOP                16
         *                                                              17
         *      CONTROL CARDS                                           18
         *                                                              19
                START      1            START THE RUN                 20
                END        RETURN CONTROL TO OPERATING SYSTEM          21
```

Figure 2A.4 Extended Program Listing for Case Study 2A with Model Segments Interchanged

Program Output

It is not evident by examining either the Block Diagram of the extended program listing how any output is produced by the model. At the end of a simulation, the GPSS Processor automatically prints out an extensive set of information pertaining to the model. This information includes statistics for each of the various entities used, i.e., for Facilities and Queues (and other entity types not yet discussed).

Most of the output produced by running the Figure 2A.2(a) model is shown in Figure 2A.3. In part (a) of that figure is displayed the assembled model. It has four noticeable features.

(1) The absolute Block Numbers assigned by the Processor appear in the "Location Field". The numbers 1 through 9 in the left column in Figure 2A.3(a) are these Location Numbers.

(2) Instead of appearing in consecutive columns and being separated by commas, the Operands have been printed left-justified in adjacent six-column fields, and the commas have been eliminated. (It is not immediately evident in Figure 2A.3(a) that six-column fields have been used to display the Operands.)

(3) All symbolic entity names in the model have been replaced with the corresponding numeric equivalents assigned by the Processor. Hence, the A Operand of the QUEUE Block (Block 2) is 1, not "JOEQ"; the A Operand of the SEIZE Block (Block 3) is 1, not "JOE", and so on.

(In Chapter 4, the method the Processor uses to establish a correspondence between symbolically-named entities and their numeric equivalents will be described.)

(4) "Comments" on the various cards have been suppressed. Cards which carry nothing but comments have also been suppressed.

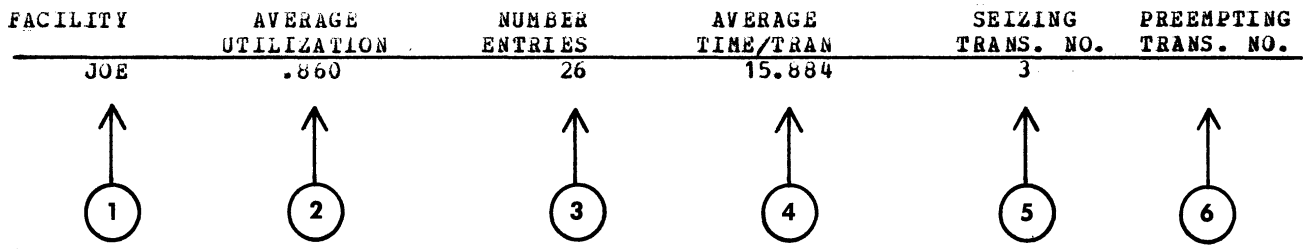
Parts (b) and (c) in Figure 2A.3 show symbol dictionaries for "Facilities" and "Queues". In the symbol dictionary for Facilities, the numeric equivalent assigned by the Processor for all symbolically-named Facilities is shown. Hence, the Facility symbolically named "JOE" is Facility 1 in the assembled model; and the Queue symbolically named "JOEQ" is Queue 1. This is consistent with the A Operands for the SEIZE-RELEASE and QUEUE-DEPART Blocks in Figure 2A.3(a). If any Block Locations had been named symbolically in the model, a corresponding symbol dictionary also would have been provided in the output. Actually, if symbolic Location Names have been used, the correspondence between them and Location Numbers is apparent in the extended program listing.

Figure 2A.3(d) shows "clock values" and "Block Counts". As indicated in the top line of that figure, there are two clocks, the "Relative Clock" and the "Absolute Clock". The distinction between these two clocks will be explained later. For now, it is enough to note that both clocks show values of 480 in Figure 2A.3(d). This simply means that the simulation shut off at simulated time 480.

Immediately under the "clock line" in Figure 2A.3(d) are shown the "Block Counts". This information appears in three columns, "Block Numbers" (labeled simply as BLOCK in the figure), "Current Count" (shown as CURRENT), and "Total Count" (shown as TOTAL). The Block Numbers correspond to those shown in Figure 2A.3(a). The "Current Count" is the count of Transactions "at" ("in") the corresponding Blocks at the time the simulation shut off. The "Total Count" is a count of the total number of Transactions which entered the corresponding Blocks during the simulation, including those that are still in the Block (if any). For example, the Total Count at Block 1 is 27, meaning that 27 Transactions entered the model through the Location 1 GENERATE Block. Similarly, the Total Count at Block 2 is 27, meaning that 27 Transactions moved into the QUEUE Block in Location 2. The Current Count at Block 2 is 1, meaning that 1 Transaction is still in the QUEUE Block, i.e., 1 customer was waiting for the barber when the model shut off. At the Block in Location 5, the ADVANCE Block, the Current Count is 1 and the Total Count is 26. That is, 26 customers have captured the barber; of the 26, 1 still has him captured. The Total Counts at the SEIZE and RELEASE Blocks are 26 and 25, respectively, which is consistent with the ADVANCE Block Counts.

In Figure 2A.3, parts (e) and (f) show the statistics gathered for the Facility JOE and the Queue JOEQ. The Facility statistics are shown again in Figure 2A.5, where the columns have been numbered for ease of reference. The Table appearing in the lower part of Figure 2A.5 indicates the significance of the entries in the various columns. Similarly, in Figure 2A.6, the Queue statistics have been repeated with column numbers included. The Table at the bottom of that figure indicates the meaning of the various Queue statistics. The tables in Figures 2A.5 and 2A.6 should be studied, making reference to the output immediately above them in the process.

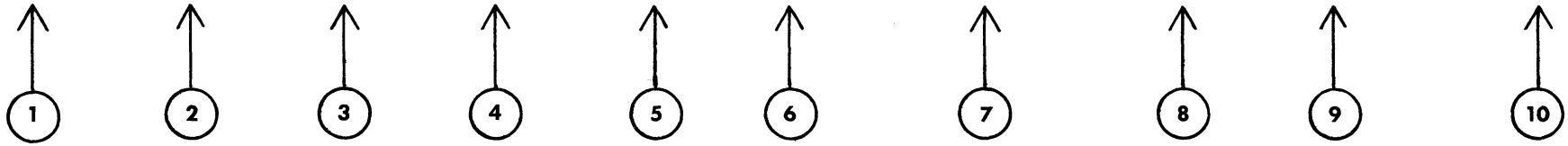
In Figure 2A.3(f), the line "$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES" appears below the line of statistical information. This is simply a definition of the so-called "dollar average Queue residence time" as it appears in column 8 in Figure 2A.3(f). This definition, which is always outputted by the Processor with Queue information, has been eliminated from Figure 2A.6 for purposes of clarity.



Column	Significance
1	Names (numeric and/or symbolic) of the various Facilities used in the model
2	Fraction of the time that the corresponding Facilities were in a state of capture during the simulation
3	Number of captures
4	Average holding time per capture
5	Number of the Transaction (if any) which currently has the Facility captured (Transaction numbers are discussed later in this chapter)
6	Number of the Transaction (if any) which currently has the Facility pre-empted (Pre-empting will not be explained until Chapter 6)

Figure 2A.5 Interpretation of the Information Shown in Figure 2A.3(e)

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
JOEQ	1	160	27	12	44.4	2.851	5.133		1



Column	Significance
1	Names (numeric and/or symbolic) of the various Queues used in the model
2	Largest value the record of Queue content ever assumed
3	Average value of the Queue content
4	Total number of entries to the Queue
5	Total number of entries to the Queue which experienced no waiting ("zero entries")
6	Percentage of total Queue entries which experienced no waiting
7	Average time that each Queue entry spent waiting in the Queue (zero entries are included in this average)
8	Average time that each Queue entry spent waiting in the Queue (zero entries are excluded from this average)
9	Name (numeric and/or symbolic) of the GPSS Table in which the distribution of Queue residence time is being tabulated (the Table concept is not discussed until Chapter 4)
10	Current value of the Queue content

Figure 2A.6 Interpretation of the Information Shown in Figure 2A.3(f)

2.18 External Control Cards Required to Run GPSS Models

After the punchcard version of a GPSS model has been prepared, the resulting card deck must be placed within an appropriate control card sequence before the model can be run. These external control cards have nothing to do with the logic of the model itself. Instead, they provide information specifying the user's account number, indicating that the task to be performed requires use of the GPSS Processor, and so on. When the GPSS model has had the applicable control cards placed around it, the resulting "job" can then be submitted for running on a computer.

Figure 2A.2 shows only the punchcard images (and Processor-supplied extensions) of the GPSS model for Case Study 2A; no external control card images are shown in the figure. The control cards which are appropriate for a job depend on the computer installation at which the job is to be run. For this reason, no attempt is made in this book to provide specific details about these external control cards.[2k] As indicated in IBM's GPSS/360 Operator's Manual, "The individual who is responsible for the installation and maintenance of the GPSS processor at a given computing center should issue a memo to all GPSS users listing the necessary control cards required for the execution of the GPSS program at that center."[2e1] It is henceforth assumed here, then, that such information has been obtained from the computing center at which the GPSS modeling is to be accomplished.

2.19 Exercises

2.19.1 (a) Using Appendix B, determine the normal maximum quantity of Transactions which can be in a model at one time. To what extent is this quantity influenced by the amount of computer memory available?

(b) Use Appendix C to look up the meaning of each of the following error message numbers. For each error indicated, show a Block or a Block sequence which could give rise to the error, or otherwise discuss its significance.
30; 201; 216; 413; 415; 416; 428; 498; 499; 500; 505; 530; 853; 854

(c) For what long-run percentage of the Transactions entering the Block "ADVANCE 5,4" will the holding time be 5?

(d) What is the difference between an "implicit queue" and an "explicit Queue" in a GPSS model?

(e) What is a "zero entry" to a Queue?

(f) When a Transaction moves into the Block "QUEUE 5,2", by what amount is the recorded content of the Queue increased?

2.19.2 (a) Using the average customer inter-arrival time and average service time in Figure 2A.1, compute the long-run utilization of the barber in Case Study 2A. Compare this with the utilization statistics in Figure 2A.3(e).

(b) Using the average customer inter-arrival time in Figure 2A.1, determine how many customers are expected to arrive at the barber shop during a typical 8-hour day. Compare this expected number of arrivals with the actual number of arrivals indicated via the Total Count at the Location 1 GENERATE Block [see Figure 2A.3(d)].

[2k] The IBM GPSS/360: OS Operator's Manual [Form Number GH20-0311] gives examples of the control cards required for execution of a GPSS/360 program when running under IBM's Operating System/360.

[2e1] Ibid, page 8.

(c) In Figure 2A.3(e), show how to compute AVERAGE UTILIZATION, given NUMBER ENTRIES and AVERAGE TIME/TRAN, and knowing the duration of the simulation. Does your independently-computed value equal 0.860 as shown in Figure 2A.3(e)?

(d) From information available in Figure 2A.3(f), give an estimate of the probability that a customer will have to wait for the barber.

(e) In Figure 2A.3(f), show how to compute AVERAGE CONTENTS, given TOTAL ENTRIES and AVERAGE TIME/TRANS, and knowing the duration of the simulation. Does your independently-computed value equal 0.160 as shown in Figure 2A.3(f)?

(f) In Figure 2A.3(f), show how to compute $AVERAGE TIME/TRANS, given AVERAGE CONTENTS, TOTAL ENTRIES, and ZERO ENTRIES. Does your independently-computed value equal 5.133 as shown in Figure 2A.3(f)?

(g) In the statistics for a Queue, would it be possible for PERCENT ZEROS to be 100.0 and MAXIMUM CONTENTS to be 1? Explain.

2.19.3 Prepare the punchcards corresponding to Figure 2A.2. Then, having determined what external control cards are necessary to run a GPSS model at your computing center, place the applicable control cards around the model and submit the resulting "job" for running. Compare the resulting program output with the output shown in Figure 2A.3. Are the Facility and Queue statistics in your output in agreement with the statistics in Figure 2A.3? It is quite possible that the statistics will not be in agreement. Because of the essential randomness of inter-arrival times and service times, results from operation of the barber shop will generally vary from simulated day to simulated day. If the results are in exact agreement, this simply means that the underlying source of random numbers in your GPSS implementation exactly matches the random number source used in producing the Figure 2A.3 output. (The random number sources in GPSS will be discussed in some detail in Chapter 3.)

2.19.4 Modify the Figure 2A.2 model to correspond to a customer inter-arrival time distribution of 15±4 minutes. Run the resulting model for 480 minutes of simulated time. Compare the barber's utilization, number of customers entering the shop, and the number of "zero entries" to the waiting line with the corresponding information in Figure 2A.3. Discuss the observed differences in terms of the changed inter-arrival time distribution.

2.19.5 Modify the Figure 2A.2 model under the assumption that the implicit time unit is 1 second, not 1 minute. Run the resulting model for the equivalent of an 8-hour day. Compare the barber's utilization, number of customers entering the shop, and the number of "zero entries" to the waiting line with the corresponding information in Figure 2A.3. Discuss any observed differences in terms of the change in the implicit time unit.

2.19.6 In the Figure 2A.1 model, the "number of customers serviced" is a random variable. That is, the number of customers serviced will, in general, vary from 8-hour day to 8-hour day. Modify the model so that it shuts off after exactly 25 customers have been serviced. [Hint: As part of the modification, eliminate the two-Block timer segment.] Run the resulting model. At what time does the model shut off (what is the reading of the Relative Clock in the program output?). Note that, in the modified model, the number of

customers serviced is strictly determined, whereas the time of model shutdown is a random variable. In the Figure 2A.1 model, the time of model shutdown is strictly determined, whereas the number of customers serviced is a random variable.

2.19.7 Mechanics arrive at a tool crib every 300±250 seconds. There they check out tools for use in repairing failed machines. The clerk who works at the crib requires 280±150 seconds to fill each mechanic's request for a tool.

(a) Build a GPSS model for this situation, then run the model through 8 hours of simulated time. Suppose that, due to lost production attributable to a failed machine, it costs 0.5¢ per second (i.e., $18 per hour) to have a mechanic wait for service at the tool crib. In this sense, what is the total cost associated with the 8-hour day you simulated?

(b) Suppose that the clerk described in (a) earns $4 per hour. The clerk can be replaced with another who earns $4.50 per hour, and who requires 280±50 seconds to fill each mechanic's request for a tool. Simulate for an 8-hour day with the alternative clerk and compute the resulting cost associated with waiting mechanics. Is it better to use the first clerk or the second at the tool crib? In light of the sample size (i.e., an 8-hour day simulation) used to compare the two alternatives, how "strong" is your conclusion? What would you do to improve your degree of confidence in the conclusion that you draw?

2.20 Internal Logic of the GPSS Processor

An effort has been made in earlier sections of this chapter to provide some insight into the operation of the GPSS Processor. The insight is necessarily still very incomplete. For example, there has been considerable discussion of "movement of Transactions" from Block to Block. Nevertheless, the Block Diagram in Figure 2A.1 no doubt still appears to be much more static in nature than dynamic. And there are still unanswered questions, such as "when a Transaction stops moving, which other Transaction is moved next by the Processor?" and "why is the implicit queue discipline first-come, first-served, within Priority Class?" These questions can only be resolved satisfactorily by understanding the logic on which the Processor itself is based. The next several Sections are devoted to consideration of this logic.

Much of the Processor's logic can be understood by considering the mechanism used to keep track of the various Transactions moving through a model. The Processor regards each Transaction as being on one of several "chains." Each Transaction on a chain may be thought of as a link in the chain. The chains are "open," not "closed," so they have a front end, and a back end. As a chain resident, then, a Transaction occupies a specific location relative to the front of the chain. A Transaction's chain location is closely involved with when it will again be that Transaction's "turn" to be moved forward in the model by the Processor. The processing sequence, in turn, has strong implications for "what occurs when," when a model is run.

There are 5 categories of chains.

(1) Current Events Chain
(2) Future Events Chain
(3) Interrupt Chains
(4) Matching Status Chains
(5) User-Defined Chains (User Chains)

As the prose suggests, there is only one Current and one Future Events Chain. In general, there is more than one Interrupt Chain, Matching Status Chain, and User Chain.

The Current Events Chain is composed of all Transactions which are scheduled to be moved through one or more Blocks at the current instant in simulated time, or as soon as possible. As the italicized phrase indicates, included on the Current Events Chain are those Transactions which are experiencing a blocking condition in the model.(2m) For example, a Transaction might be temporarily blocked because it is scheduled to move into a SEIZE Block, but the Facility it wants is already in a state of capture.

The Future Events Chain consists of those Transactions not scheduled to move through one or more Blocks until some future time. This condition can only result in two ways.

(1) A Transaction is at an ADVANCE Block, and is not scheduled to attempt to move to the next sequential Block until a later time.

(2) A Transaction has been scheduled to enter a model at some future time via a GENERATE Block.

Actually, a third situation can lead to inclusion of Transactions on the Future Events Chain, but only in a highly specialized context that need not be considered here.

Suppose now that a GPSS model is under discussion. Briefly consider the question, where is a particular Transaction which is "in the model?" Note that the answer can be given from two entirely different points of view.

(1) From the Block Diagram point of view, the Transaction is at a particular Block in the model.

(2) From the point of view of chains, the Transaction is on a particular chain.

That is, a Transaction is simultaneously "at a Block" and "on a chain." Whether a "Block-oriented" or "chain-oriented" answer is given to the above question depends on the context in which a Transaction is being discussed. The distinction between the two different orientations should be kept clearly in mind.

Transactions on the Future Events Chain are ordered according to their scheduled time of future movement. For example, assume the simulated clock currently reads 48. Then, a Transaction scheduled to move from an ADVANCE Block at time 54 is nearer the front of the Future Events Chain than another Transaction scheduled to move from the same (or another) ADVANCE Block at, say, time 59.

GPSS updates the model by scanning the Current Events Chain from front to back, Transaction by Transaction. As each next Transaction is encountered, the Processor "picks it up" and moves it forward along its scheduled path in the model until one of three situations is encountered:

(1) The Transaction moves into an ADVANCE Block where a positive holding time is computed. When this happens, the Processor then puts the Transaction on the Future Events Chain, merging it into that chain according to the time it will attempt to move to the next sequential Block.

(2) A blocking condition occurs, meaning the Transaction cannot enter some scheduled next Block. When this happens, the Transaction is left by the Processor on the Current Events Chain.

Note that the blocked Transaction may have successfully moved through several Blocks before encountering the blocking condition. The GPSS Processor will, of course, update its record of the Transaction's Block Location in the model, even though, from a chain-oriented viewpoint, the Transaction is still in its "old" location on the Current Events Chain.

(2m) At the analyst's option, blocked Transactions can be removed from the Current Events Chain and put on a User Chain. This may be done to decrease model execution time, or to implement a non-standard queue discipline, or for both of these reasons. This relatively advanced topic is discussed in Chapter 6.

(3) The Transaction moves into a TERMINATE Block. When this happens, the Processor destroys the Transaction, removing it from the model.

When a Transaction has finally stopped moving, the Processor takes one of two steps.

(1) Continuing toward the back of the Current Events Chain, it picks up the next Transaction and tries to move it forward in the model, or

(2) Without advancing the clock, it re-starts its scan of the Current Events Chain. Re-starting the scan means the Processor returns to the front of the chain, picks up the first Transaction, and moves it forward in the model if possible. When that Transaction stops moving, the Processor again takes one of the two steps now being described, and so on.

The scan is re-started only under very special conditions. The conditions depend on which Block subroutines were executed by the Transaction which has just stopped moving. When a Transaction moves through a SEIZE or a RELEASE Block, the scan is re-started after that Transaction comes to rest.[2n] The rationale of the scan re-start when a RELEASE Block has been executed is that previously-blocked Transactions may now be able to move, because a Facility has been released. The purpose of re-scanning is to process any Transactions near the front of the Current Events Chain which are in that category.

Now assume that the Transaction at the back of the Current Events Chain has just been processed. There is no "next Transaction" on that chain. (There may be other Transactions on the Current Events Chain; if so, they are toward the front of the chain, experiencing blocking conditions.) As its next step, the Processor examines the Transaction at the front of the Future Events Chain. It advances the simulation clock to the time that Transaction is scheduled to make its next move. That Transaction is then transferred from the Future to the Current Events Chain. Any other Transactions scheduled to make their move at the new clock reading are also transferred from the Future to the Current Events Chain. Each incoming Transaction is merged into the Current Events Chain according to its Priority Level. The higher the Priority Level, the closer to the front of the Current Events Chain is the Transaction. In case of ties, each newcomer is hooked onto the Current Events Chain as the last member in its Priority Class.

After the transfer of Transactions from the Future to the Current Events Chain is complete, the Processor begins its scan of the Current Events Chain anew. The basic cycle is then repeated. It is in this overall fashion that the Processor moves the model of a system forward in simulated time.

The various steps taken by the Processor, as just described, are summarized in diagrammatic form in Figures 2.23 and 2.24. Figure 2.23 shows the Clock Update Phase. Figure 2.24 shows the Scan Phase. The several paragraphs in this section should be re-read, and the correspondence between Figures 2.23 and 2.24, and the prose description of the Processor's steps, should be noted.

The major segments of the Processor's logic have now been described. They will be brought into sharper focus by going through the details of a numeric example, using the one-line, one-server case as the system being modeled.

2.21 A First Example of the Use of Current and Future Events Chains

An example showing how the GPSS Processor uses the Current and Future Events Chains will now be given for the one-line, one-server model in Figure 2A.1. This sequence will be followed in presenting the example.

[2n] Execution of certain other Blocks not yet studied also leads to scan re-starts in this fashion.

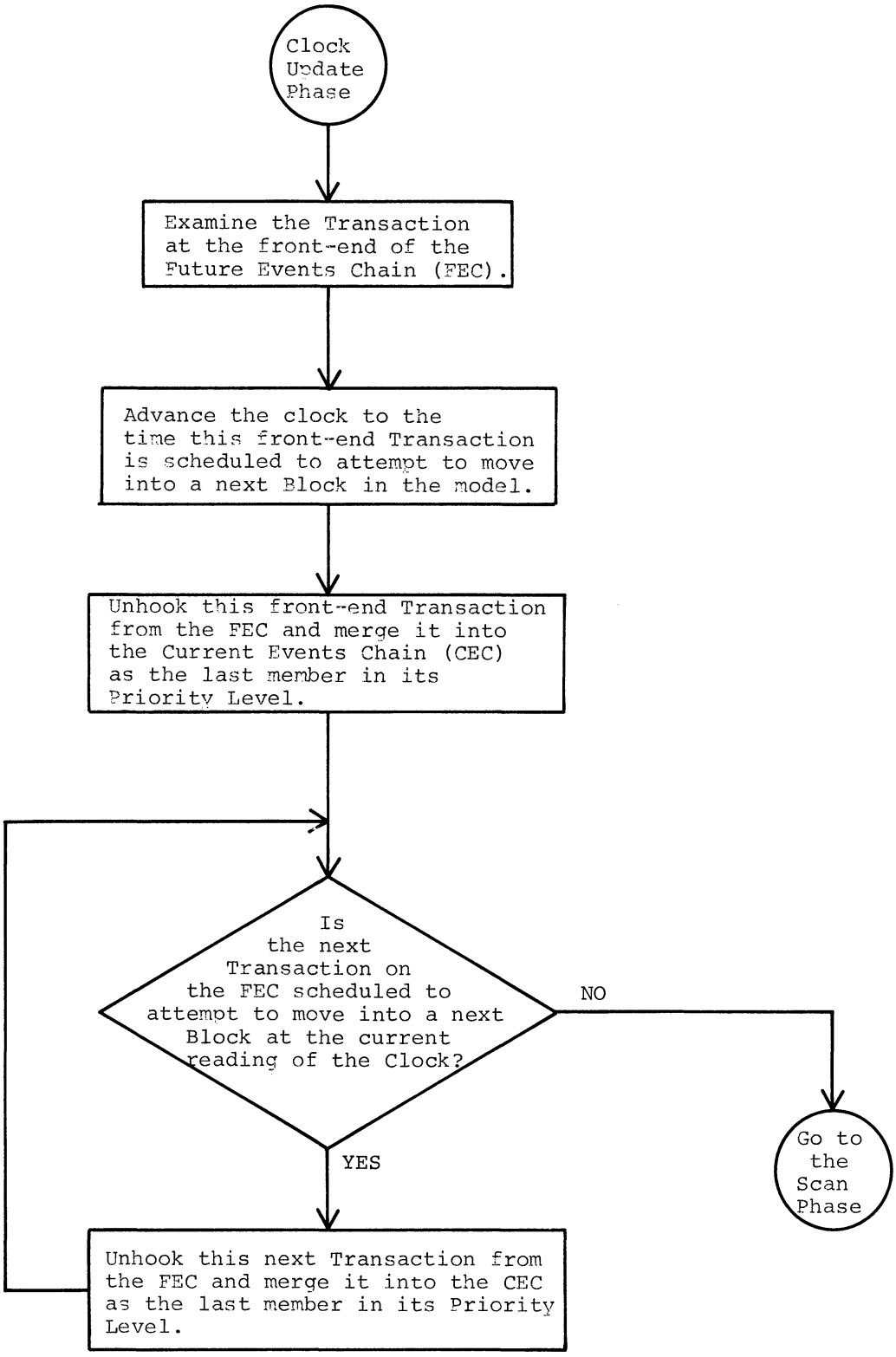


Figure 2.23 GPSS Processor's Clock Update Phase

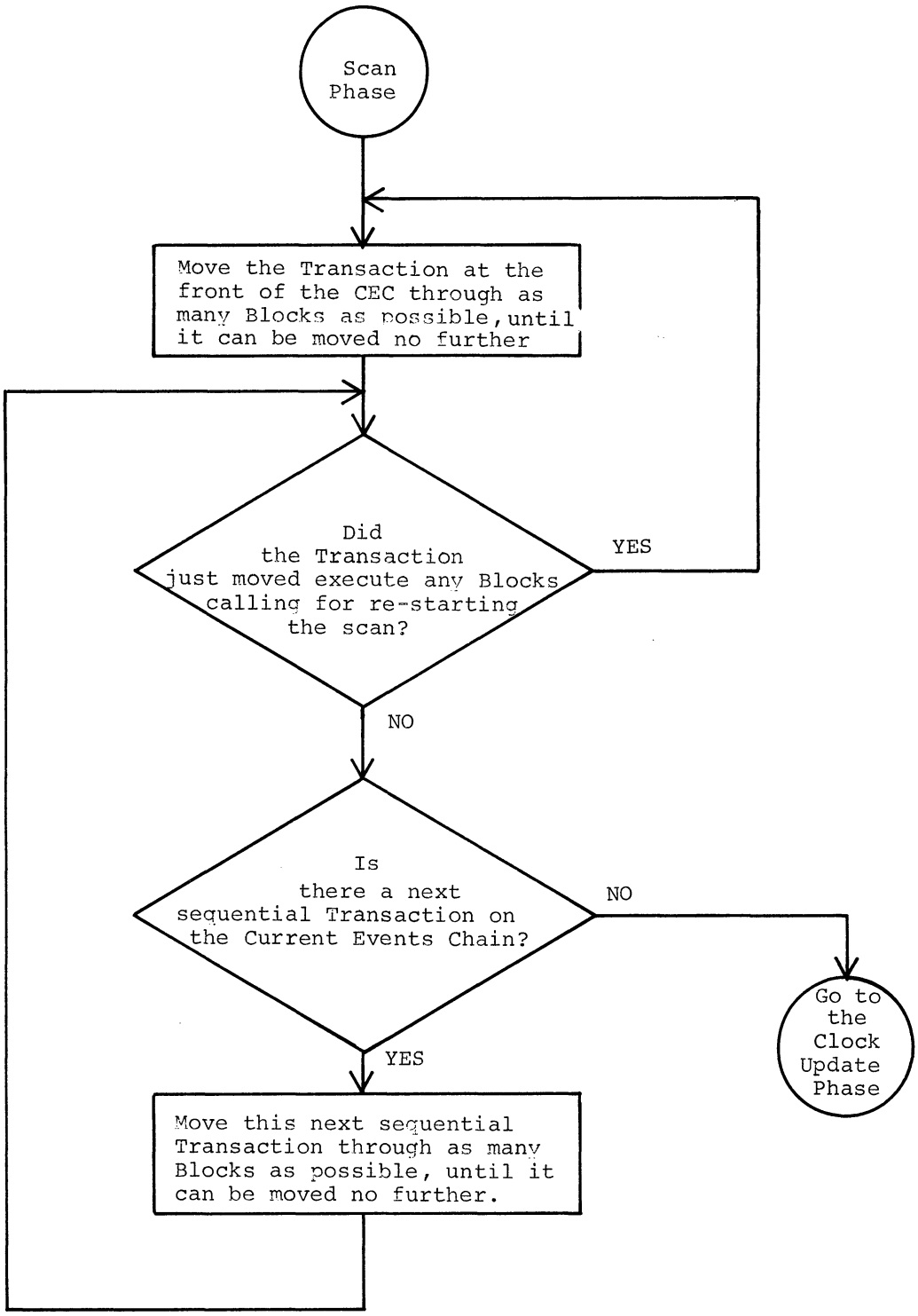


Figure 2.24 GPSS Processor's Scan Phase

(1) A notation will be adopted for recording pertinent information about Transactions.

(2) A sequence of inter-arrival and service times at the GENERATE and ADVANCE Blocks will be assumed.

(3) A tabular display will be introduced to record the time-dependent positions Transactions occupy on the Current and Future Events Chains.

(4) The steps taken by the Processor as it manipulates Transactions on the Current and Future Events Chains will be explained, showing how the Processor updates information about Transactions as they move through a model.

2.21.1 Notation for Transactions

The four pieces of information to be recorded for each Transaction are its number ("Transaction Number"), the time it is scheduled to attempt to move into a next Block ("Move Time"), its "Priority Level," and the number of the next Block it will attempt to move into ("Next Block Attempted"). This information will be recorded in a four-tuple in the order indicated. The general appearance of such a four-tuple, then, takes the form shown in Figure 2.25(a). A numeric example is shown in Figure 2.25(b). The numeric example indicates that Transaction 5 will attempt at time 68 to move into Block 6. Transaction 5 has a Priority Level of 0.

[Transaction Number, Move Time, Priority Level, Next Block Attempted]

(a) General Form

[5,68,0,6]

(b) Numeric Example

Figure 2.25 Transaction Information As Recorded in Four-Tuples

As suggested above, the GPSS Processor assigns numbers to Transactions. The range of feasible Transaction numbers depends on how many Transactions can simultaneously exist in a model. This quantity is determined by the amount of computer memory available. At a memory level of 64K bytes, the maximum normal quantity of Transactions is 200. At this level, the Transaction numbers then range from 1 to 200.

Transactions belong to either one of two groups. One group is the "latent pool" of Transactions, those not currently "in existence" in a model. The other group is the "active pool" of Transactions, those which have entered the model through one or more GENERATE Blocks,[2oh] and have not yet been removed from the model.

Before a simulation starts, Transactions in the latent pool are ordered by increasing Transaction number. In a 64K model, Transactions in the latent pool therefore initially form a list in which the Transactions are ordered 1,2,3,4,5,...,198,199,200. This list can be thought of on a "top-to-bottom" basis. Transaction 1 is at the top of the list, Transaction 2 is behind it, and so on, until finally Transaction 200 is at the bottom of the list.

[2oh] There is one other Block at which Transactions can be created as well. Discussion of that Block is deferred until Chapter 4.

When conditions call for scheduling the entry of a Transaction into the model, the Processor fetches the Transaction at the top of the list from the latent pool. This Transaction is then brought into the model, via the Future Events Chain, by a procedure to be described below. Conversely, when model conditions call for elimination of a particular Transaction, it is removed from the model and put back into the latent pool at the top of the list. The movement of Transactions between the latent and active pools is suggested in Figure 2.26.

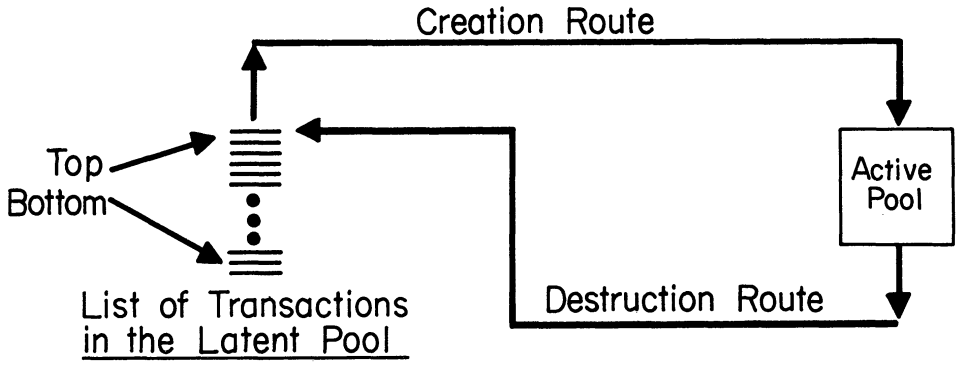


Figure 2.26 Transaction Movement Between the Latent and Active Pools

As for Block Numbers, the steps followed by the Processor in assigning Numbers to Blocks was discussed in Section 2.5. The Block Numbers are assigned in the order in which the cards for the various Block images have been placed in the card deck. For convenience, Figure 2.27 is a repetition of the extended program listing in Figure 2A.2, with the Processor-assigned Block Numbers shown in the leftmost column.

```
BLOCK                                                                CARD
NUMBER  *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS           NUMBER
               SIMULATE                                                1
        *                                                              2
        *      MODEL SEGMENT 1                                         3
        *                                                              4
1              GENERATE   18,6          CUSTOMERS ARRIVE               5
2              QUEUE      JOEQ          ENTER THE LINE                 6
3              SEIZE      JOE           CAPTURE THE BARBER             7
4              DEPART     JOEQ          LEAVE THE LINE                 8
5              ADVANCE    16,4          USE THE BARBER                 9
6              RELEASE    JOE           FREE THE BARBER               10
7              TERMINATE                 LEAVE THE SHOP                11
        *                                                             12
        *      MODEL SEGMENT 2                                        13
        *                                                             14
8              GENERATE   480           TIMER ARRIVES AT TIME 480     15
9              TERMINATE  1             SHUT OFF THE RUN              16
        *                                                             17
        *      CONTROL CARDS                                          18
        *                                                             19
               START      1             START THE RUN                 20
               END        RETURN CONTROL TO OPERATING SYSTEM          21
```

Figure 2.27 Repetition of Figure 2A.2

2.21.2 Assumed Inter-Arrival and Service Times

Table 2.7 contains the sequence of sample values assumed to result when the GPSS Processor samples the first four times from the 18±6 distribution at the Block 1 GENERATE Block.

Number of Sample	Sampled Value
1	14
2	13
3	17
4	15

Table 2.7 Assumed Inter-Arrival Times at Block 1 in Figure 2.27

Table 2.8 shows the sample values assumed to result when the Processor samples the first three times from the 16±4 distribution at the Block 5 ADVANCE Block.

Number of Sample	Sampled Value
1	18
2	12
3	14

Table 2.8 Assumed Holding Times at Block 5 in Figure 2.27

2.21.3 Chain History for the Figure 2.27 Model

Figure 2.28 displays the Current and Future Events Chains for the first few clock readings as the Figure 2.27 model is moved through simulated time. Each line of information in Figure 2.28 has been given a number (column one). This has been done so specific lines can be referred to during the discussion of the figure. The various times registered by the clock as the simulation proceeds are also shown in Figure 2.28, in column two. The third column shows the Current Events Chain. The fourth column shows the Future Events Chain. Viewing a chain from left-to-right in the figure corresponds to examining it from front-to-back.

Encoding information in four-tuples, the Transactions which are chain residents are shown in Figure 2.28, occupying consecutive positions in a row in columns three and four. From time to time, a chain may have no Transactions on it. This condition is indicated by use of the word "Empty."

Line Number	Simulated Time	Current Events Chain Front of Chain ←	Future Events Chain Front of Chain ←
1	Before Input Phase	Empty	Empty
2	After Input Phase	Empty	[1,14,0,1] [2,480,0,8]
3	14	[1,ASAP,0,1]	[2,480,0,8]
4	14	Empty	[3,27,0,1] [1,32,0,6] [2,480,0,8]
5	27	[3,ASAP,0,1]	[1,32,0,6] [2,480,0,8]
6	27	[3,ASAP,0,3]	[1,32,0,6] [4,44,0,1] [2,480,0,8]
7	32	[3,ASAP,0,3] [1,ASAP,0,6]	[4,44,0,1] [2,480,0,8]
8	32	Empty	[4,44,0,1] [3,44,0,6] [2,480,0,8]
9	44	[4,ASAP,0,1] [3,ASAP,0,6]	[2,480,0,8]
10	44	Empty	[4,46,0,6] [1,59,0,1] [2,480,0,8]

Figure 2.28 Chain History for First Example of the Use of Current and Future Events Chains

Lines 1 and 2 in Figure 2.28 display chain information before and after the "model input phase" is finished. The Processor's reading-in of the model constitutes the "input phase." Beneath Lines 1 and 2, two lines of information are shown for each clock reading. The first line at any given clock value shows the chains after the most recent Clock Update Phase is finished, but before the next Scan Phase has begun. Such information, then, is the "input" to the next Scan Phase. The second line at the same clock value shows the chains after the next Scan Phase is finished, and just before the next Clock Update Phase is begun. This information is the "input" to the next Clock Update Phase.

Remember that between the first and second lines at any given clock value, the steps outlined in Figure 2.24 (Scan Phase) have been performed by the GPSS Processor. In general, many "intermediate moves" are made by Transactions as those steps are performed. The Figure 2.28 history does not show those intermediate moves, one-by-one. Instead, in the second line at each time frame, the aggregate consequence of all those intermediate moves is displayed.

Similarly, between the second line at any given clock value, and the first line at the next clock value, the steps outlined in Figure 2.23 (Clock Update Phase) have been performed by the Processor. Again, no attempt is made in Figure 2.28 to show how the chains change, step-by-step, as the Clock Update Phase proceeds. Instead, the first line at each time frame shows the chains after the Clock Update Phase is finished.

The steps followed by the Processor to produce the chain history in Figure 2.28 will now be explained.

2.21.4 Explanation of the Figure 2.28 Chain History

Input Phase (From Line 1 to Line 2)

The first thing the Processor does is input the model. Before this input phase begins, the Current and Future Events Chains are empty (Line 1, Figure 2.28). During the input phase, the Processor examines each card as it is read to determine if it is a GENERATE card. When a GENERATE card is read, a sample is drawn from the inter-arrival time distribution described by the A and B Operands. The Processor then pre-schedules the first arrival at the corresponding GENERATE Block by fetching the Transaction from the top of the list in the latent pool and hooking it onto the Future Events Chain, scheduled to enter the GENERATE Block at a time equal to the sampled value.(2p)

For the Figure 2.27 model, when the Block 1 GENERATE card is read during the input phase, a sample is drawn from the 18±6 population. This first sampled value, via Table 2.7, is 14. The Processor then fetches Transaction 1 from the latent pool and hooks it onto the Future Events Chain, scheduled to enter Block 1 at time 14.

As the input phase continues, the Block 8 GENERATE card is eventually read. For this Block, a deterministic inter-arrival time of 480 is in effect. The Processor fetches Transaction 2 from the latent pool and hooks it onto the FEC, scheduled to enter Block 8 at time 480. Because the Processor orders Transactions on the FEC according to their Move Time, Transaction 2 is put behind Transaction 1 on the Future Events Chain.

(2p) If the value drawn from an inter-arrival time distribution during the input phase is zero, it is automatically re-defined by the GPSS Processor to be 1.

After the Block 9 TERMINATE card is read, the input phase is finished. Examination of the Future Events Chain (Line 2, Figure 2.28) now shows two Transactions on it, one for each GENERATE Block in the model. Transaction 1 represents the first customer, on his way to the barber shop. He will arrive at the shop at simulated time 14. The scheduled movement of Transaction 1 into Block 1 is analogous to the customer's arrival at the door of the shop. Transaction 2 is the timer, on his way to shut off the model. When the simulation clock eventually reaches a value of 480, the timer will "arrive" and, moving into the TERMINATE Block, cause the simulation to shut off.

With the input phase complete, the Processor now goes into the Clock Update Phase. After this first clock updating, the Scan Phase will be performed for the first time. Then the Clock Update Phase will be performed a second time, the Scan Phase will be performed a second time, the Clock Update Phase will be performed a third time, and so on.

First Clock Update Phase (From Line 2 to Line 3)

The Processor first sets the clock at 14, the Move Time of the Transaction (Transaction 1) at the front of the Line 2 Future Events Chain. It then transfers Transaction 1 from the Future Events Chain to the previously-empty Current Events Chain. The next Transaction (Transaction 2) on the FEC does not have a Move Time of 14. The first execution of the Clock Update Phase is therefore finished.

Note then, at Line 3 in Figure 2.28, that the Current Events Chain contains the single Transaction designated [1,ASAP,0,1]. The time entry in the four-tuple is "ASAP," for "As Soon As Possible." All Transactions on the CEC have ASAP as their Move Time. This is because they want to move into their next Block now or, in the event the Block will not now accept them, as soon as possible.

With the first Clock Update Phase complete, the first execution of the Scan Phase can now begin.

First Scan Phase (From Line 3 to Line 4)

Picking up Transaction 1 from the front-end of the Line 3 CEC, the Processor moves it into Block 1 (the GENERATE Block), then immediately attempts to move it into Block 2 (the QUEUE Block). The QUEUE Block cannot refuse entry, so the attempted move is successful.

Now, because a Transaction has just successfully exited a GENERATE Block, the Processor temporarily stops moving it and pre-schedules the arrival of a next Transaction at that GENERATE Block. Sampling for the second time from the 18±6 inter-arrival time distribution, a sample value of 13 is obtained per Table 2.7. The Transaction at the top of the list in the latent pool (Transaction 3) is fetched and hooked onto the Future Events Chain, scheduled to move to Block 1 at future time "now + 13," that is, at time 27. Note carefully that the sampled inter-arrival time value is added to the clock's current value to determine the arrival time of the next Transaction.

As to the operation of GENERATE Blocks, two points should be noted here:

(1) The Processor does not pre-schedule the next arrival at a GENERATE Block until the preceding Transaction has successfully moved to the next Block. If the next Block is of a type which can refuse entry (such as a SEIZE Block), the pre-scheduling step might not take place until some time after a Transaction has arrived at the GENERATE Block.

(2) The Processor interrupts movement of the Transaction exiting the GENERATE Block while it pre-schedules its successor's arrival. After the pre-scheduling is finished, the forward movement of the exiting Transaction is re-initiated.

Figure 2.29 illustrates these two points in the form of a flowchart.

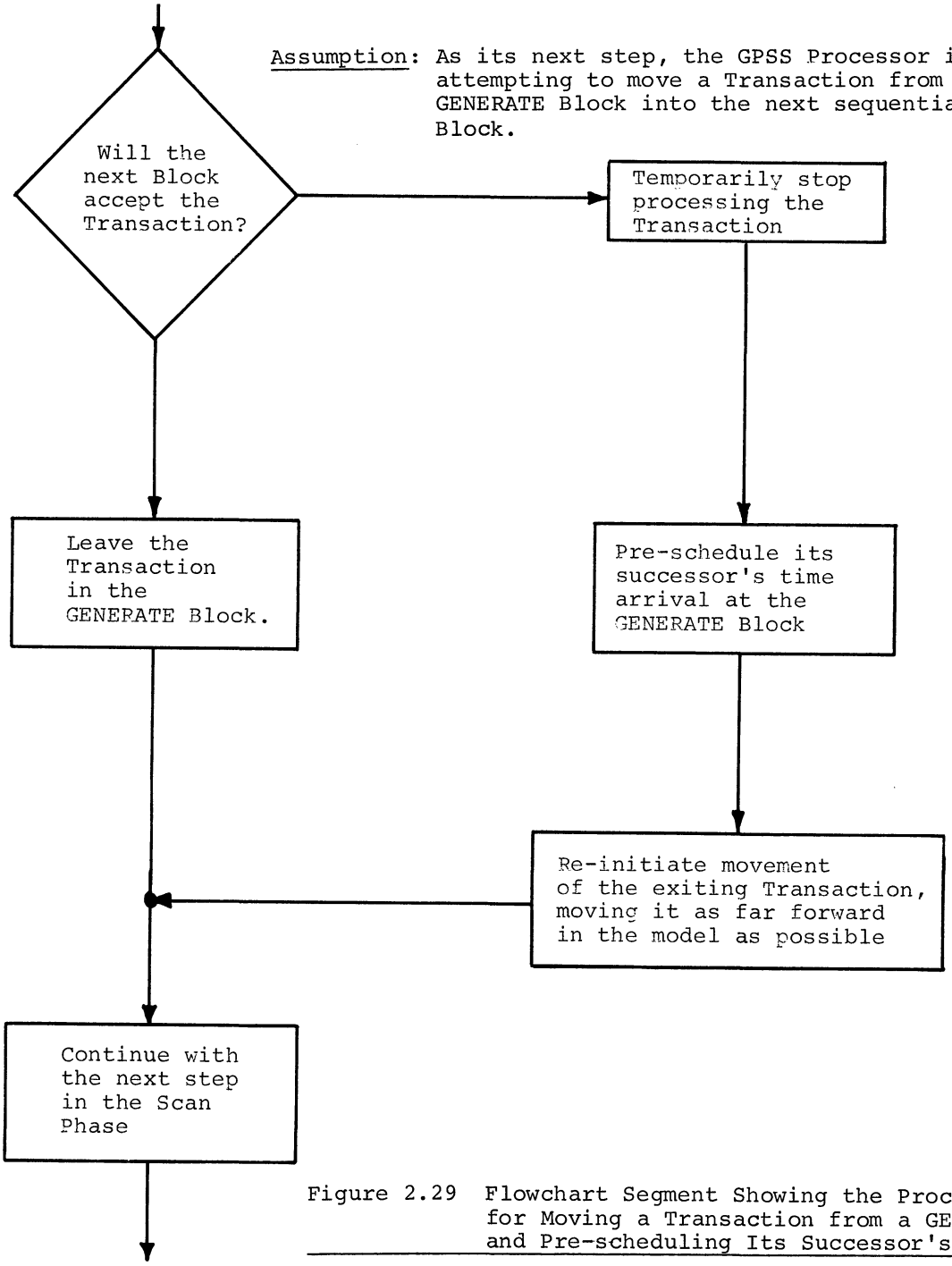


Figure 2.29 Flowchart Segment Showing the Processor's Logic for Moving a Transaction from a GENERATE Block, and Pre-scheduling Its Successor's Arrival

With pre-scheduling completed, Transaction 1 now executes the QUEUE Block subroutine. From the QUEUE Block, the Processor then successfully moves the Transaction into the SEIZE, DEPART, and ADVANCE Blocks. At the ADVANCE, a holding time is determined by sampling from the 16±4 distribution. Per Table 2.8, the first sampled value is 18. Transaction 1 is therefore hooked onto the FEC, scheduled to move into Block 6 (the RELEASE Block) at future time "now + 18," that is, at time 32.

Now, because Transaction 1 caused a SEIZE Block to be executed as part of its movement, the Processor re-starts its scan of the Current Events Chain. But the chain is empty. The next step, then, is to perform the Clock Update Phase.

Before continuing, note that the Transactions on the Future Events Chain at Line 4, considered in left-to-right order, have the significance shown in Table 2.9.

Transaction Number	Significance
3	Second customer, on his way to the shop
1	First customer, having his hair cut
2	The timer Transaction

Table 2.9 Significance of Transactions on Future Events Chain, Line 4, Figure 2.28

Second Clock Update Phase (From Line 4 to Line 5)

The Processor advances the clock to 27, the Move Time of the Transaction (Transaction 3) at the front of the Line 4 Future Events Chain. It then transfers Transaction 3 from the FEC to the previously-empty CEC. The next Transaction on the FEC (Transaction 1) does not have a Move Time of 27. The second execution of the Clock Update Phase is therefore finished. The next Scan Phase execution can begin.

Second Scan Phase (From Line 5 to Line 6)

Picking up Transaction 3 from the front-end of the Line 5 Current Events Chain, the Processor moves it into Block 1 (the GENERATE Block), then determines that it can immediately move into Block 2. Temporarily interrupting the processing of Transaction 3, the Processor then pre-schedules the arrival of a next Transaction at the Block 1 GENERATE Block. Corresponding to an inter-arrival time of 17 (third sample drawn from the 18±6 distribution, per Table 2.7), Transaction 4 is fetched from the latent pool and hooked onto the FEC, scheduled to move to Block 1 at time 44. It occupies a position on the FEC between Transactions 1 and 2, per the FEC ordering criterion.

The Processor now resumes moving Transaction 3, completing execution of the QUEUE Block subroutine. Transaction 3 then unsuccessfully attempts to move into the SEIZE Block. The Facility JOE is in a state of capture, so entry is denied. The Processor therefore leaves Transaction 3 on the CEC, scheduled to enter Block 3 "As Soon As Possible."

With Transaction 3 at rest, the Processor continues to the next Transaction on the Line 5 Current Events Chain. But there is no next Transaction there. The next step, then, is to perform the Clock Update Phase. Note that the clock is about to be advanced even though the Current Events Chain is not empty.

Before proceeding, take note that the Transactions at Line 6 in Figure 2.28 have the significance shown in Table 2.10

Chain	Transaction Number	Significance
CEC	3	Second customer, waiting for the barber
FEC	1	First customer, having his hair cut
FEC	4	Third customer, on his way to the shop
FEC	2	The timer Transaction

Table 2.10 Significance of Transactions on Current and Future Events Chains, Line 6, Figure 2.28

Third Clock Update Phase (From Line 6 to Line 7)

The Processor advances the clock to 32, the Move Time of the Transaction (Transaction 1) at the front of the Line 6 Future Events Chain. It then transfers Transaction 1 to the Current Events Chain, where it is merged in as the last member in its Priority Class. In the Line 7 CEC, note that Transaction 1 is behind Transaction 3. (Transaction 3 was already on the Current Events Chain when the third execution of the Clock Update Phase began.)

The next Transaction on the FEC (Transaction 4) does not have a Move Time of 32. The next execution of the Scan Phase can consequently begin.

At this point, consider the significance of the two Transactions on the Line 7 Current Events Chain, as indicated in Table 2.11.

Transaction Number	Significance
3	Second customer, waiting for the barber
1	First customer, just at the point of having his haircut completed

Table 2.11 Significance of Transactions on Current Events Chain, Line 7, Figure 2.28

Third Scan Phase (From Line 7 to Line 8)

Picking up Transaction 3 at the front-end of the Line 7 CEC, the Processor fails in its attempt to move it into Block 3 (SEIZE). Entry to Block 3 is denied, of course, because the Facility is still engaged. Transaction 3 consequently remains on the CEC, still scheduled to enter Block 3 as soon as possible. Continuing to the next Line 7 CEC Transaction (Transaction 2), the Processor moves it into Block 6 (RELEASE), and from there into Block 7 (TERMINATE), where it is removed from the model and returned to the top of the list of Transactions in the latent pool.

Now, because Transaction 1 caused a RELEASE Block to be executed, the Processor re-starts the scan of the Current Events Chain. Transaction 3 is picked up again, and attempts again to move into Block 3. This time, the attempt is successful. The SEIZE-ADVANCE-RELEASE sequence follows, and a holding time of 12 is drawn from the 16±4 distribution (second sample, Table 2.8). Transaction 3 is then hooked onto the FEC, scheduled to enter Block 6 at time "now + 12", or 44. Note that, on the FEC, Transaction 3 is involved in a time-tie with Transaction 4, which also has a Move Time of 44. Because Transaction 4 was already on the FEC, Transaction 3 is put behind it. Again, whenever time-ties occur on the FEC, the incoming Transaction is merged onto the chain as the last member in its Move Time category.

Now, because Transaction 3 caused a SEIZE Block to be executed, the Processor re-starts its scan of the Current Events Chain. But the chain is empty. The next step, then, is to perform the Clock Update Phase.

At this point, Transactions on the Future Events Chain in Line 8 have the significance indicated in Table 2.12.

Transaction Number	Significance
3	Second customer, having his hair cut
4	Third customer, on his way to the shop
2	The timer Transaction

Table 2.12 Significance of Transactions on the Future Events Chain, Line 8, Figure 2.28

Fourth Clock Update Phase (From Line 8 to Line 9)

The Processor advances the clock to 44, the Move Time of the Transaction (Transaction 4) at the front of the Line 8 Future Events Chain. It then transfers Transaction 4 from the FEC to the previously-empty CEC. The next Transaction on the FEC (Transaction 3) also has a Move Time of 44. It, too, is transferred to the Current Events Chain, where it is put behind Transaction 4, as the last member in its Priority Class. The next Transaction

on the Future Events Chain (Transaction 2) does not have a Move Time of 44. The fourth Clock Update Phase is therefore finished.

Before continuing, consider the significance of the two Transactions on the Line 9 Current Events Chain, as indicated in Table 2.13.

Transaction Number	Significance
4	Third customer, just arriving at the door to the shop
3	Second customer, just at the point of having his haircut completed

Table 2.13 Significance of Transactions on Current Events Chain, Line 9, Figure 2.28

Fourth Scan Phase (From Line 9 to Line 10)

Picking up Transaction 4 at the front-end of the Line 9 CEC, the Processor moves it into Block 1, then determines that it can move into Block 2. Temporarily interrupting the processing of Transaction 4, the Processor pre-schedules the arrival of its successor at the Block 1 GENERATE Block. Transaction 1 is fetched from the top of the list of Transactions in the latent pool and hooked onto the Future Events Chain, scheduled to move to Block 1 at time "now + 15", or 59. (Per Table 2.7, the fourth sampled value from the 18±6 distribution is 15.)

Note that it is Transaction 1 which is coming back into the model from the latent pool. The Transaction had previously been in the model, simulating the first customer of the day; it now comes back into play, simulating the day's fourth customer. Because the Processor always takes Transactions from and returns them to the top of the list in the latent pool, it should now be clear that some Transactions may "live many lives" during the course of a simulation.

With its movement resumed, Transaction 4 then finds it must remain in Block 2 (QUEUE), because the SEIZE Block will not accept it. The Processor then moves the next Line 9 CEC Transaction (Transaction 3) through the RELEASE-TERMINATE sequence, returning it to the latent pool. Then, because a RELEASE Block was executed, the Processor re-starts its scan of the Current Events Chain. Transaction 4 is picked up again and moved through the SEIZE-DEPART-ADVANCE sequence. Per Table 2.8, the third value sampled from the 16±4 distribution at the ADVANCE Block is 14. Transaction 4 is therefore hooked onto the Future Events Chain, scheduled to move to Block 6 at time 46.

Now, because a SEIZE Block was executed, the Processor re-starts its scan of the Current Events Chain. But the chain is empty. The next step, then, is to execute the Clock Update Phase for the fifth time.

This completes the explanation of the chain history shown in Figure 2.28.

2.22 Exercises

2.22.1 (General Questions Based on Sections 2.20 and 2.21)

(a) What is "Move Time"?

(b) What information is provided by the fourth entry in a "Transaction four-tuple"?

(c) How many Current and Future Events Chains are there?

(d) How are Transactions ordered on the Future Events Chain?

(e) In what sense can ties occur on the Future Events Chain? When ties do occur, how are they resolved?

(f) How are Transactions ordered on the Current Events Chain?

(g) In what sense can ties occur on the Current Events Chain? When ties do occur, how are they resolved?

(h) When the Clock Update Phase begins, how does the GPSS Processor determine the time of the next future event in the model?

(i) After the simulation clock is advanced, how many Transactions are unhooked from the Future Events Chain and brought to the Current Events Chain?

(j) Under what conditions is a Transaction removed from the Current Events Chain and hooked back onto the Future Events Chain?

(k) What is the distinction between the "latent pool" and the "active pool" of Transactions in a model?

(el) How are Transactions ordered in the latent pool?

(m) Why is it never necessary to resolve ties relative to Transaction ordering in the latent pool?

(n) How is it possible for a Transaction to "live many lives" during the course of a simulation run?

(oh) Critically discuss this statement: "After each updating of the simulation clock, the GPSS Processor scans the Current Events Chain exactly one time, then proceeds with the next updating of the clock".

(p) Why is it that the Current Events Chain is not necessarily empty when the Processor, as its next move, performs the Clock Update Phase?

(q) Critically discuss this statement: "After the model input phase is complete, there is exactly one Transaction on the Future Events Chain for every GENERATE Block in the model".

(r) Critically discuss this statement: "The smallest Move Time that a Transaction can ever have is 1".

(s) Is it true that the Current Events Chain is always empty when the input phase has just been completed?

2.22.2 (Specific Questions Based on the Section 2.21 Example of Chain Usage)

(a) What is the Priority Level of all Transactions in the Figure 2.27 model? Why is this Priority Level the one in effect?

(b) Will Transaction 2 ever come into play in the model as a customer at the barber shop? Why or why not?

(c) Extend Figure 2.28 to include Lines 11, 12, 13, and 14. In doing this, assume that the next inter-arrival time sampled at Block 1 is 16, and the next service time sampled at Block 5 is 17.

Now answer these questions.

(1) What is the clock reading at lines 11 and 12?

(2) What is the clock reading at lines 13 and 14?

(3) What is the number of the Transaction representing the fourth customer?

(4) How much idle time does the barber experience between finishing the third customer, and beginning work on the fourth customer?

(5) What is the number of the Transaction simulating the fifth customer?

(6) When will the fifth customer arrive at the barber shop?

(7) How long will the fifth customer have to wait for service, if at all?

2.22.3 Consider the model in Figure 2.27. Assume that the Block 1 and Block 5 Operands are "30,25" and "18,3", respectively, and that the first portions of the inter-arrival and holding time sequences actually realized in a particular simulation are as shown below.

Inter-arrival Time Sequence: 7,10,3,6,49,7, . . .

Service Time Sequence: 19,15,12,15,21, . . .

Using the approach illustrated in Section 2.21, track the progress made by Transactions through the model in terms of their residence on the Current and Future Events Chains. Then answer these questions:

(a) What value is registered by the simulation clock at Line 10 in the chain history?

(b) What is the number of the fourth Transaction to engage the barber?

(c) Are there ever exactly two Transactions on the Current Events Chain <u>after</u> a Scan Phase completion? What are the numbers of the two Transactions involved when and if this condition arises?

2.22.4 Referring to Figure 2.29, discuss the difference between the two Block Diagram segments in Figure P2.22.3.

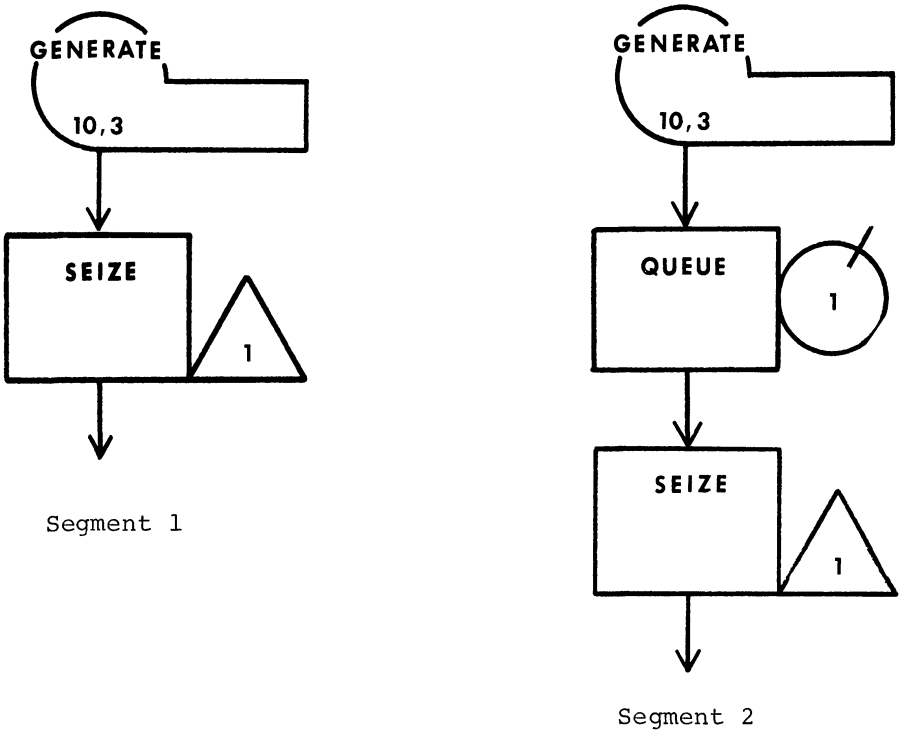


Figure P2.22.3

2.22.5 (a) Assume that the QUEUE-DEPART Block pair is removed from the model in Figure 2.27. Using the inter-arrival and service time sequences shown in Tables 2.7 and 2.8, state the times at which the second, third, and fourth customers of the day arrive at the barber shop.

(b) Suppose that an analyst does not want to collect Queue data for the system modeled in Figure 2.27. As (a) above demonstrates, he cannot simply remove the QUEUE-DEPART Block pair from the model, because this produces a distortion in the inter-arrival time sequence he wants to be in effect. Show how he can avoid the distortion by introducing an ADVANCE Block between the GENERATE and SEIZE Blocks.

2.22.6 Consider the model in Figure P2.22.6.

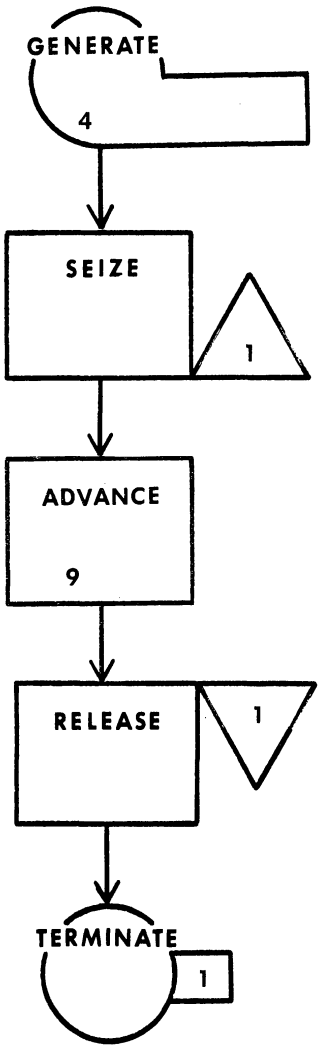


Figure P2.22.6

Assume that the model is run, and that 4 is entered as the A Operand on the START Card.

(a) At what time does Transaction 1 arrive at the GENERATE Block?
(b) At what time does its successor arrive at the GENERATE Block?
(c) At what time does its successor exit the GENERATE Block?
(d) At what time does Transaction 3 arrive at the GENERATE Block?

(e) When is Facility 1 seized for the third time?

(f) How much time elapses between existing of successive Transactions from the GENERATE Block?

(g) At what time does the simulation run shut off?

2.22.7 Consider the two Block Diagram segments in Figure P2.22.7.

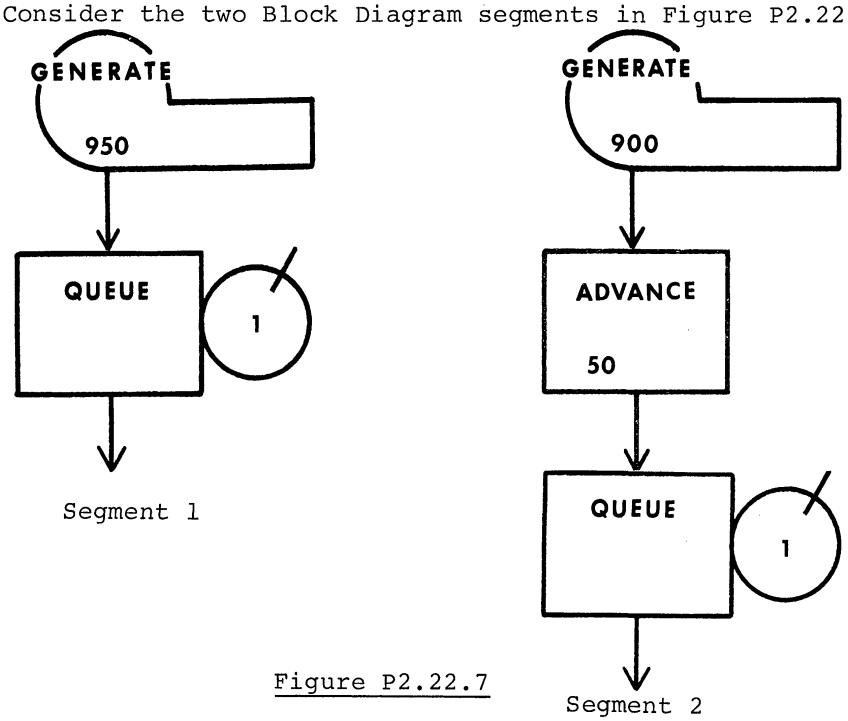


Figure P2.22.7

(a) At what times do the first three Transactions in Segment 1 move into the QUEUE Block?

(b) At what times do the first three Transactions in Segment 2 move into the QUEUE Block?

2.22.8 In a certain model, Transactions represent planes arriving at an airport. Upon arriving, they join the Queue STACK, which simulates planes stacked over the airport, waiting their turn to land. Planes are scheduled to to arrive at the airport every 900 time units, but they may be 50±50 time units late. Consecutive arrivals are independent of each other. That is, the delay possibly experienced by an arriving plane is not to influence the arrival time of the next plane.

Show a model segment simulating the planes arriving at the airport and joining the Queue.

2.23 Model-Produced Printouts of Current and Future Events Chains

In some treatments of GPSS, the topic of Chains is deferred until quite late, or is not covered at all. It is true that a person can "do a lot of modeling" in GPSS without being aware of the underlying chain concepts. It is also true, unfortunately, that a person can "do a lot of invalid modeling" when the chain concepts are not taken into account. When modeling systems that are even only somewhat complex, the analyst inevitably comes upon questions which take the form, "what happens next in the model?" Because events can occur simultaneously in real systems but are caused to occur sequentially in a GPSS model, the question of "what happens next" in GPSS can be of the utmost importance. An explanation of what happens next can always be given in terms of chain concepts. The analyst

unfamiliar with these concepts must either content himself with vague answers to the "what next" questions, or ignore the questions entirely and simply hope that the model will be valid, or approximately valid, anyway. This latter approach, though expedient, is hardly acceptable.

Rather than ignoring chains, the analyst familiar with them will find it convenient to think in terms of them in coming to a better understanding of the behavior of his model. From time to time, he may also want to obtain a printout of the chains to reinforce his conclusions about how a particular model is operating. Chain printouts can be obtained in these several different ways.

(1) When certain error conditions are encountered during execution of a model, the GPSS Processor automatically prints out the Current and Future Events Chains for possible use by the analyst in "debugging" his model.

(2) At the end of a simulation run, the GPSS Processor will include chain printouts as part of its standard output if 1 has been punched as the D Operand on the START card.

(3) During a simulation, chain printouts can be produced at whatever simulated times the analyst desires. This effect is only accomplished, however, when the analyst makes explicit arrangements for it in the logic of his model. The results constitute a series of "snapshots" of the chains as the model is moved forward in simulated time.

It is of interest now to see "what chains look like" when they are printed out by the GPSS Processor. Suppose the approach mentioned in (2) above is used to obtain such a printout with the Figure 2A-2 model. All that is involved is entering a 1 as the D Operand on the START card, and re-submitting the model for running. The resulting output is shown in Figures 2.30 and 2.31, where the Current and Future Events Chains are shown, respectively.

In Figures 2.30 and 2.31, columns containing the information carried in the "Transaction four-tuples" previously introduced in Section 2.21 have been labeled A, B, C, and D. In particular, columns A, B, C, and D contain "Transaction Number", "Move Time" (i.e., Block Departure Time), "Priority Level", and "Next Block Attempted", respectively. The significance of this information is summarized in Figure 2.30. The other columns of information in Figures 2.30 and 2.31 will not be explained now.

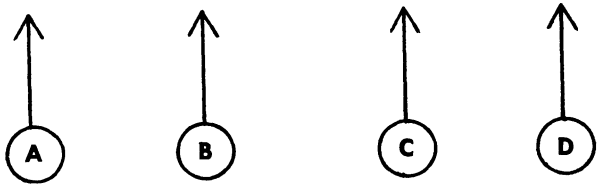
Consider the two Transactions shown on the Current Events Chain in Figure 2.30. Remember that Figure 2.30 shows the appearance of the chain at simulated time 480, when the simulation shut off. "Interpretations" are given for the two Transactions in Table 2.14, in their order of appearance on the chain.

It might be noted that, on the Current Events Chain, the Processor does not "ASAP" in the "Move Time" column. The ASAP designation was introduced in Section 2.2 only to assist in the explanation.

Now consider the two Transactions shown on the Future Events Chain in Figur 2.31. Interpretations are given for the two Transactions in Table 2.15, in their ord of appearance on the chain.

CURRENT EVENTS CHAIN

TRANS	BDT	BLOCK	PR	SF	NBA	SET	MARK-TIME	P1	P2	P3	P4	SI	TI	DI	CI	MC	PC	PF
4	472	2			3	4	472	0	0	0	0	1		1	2			
								0	0	0	0							
								0	0	0	0							
3	480	5			6	3	453	0	0	0	0				2			
								0	0	0	0							
								0	0	0	0							



Column	Significance
A	Transaction Number
B	Block Departure Time (BDT); same as time of attempted move into next Block
C	Priority Level (PR); "blank" entries indicate a Priority Level of 0
D	Next Block Attempted (NBA); number of the next Block the Transaction will attempt to enter

Figure 2.30 Current Events Chain When the Figure 2A.2 Model Shuts Off

FUTURE EVENTS CHAIN

TRANS	BDT	BLOCK	PR	SF	NBA	SET	MARK-TIME	P1	P2	P3	P4	SI	TI	DI	CI	MC	PC	PF
1	489				1	1	-27	0	0	0	0				4			
								0	0	0	0							
								0	0	0	0							
5	960				8	5	-1	0	0	0	0				4			
								0	0	0	0							
								0	0	0	0							

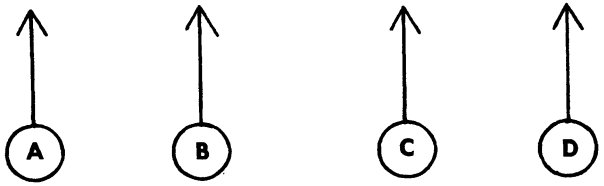


Figure 2.31 Future Events Chain When the Figure 2A.2 Model Shuts Off

(Significance of Columns A, B, C, and D is Shown in Figure 2.30)

Transaction Number	Transaction Information As A Four-Tuple	Significance
4	[4,472,0,3]	Transaction 4 is a customer at the barber shop, waiting for the barber. Its "Next Block Attempted" is Block 3 (SEIZE). The barber is busy, servicing Transaction 3. (See the significance of Transaction 3, below.)
3	[3,480,0,6]	Transaction 3 is a customer just about to finish having his hair cut. Note that it is poised to move into Block 6 (RELEASE) at time 480, the time of model shutoff. It is left as an exercise in the next section to explain why the model shut off before the RELEASE had a chance to take place.

Table 2.14 Interpretations for the Transactions in Figure 2.30

Transaction Number	Transaction Information As A Four-Tuple	Significance
1	[1,489,0,1]	Transaction 1 is the next customer on the way to the barber shop. He is scheduled to move to Block 1 (GENERATE) at time 489. He will be "9 minutes too late".
5	[5,960,0,8]	It is left as an exercise in the next section to explain the significance of Transaction 5.

Table 2.15 Interpretations for the Transactions in Figure 2.31

The way the analyst can get chain printouts during a simulation will be described in Section 2.41. For the time being, it is enough to know what a chain printout looks like, and how one can be produced at the end of a run.

2.24 Exercises

2.24.1 In Figure 2.30, why is Transaction 3 still on the Current Events Chain, not yet having moved into the RELEASE Block, when the model shuts off?

2.24.2 In Figure 2.31, what is the significance of Transaction 5 on the Future Events Chain? Why was that Transaction fetched from the latent pool and hooked onto the Future Events Chain before the model shut off?

2.24.3 In Figure 2.27, modify the A and B Operands at Blocks 1, 5, and 8 to make them "30,0", "20,0", and "90,0", respectively. Then do the following.

(a) By hand, produce a chain history analogous to Figure 2.28 for the resulting model. Note that, with the timer Transaction "arriving" at time 90, only a few time frames will be required before the model shuts off.

(b) Submit the modified model for a run on the computer, punching 1 as the D Operand on the START card so that the Current and Future Events Chains will be printed out when the model shuts off.

(c) Compare the Current and Future Events Chains as printed out with their appearance in the last row in your hand-produced chain history. They should, of course, be in agreement with each other. If they are not, an error has been made in developing the hand-produced history. Correct the hand-produced history until it agrees with that printed out by the computer.

(d) Examine the hand-produced chain history to determine for how many of the 90 simulated minutes the barber is idle. Use this information to compute his utilization (fraction of the time he is busy). Your hand-computed utilization should be in agreement with that appearing in the Facility information produced in the computer output.

2.25 Case Study 2B: Extended Modeling of the One-Line, One-Server Queuing System

(1) Statement of the Problem

Two types of customers arrive at a one-chair barber shop. Customers of the first type want only a haircut. Their inter-arrival time distribution is 35±10 minutes, uniformly distributed. Customers of the second type want a shave as well as a haircut. Their inter-arrival time distribution is 60±20 minutes, uniformly distributed. The barber provides service to his customers first-come, first-served. This situation can be visualized in terms of Figure 2B.1, where "circles" represent customers who want only a haircut, and "squares" signify customers wanting both a shave and a haircut. At the time represented in Figure 2B.1, a "haircut only"

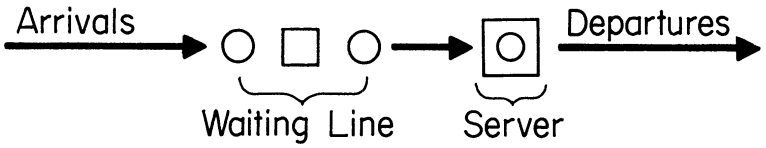


Figure 2B.1 First-Come, First-Served Queue Discipline with Two Types of Customers

customer is in the barber's chair, while "haircut only", "shave and haircut", and "haircut only" customers are waiting, in that order, for service.

It takes the barber 18±6 minutes to give a haircut. When he gives a shave, 10±2 minutes are required. These service times are uniformly distributed.

Model the barber shop in GPSS, making provisions to collect data on the waiting line that forms ahead of the barber. Then run the model through 8 hours of simulated time. Interpret the output produced by the model in the context of the barber shop.

(2) Approach Taken in Building the Model

The first impulse might be to attempt to model this one-line, one-server system with a single sequence of Blocks, analogous to the Figure 2A.1 model. But then such questions would arise as "how can one GENERATE Block be used to simulate both types of arrivals?", and "how can a distinction be made as to the different service times required by haircut-only, and shave-and-haircut, customers?" The conclusion can be rapidly drawn that the wherewithal required to model this system with a single sequence of Blocks has not yet been developed.

The system is easily modeled, however, with two distinct major Block segments, or sequences. One sequence models the haircut-only customers; the other models the shave-and-haircut customers. In each of the two sequences, a QUEUE-DEPART Block pair referencing one and the same waiting line is included so that customers moving through each sequence contribute to the waiting line statistics being gathered. And, in each of the two segments, a SEIZE-RELEASE Block pair referencing one and the same Facility is used to accomplish simulation of the barber himself. In the haircut-only segment, a single ADVANCE Block is used to simulate service time; in the shave-and-haircut segment, a pair of consecutive ADVANCE Blocks is used to simulate the time required to give a shave, and then a haircut, respectively. When approached in this fashion, then, it is relatively simple to model the system.

(3) Table of Definitions

Time Unit: 1 Minute

GPSS Entity	Interpretation
Transactions	
Model Segment 1	Haircut-only Customers
Model Segment 2	Shave-and-haircut Customers
Model Segment 3	A timer
Facilities	
JOE	The barber
Queues	
JOEQ	The Queue used to gather statistics on the combined waiting experience of both customer types

Table 2B.1 Table of Definitions for Case Study 2B

(5) Program Listing

```
BLOCK                                                                  CARD
NUMBER   *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS             NUMBER
                SIMULATE                                                 1
         *                                                               2
         *      MODEL SEGMENT 1                                          3
         *                                                               4
1                GENERATE   35,10       HAIRCUT-ONLY CUSTOMERS ARRIVE    5
2                QUEUE      JOEQ        ENTER THE LINE                   6
3                SEIZE      JOE         CAPTURE THE BARBER               7
4                DEPART     JOEQ        LEAVE THE LINE                   8
5                ADVANCE    18,6        USE BARBER                       9
6                RELEASE    JOE         FREE THE BARBER                 10
7                TERMINATE               LEAVE THE SHOP                  11
         *                                                              12
         *      MODEL SEGMENT 2                                         13
         *                                                              14
8                GENERATE   60,20       HAIRCUT-AND-SHAVE CUSTOMERS ARRIVE 15
9                QUEUE      JOEQ        ENTER THE LINE                  16
10               SEIZE      JOE         CAPTURE THE BARBER              17
11               DEPART     JOEQ        LEAVE THE LINE                  18
12               ADVANCE    10,2        USE BARBER FOR SHAVE            19
13               ADVANCE    18,6        USE BARBER FOR HAIRCUT          20
14               RELEASE    JOE         FREE THE BARBER                 21
15               TERMINATE               LEAVE THE SHOP                  22
         *                                                              23
         *      MODEL SEGMENT 3                                         24
         *                                                              25
16               GENERATE   480         TIMER ARRIVES AT TIME 480       26
17               TERMINATE  1           SHUT OFF THE RUN                27
         *                                                              28
         *      CONTROL CARDS                                           29
         *                                                              30
                START      1,,,1       START THE RUN;  GET CHAIN PRINTOUT AT END 31
                END        RETURN CONTROL TO OPERATING SYSTEM           32
```

Figure 2B.3 Extended Program Listing for Case Study 2B

(4) Block Diagram

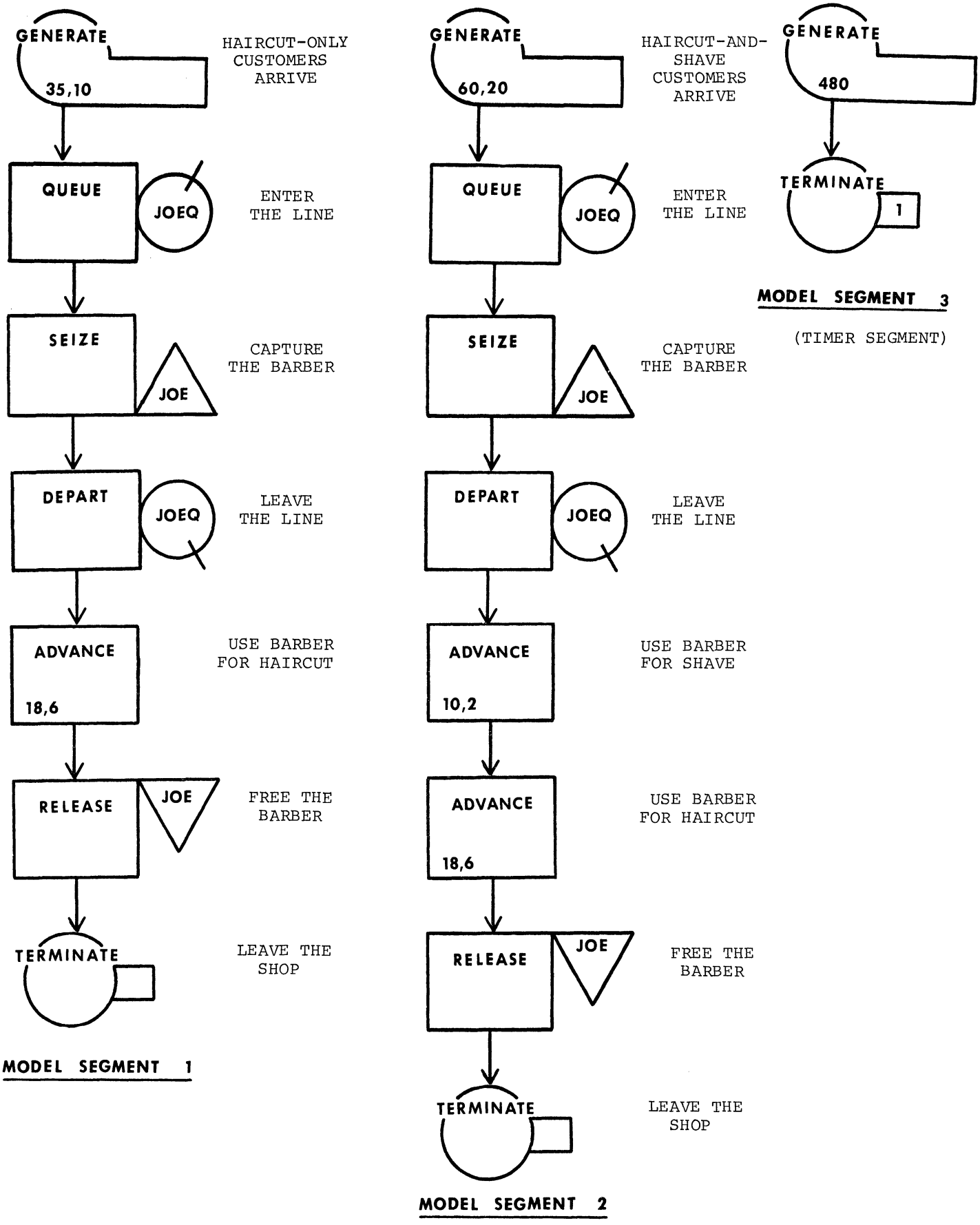


Figure 2B.2 Block Diagram for Case Study 2B

RELATIVE CLOCK 480 ABSOLUTE CLOCK 480

BLOCK COUNTS

BLOCK	CURRENT	TOTAL	BLOCK	CURRENT	TOTAL	BLOCK	CURRENT	TOTAL
1	0	13	11	0	8			
2	1	13	12	0	8			
3	0	12	13	1	8			
4	0	12	14	0	7			
5	0	12	15	0	7			
6	0	12	16	0	1			
7	0	12	17	0	1			
8	0	8						
9	0	8						
10	0	8						

(a) Clock Values and Block Counts

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
JOE	.897	20	21.549	1	

(b) Facility Statistics

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
JOEQ	2	.524	21	4	19.0	12.000	14.823		1

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(c) Queue Statistics

CURRENT EVENTS CHAIN

TRANS	BDT	BLOCK	PR	SF	NBA	SET	MARK-TIME	P1	P2	P3	P4	SI	TI	DI	CI	MC	PC	PF
6	457	2			3	6	457	0	0	0	0	1		1	2			
								0	0	0	0							
								0	0	0	0							

(d) Current Events Chain

FUTURE EVENTS CHAIN

TRANS	BDT	BLOCK	PR	SF	NBA	SET	MARK-TIME	P1	P2	P3	P4	SI	TI	DI	CI	MC	PC	PF
1	490	13			14	1	457	0	0	0	0				4			
								0	0	0	0							
								0	0	0	0							
4	502				1	4	-13	0	0	0	0				4			
								0	0	0	0							
								0	0	0	0							
5	504				8	5	-8	0	0	0	0				4			
								0	0	0	0							
								0	0	0	0							
2	960				16	2	-1	0	0	0	0				4			
								0	0	0	0							
								0	0	0	0							

(e) Future Events Chain

Figure 2B.4 Selected Program Output for Case Study 2B

(7) Discussion

Model Logic

The Block Diagram corresponding to the approach described above appears in Figure 2B.2. Segment 1 models the haircut-only customers. Segment 2 models the shave-and-haircut customers. Segment 3 provides the logic for the timer.

Note how Segments 1 and 2 each contain a QUEUE-DEPART Block pair referencing the same Queue (JOEQ). Also note how each of these two segments contains a SEIZE-RELEASE Block pair referencing the same Facility (JOE). The possibility of having more than one Block pair reference a given Queue or a given Facility was not previously mentioned, but there is no reason why this cannot be done. Indeed, the flexibility in allowing multiple points of reference to given GPSS entities is made evident in this example.

Program Output

Statistics describing the performance of the one Facility in the model are shown in Figure 2B.4(b). These statistics represent the aggregate effect of both types of customers on the barber. Under NUMBER ENTRIES, it is seen that the barber was captured by 20 customers in total. No distinction is made as to how many of these 20 customers were of the haircut-only type, or how many were of the shave-and-haircut variety. The output also shows that the barber was busy 89.7% of the time, but does not distinguish between that part of his time spent serving haircut-only customers, and that part spent serving shave-and-haircut customers.

Similarly, statistics in Figure 2B.4(c) describing the behavior of the one Queue in the model are aggregate in nature. No distinctions are made between the waiting experiences of the haircut-only customers, vs. the shave-and-haircut customers. The model simply is not designed to make these distinctions. If such distinctions are to be made, the model must be re-designed accordingly.

Assume it is important to gather Queue statistics segregated according to customer type, as well as aggregated over both types of customers. To get the segregated statistics, the Figure 2B.2 model must be embellished by introducing an additional Queue in Segment 1, and an additional Queue in Segment 2. When this is done, the model will simultaneously maintain three separate sets of waiting line records.

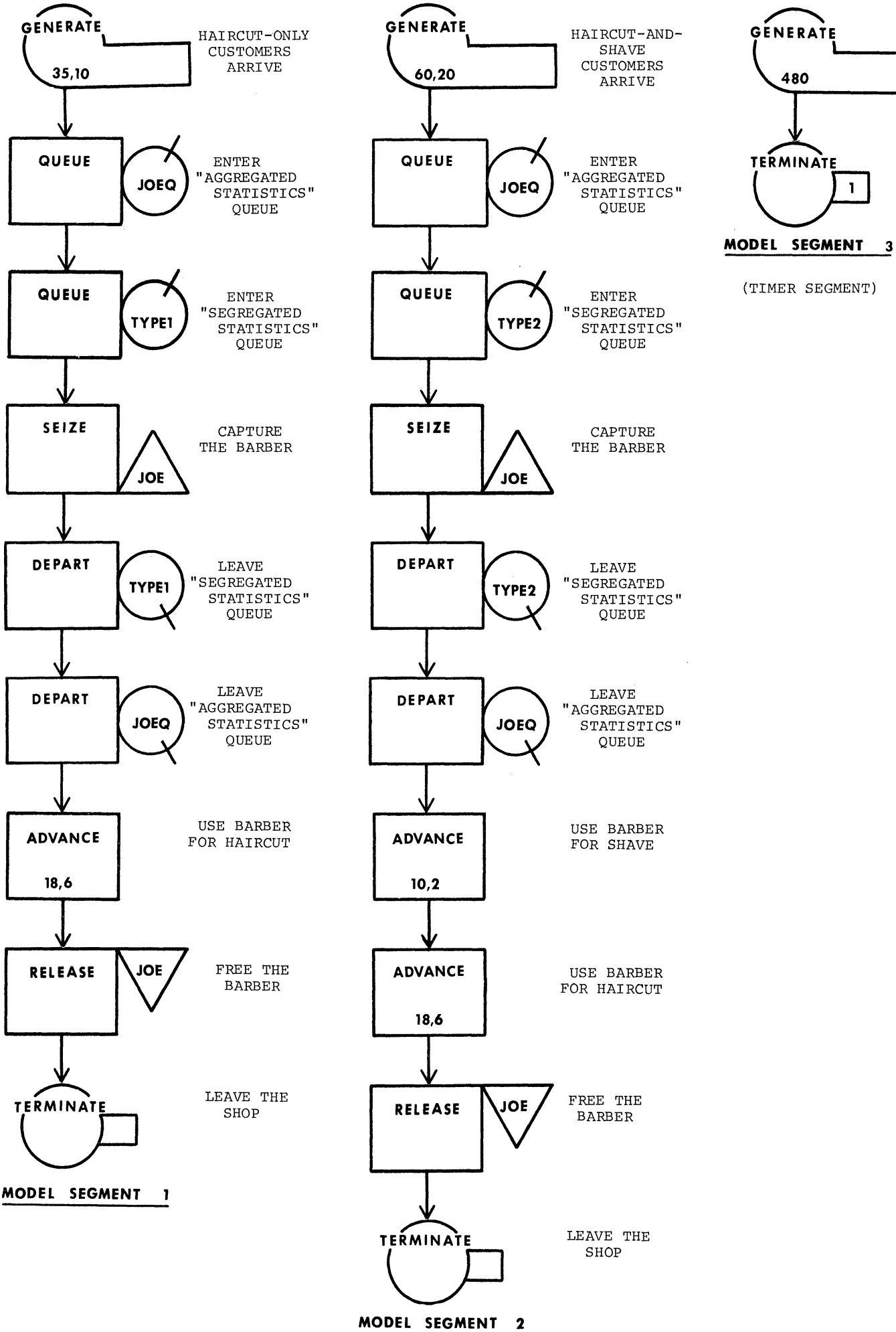


Figure 2B.5 Case Study 2B Modeled with Segregated and Aggregated Queues for the Two Types of Customers

Figure 2B.5 shows the Figure 2B.2 Block Diagram as modified to take these suggested additions into account. Additional Queues named TYPE1 and TYPE2 have been introduced in Segments 1 and 2, respectively. Note that, when a customer arrives in Segment 1, he joins the Queue JOEQ, then the Queue TYPE1, and then attempts to capture the Facility. When a capture is made, the customer then departs the Queues TYPE1 and JOEQ before service time is simulated at the ADVANCE Block. While waiting to capture, then, a haircut-only customer contributes simultaneously to the records being kept for two statistically different waiting lines. One set of records is unique to the haircut-only customer type; the other is a set maintained for all customers waiting for the Facility, independent of type. The same series of observations can be made for the shave-and-haircut customers simulated in model Segment 2.

It is important to realize that the inherent logic in the Figure 2B.5 model does not differ from that in Figure 2B.2. Logically, there is still only "one line" of customers waiting to capture the barber. The queue discipline exercised by the barber is still first-come, first-served. Whether the analyst chooses to maintain segregated waiting line statistics or not is quite beside the point. In short, as indicated in Section 2.15, residence in a GPSS Queue is simply a matter of records.

Note in Figure 2B.3 that 1 has been entered as the D Operand in the START Card. As a result, the Current and Future Events Chains were printed out by the Processor at the end of the simulation. These chains are shown in Figure 2B.4(d) and (e). Block Counts are shown in Figure 2B.4(a). The next set of exercises contains several questions based on this information.

2.26 A Second Example of the Use of Current and Future Events Chains

In glancing at Figure 2B.2, it might hastily be concluded that some sort of "redundancy" exists with respect to the Facility JOE or, for that matter, with respect to the Queue JOEQ. In terms of the Facility, for example, there is a tendency to ask, "how can one Facility be used at two different locations?" Questions like this apparently arise because the Facility is referenced from two pairs of SEIZE-RELEASE Blocks in the model. The number of pairs of SEIZE-RELEASE Blocks in the model, however, is not necessarily identical to the number of servers in the system being modeled. In Case Study 2B, there is only one server. There is no reason, however, why use of that server cannot be simulated with two or more pairs of SEIZE-RELEASE Blocks in the model of the system.

The fact that there is no redundancy with respect to the Facility or Queue in Figure 2B.2 is best emphasized through a numeric example showing how the GPSS Processor uses the Current and Future Events Chains in performing the simulation. Following the plan established in Section 2.21, then, such an example will now be presented.

Figure 2.32 is a repetition of the extended program listing for the Case Study 2B model.

```
BLOCK                                                                    CARD
NUMBER   *LOC   OPERATION   A,B,C,D,E,F,G          COMMENTS              NUMBER
                SIMULATE                                                    1
         *                                                                  2
         *      MODEL SEGMENT 1                                             3
         *                                                                  4
1                GENERATE    35,10       HAIRCUT-ONLY CUSTOMERS ARRIVE      5
2                QUEUE       JOEQ        ENTER THE LINE                     6
3                SEIZE       JOE         CAPTURE THE BARBER                 7
4                DEPART      JOEQ        LEAVE THE LINE                     8
5                ADVANCE     18,6        USE BARBER                         9
6                RELEASE     JOE         FREE THE BARBER                   10
7                TERMINATE                LEAVE THE SHOP                    11
         *                                                                  12
         *      MODEL SEGMENT 2                                             13
         *                                                                  14
8                GENERATE    60,20       HAIRCUT-AND-SHAVE CUSTOMERS ARRIVE 15
9                QUEUE       JOEQ        ENTER THE LINE                    16
10               SEIZE       JOE         CAPTURE THE BARBER                17
11               DEPART      JOEQ        LEAVE THE LINE                    18
12               ADVANCE     10,2        USE BARBER FOR SHAVE              19
13               ADVANCE     18,6        USE BARBER FOR HAIRCUT            20
14               RELEASE     JOE         FREE THE BARBER                   21
15               TERMINATE                LEAVE THE SHOP                    22
         *                                                                  23
         *      MODEL SEGMENT 3                                             24
         *                                                                  25
16               GENERATE    480         TIMER ARRIVES AT TIME 480         26
17               TERMINATE   1           SHUT OFF THE RUN                  27
         *                                                                  28
         *      CONTROL CARDS                                               29
         *                                                                  30
                 START       1,,,1       START THE RUN;  GET CHAIN PRINTOUT AT END  31
                 END         RETURN CONTROL TO OPERATING SYSTEM             32
```

Figure 2.32 Repetition of Figure 2B.2

Using the Block Numbers in Figure 2.32, Table 2.16 displays the sequences of inter-arrival times and service times which will be assumed in effect at the various GENERATE and ADVANCE Blocks in the model.

Block Number	Block Type	System Aspect Simulated	Assumed Sequence of Inter-Arrived Times or Service Times
1	GENERATE	Arrival of Haircut-Only Customers	44, 28, 35, 40, . . .
5	ADVANCE	Servicing of Haircut-Only Customers	20, 18, 22, 17, . . .
8	GENERATE	Arrival of Shave-and-Haircut Customers	41, 72, 52, 60, . . .
12	ADVANCE	Shaving of Shave-and-Haircut Customers	9, 11, 10, 12, . . .
13	ADVANCE	Hair-Cutting of Shave-and-Haircut Customers	14, 15, 20, 19, . . .
16	GENERATE	Timer	480

Table 2.16 Inter-Arrival Times and Service Times Used for the Second Chain Example

Remembering that the four-tuple notation used to record information about Transactions takes the form

[Transaction Number, Move Time, Priority Level, Next Block Attempted],

Figure 2.33 shows the history of Transaction residence on the Current and Future Events for the first few readings of the simulated clock.

Input Phase (From Line 1 to Line 2)

As shown at Line 1 in Figure 2.33, the chains are empty when the input phase begins. During the input phase, Block Number 1 is found to be a GENERATE Block. Transaction 1 is therefore fetched from the latent pool and hooked onto the Future Events Chain, scheduled to move to Block 1 at time 44, per Table 2.16. Next, a GENERATE Card is found at Block Location 8. Transaction 2 is consequently hooked onto the Future Events Chain, scheduled to move to Block 8 at time 41, per Table 2.16. Finally, a third and last GENERATE Card is found at Block Location Transaction 3 is therefore hooked onto the FEC, scheduled to move to Block 16 at time 480, per Table 2.9.

Line 2 in Figure 2.33 shows these three Transactions on the Future Events Chain at the completion of the model input phase. Note that Transaction 2 preceeds Transaction 1 on the chain, because of its smaller Move Time.

Before continuing to the first Clock Update Phase, note the significance of the Line 2 Future Events Chain Transactions, as indicated in Table 2.17.

Transaction Number	Significance
2	First haircut-only customer, on his way to the shop
1	First shave-and-haircut customer, on his way to the shop
3	The timer Transaction

Table 2.17 Significance of Transactions on Future Events Chain, Line 2, Figure 2.33

First Clock Update Phase (From Line 2 to Line 3)

The Processor first sets the clock at 41, the Move Time of the Transaction (Transaction 2) at the front of the Line 2 Future Events Chain. It next transfers Transaction 2 to the previously-empty Current Events Chain. The next Transaction (Transaction 1) on the Future Events chain does not have a Move Time of 41. The first execution of the Clock Update Phase is therefore finished.

Note at the Line 3 Current Events Chain that Transaction 2 is the first haircut-only customer, just arriving at the barber shop.

First Scan Phase (From Line 3 to Line 4)

Picking up Transaction 2 from the front-end of the Current Events Chain at Line 3, the Processor moves it to Block 8, then begins to move it into Block 9. Temporarily interrupting the processing of Transaction 2, the Processor now pre-schedules the arrival of a next Transaction at the Block 8 GENERATE Block. Corresponding to an inter-arrival time of 72 (second sample drawn from the 60 ± 20 distribution, per Table 2.9), Transaction 4 is fetched from the latent pool and hooked onto the FEC, scheduled to move to Block 8 at time 41 + 72, or 113.

The movement of Transaction 2 is then resumed. The Transaction moves through the QUEUE-SEIZE-DEPART-ADVANCE sequence (Blocks 9, 10, 11, and 12). At Block 12, a holding

Line Number	Simulated Time	Current Events Chain Front of Chain ←	Future Events Chain Front of Chain ←
1	Before Input Phase	Empty	Empty
2	After Input Phase	Empty	[2,41,0,8] [1,44,0,1] [3,480,0,16]
3	41	[2,ASAP,0,8]	[1,44,0,1] [3,480,0,16]
4	41	Empty	[1,44,0,1] [2,50,0,13] [4,113,0,8] [3,480,0,16]
5	44	[1,ASAP,0,1]	[2,50,0,13] [4,113,0,8] [3,480,0,16]
6	44	[1,ASAP,0,3]	[2,50,0,13] [5,72,0,1] [4,113,0,8] [3,480,0,16]
7	50	[1,ASAP,0,3] [2,ASAP,0,13]	[5,72,0,1] [4,113,0,8] [3,480,0,16]
8	50	[1,ASAP,0,3]	[2,64,0,14] [5,72,0,1] [4,113,0,8] [3,480,0,16]
9	64	[1,ASAP,0,3] [2,ASAP,0,14]	[5,72,0,1] [4,113,0,8] [3,480,0,16]
10	64	Empty	[5,72,0,1] [1,84,0,6] [4,113,0,8] [3,480,0,16]

Figure 2 33 Chain History for Second Example of the Use of Current and Future Events Chains

time of 9 minutes is sampled, per Table 2.9. Transaction 2 is then hooked onto the FEC, scheduled to move to Block 13 at time 50.

Because a SEIZE Block was executed, the Processor now re-starts its scan of the Current Events Chain. But the chain is empty. The next step, then, is to perform the Clock Update Phase.

Note that at Line 4 in Figure 2.33, Transactions on the Future Events Chain have the significance indicated in Table 2.18. In particular, consider the first shave-and-

Transaction Number	Significance
1	Haircut-only customer, on his way to the shop
2	First shave-and-haircut customer, being shaved
4	Second shave-and-haircut customer, on his way to the shop
3	The timer Transaction

Table 2.18 Significance of Transactions on Future Events Chain, Line 4, Figure 2.33

haircut customer, who is now being shaved. He moved through the QUEUE-DEPART Block pair at Blocks 9 and 11 at clock reading 41, thereby counting as a "zero entry" to the Queue JOEQ. He captured the Facility JOE by moving into the SEIZE Block which is Block 10 in the model.

Second Clock Update Phase (From Line 4 to Line 5)

Examining the Transaction at the front-end of the FEC in Line 4, the Processor advances the clock to a value of 44. The front-end FEC Transaction is then transferred to the previously-empty CEC. There are no other FEC-to-CEC transfers at time 44. The next Scan Phase can now begin.

Second Scan Phase (From Line 5 to Line 6)

Picking up Transaction 1 from the front-end of the CEC at Line 5, the Processor moves it to Block 1, then begins to move it into Block 2. Temporarily interrupting the processing of Transaction 1, the Processor next pre-schedules the arrival of a next Transaction at the Block 1 GENERATE Block. Corresponding to an inter-arrival time of 28 (Second sample drawn from the 35±10 distribution, per Table 2.9), Transaction 5 is fetchd from the latent pool and hooked onto the FEC, scheduled to move to Block 1 at time 44 + 28, or 72.

The Processor then resumes moving Transaction 1, completing execution of the QUEUE Block subroutine (Block 2). The Transaction is then denied entry to the SEIZE Block (Block 3). The reason, of course, is that the referenced Facility, JOE, is in a state of capture. It was captured at simulated time 41 by Transaction 2, which executed the SEIZE at that time at Block 10. The Processor consequently leaves Transaction 1 on the Current Events Chain, scheduled to SEIZE at Block 3 as soon as possible. While waiting to move into Block 3, Transaction 1 still contributes, of course, to the statistics being gathered for the Queue JOEQ. Hence, it is "present" in the Queue for which Transaction 1 was a "zero entry" at simulated time 41, via execution of the QUEUE-DEPART pair at locations 9 and 11.

Lines 7, 8, 9, and 10 in the Figure 2.33 chain history will not be commented upon in this discussion. It should already be quite clear that multiple references to GPSS entities in a model pose no problems for the GPSS Processor. The essential point is that there is only one Current Events Chain, and one Future Events Chain. By using

these two chains, the Processor can effectively and accurately record a variety of system events occurring in a model. Within this framework, multiple entity references are easily accommodated. The practical consequence of this, as the Figure 2B.2 model illustrates, is to provide the analyst with great flexibility in his approach to the model-building process.

2.27 Exercises

2.27.1 (Questions Based on Figure 2B.4, Case Study 2B)

(a) Note that the Current Count at Block 2 is 1, from Figure 2B.4(a). What is the number of the Transaction currently at that Block?

(b) Note that the Current Count at Block 13 is 1, from Figure 2B.4(a). What is the number of the Transaction currently at that Block?

(c) At Figure 2B.4(c), the current content of the Queue JOEQ is indicated to be 1. What Transaction is contributing to the content of the Queue? For how many time units has it been in the Queue?

(d) Which type of customer will next arrive at the shop, haircut-only, or shave-and-haircut?

(e) How many of each type of customer have been completely serviced when the simulation shuts off?

2.27.2 (Questions Based on Figure 2.33)

(a) At Line 7 in Figure 2.33, explain the significance of Transaction 2 on the Current Events Chain.

(b) At Line 8, explain the significance of Transaction 2 on the Future Events Chain.

(c) At what time does the first haircut-only customer get out of the barber's chair?

(d) At simulated time 64, Transaction 2 is returned to the latent pool of Transactions. At what simulated time will this Transaction be brought back into the model? Will it represent a haircut-only, or a shave-and-haircut customer at that time? (Note: it may be necessary to extend the Figure 2.33 chain history to answer these questions. If this is done, note that additional time values are available in Table 2.16.)

2.27.3 Using Block Numbers from Figure 2.32, Table T2.27.3 displays sequences of inter-arrival times and service times assumed to be in effect initially at the various GENERATE and ADVANCE Blocks there.

Block Number	Block Type	System Aspect Simulated	Assumed Sequence of Inter-Arrival Times or Service Times
1	GENERATE	Arrival of Haircut-Only Customers	31, 38, 29, 42, . . .
5	ADVANCE	Servicing of Haircut-Only Customers	15, 19, 14, . . .
8	GENERATE	Arrival of Shave-and-Customers	51, 32, 64, 49, . . .
12	ADVANCE	Shaving of Shave-and-Haircut Customers	9,11,10, . . .
13	ADVANCE	Hair-Cutting of Shave-and-Haircut Customers	14,17,12, . . .
16	GENERATE	Timer	480 (deterministic)

Using the inter-arrival times and service times in the table, construct a chain history similar to Figure 2.33, showing the Transactions resident on the Current and Future Events Chains through Line 10 in the history. Then answer these questions.

(a) Which Transaction represents the first haircut-only customer of the day?

(b) What is the clock reading at Line 10?

(c) Where is the first haircut-only customer at the time Line 10 is reached?

(d) Which Transaction represents the first shave-and-haircut customer of the day?

(e) Where is the first shave-and-haircut customer at the time Line 10 is reached?

(f) What type customer does Transaction 4 represent?

(g) What is the significance of Transaction 4 as it appears at Line 10?

2.27.4 Prepare the punchcards corresponding to the Figure 2B.5 model and submit the model for running on the computer. Compare and contrast the outputted Queue statistics for the Queues TYPE1, TYPE2, and JOEQ.

2.27.5 Suppose that the "QUEUE TYPE1" and "QUEUE JOEQ" Blocks in Segment 1, Figure 2B.5, were interchanged. Do you think it would then be necessary to interchange the "DEPART TYPE1" and "DEPART JOEQ" Blocks in Segment 1 for the model to be valid? Discuss this situation.

2.27.6 Figure 2B.5 shows how to gather segregated and aggregated Queue statistics for the two customer types in the system. Show how to gather segregated and aggregated Facility statistics in the system. That is, show how to construct the model so that it continues to gather aggregate Facility statistics, and in addition measures the fraction of the time the barber is engaged by haircut-only customers, the fraction of the time he is engaged by shave-and-haircut customers, the average time he spends with each customer type, and so on.

2.27.7 Three types of mechanics arrive at a tool crib to check out tools. Their inter-arrival times and service time requirements are shown in Table T2.27.7.

Type of Mechanic	Distribution of Inter-Arrival Times, Minutes	Distribution of Service Times, Minutes
1	30±10	12±5
2	20±8	6±3
3	15±5	3±1

Table T2.27.7

Only one clerk works at the tool crib. Build a GPSS model for the tool crib, then run a simulation with the model until 16 Type 1 mechanics have received service at the crib. Design the model so that it maintains separate Queue statistics for each type of mechanic. Compare and contrast the waiting-line experiences of the three mechanic types. Also compute the expected utilization of the server and compare your result with the model-measured utilization.

2.28 Case Study 2C: Alternative Queue Disciplines in the One-Line, One-Server Queuing System

(1) Statement of the Problem

In a certain factory, a tool crib is manned by a single clerk. The clerk checks out tools to mechanics, who use them to repair failed machines. (The tools involved are too expensive, and too numerous, for each mechanic to have each tool in his tool box.) The time to process a tool request depends on the type of tool. Requests fall into two categories. Pertinent data are shown in Table 2C.1.

Category of Tool Request	Mechanic Inter-Arrival Time, Seconds	Service Time, Seconds
1	420±360	300±90
2	360±240	100±30

Table 2C.1 Inter-Arrival Times and Service Times of Mechanics Using the Tool Crib

The clerk has been serving the mechanics first-come, first-served, independent of request. This queue discipline is shown in Figure 2C.1 where "circles" and "triangles" are mechanics making requests in Category 1 and 2, respectively.

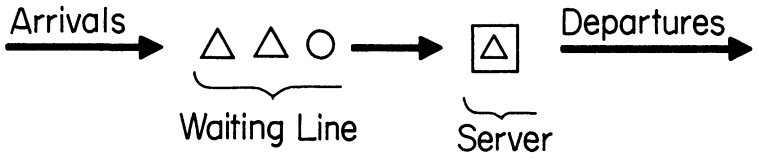


Figure 2C.1 First-come, First-served Queue Discipline, Two Customer Types with no Priority Distinctions

In Figure 2C.1 a Category 2 request is being served, while one request in Category 1, and two in Category 2, are waiting in that order for the server.

Because failed machines are out of production, it costs 0.25¢ per second ($9 per hour) when a mechanic waits for service at the tool crib. This cost is independent of the tool to be checked out. Management believes total mechanic waiting time can be reduced if Category 2 requests are serviced at the tool crib before those in Category 1.(2q) That is, only when no Category 2 requests are waiting is the clerk to service requests in Category 1. This queue discipline is pictured in Figure 2C.2, where, in effect, the line that forms ahea of the server consists of two segments. The segment at the front of the line is of "high priority"; that at the back of the line is of "low priority". The

(2q) The Category 2 service time is smaller than that for Category 1. When the server's rule for selecting his next customer is "do the job expected to take the least time", a "shortest imminent operation" queue discipline is said to be in effect.

The Figure 2C.2 queue discipline is said to be "first-come, first-served, within Priority Class".

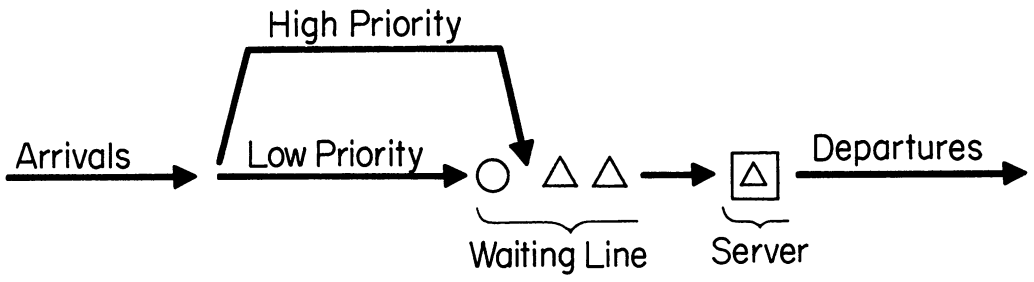


Figure 2C.2 First-Come, First-Served Queue Discipline, Two Customer Types with Priority Distinctions

The waiting-mechanic situation in Figure 2C.2 is identical to that in Figure 2C.1. The server is currently working on a Category 2 request ("triangle"). Two other Category 2 requests are waiting; and one Category 1 request is waiting ("circle"). Only when the high-priority segment of the line is empty will the low-priority segment be served.

Model the tool crib for each of the two queue disciplines indicated, simulating each case for an 8-hour work day. Does "first-come, first-served, within Priority Class" result in a smaller aggregate mechanic waiting time? In terms of lost production cost, what daily savings can be realized when priority distinctions are made? Do not include in the cost consideration the time mechanics spend waiting while they are being served.

(2) Approach Taken in Building the Model

The system to be modeled is very similar to that in Case Study 2B. That is, there are two different types of "customers" arriving at a single server. The differences are in the distributions of inter-arrival times, and service times, for the two arrival types. A further complication here, however, is that one arrival type is to have processing priority over the other type. The model can be built with an approach similar to that in Case Study 2B, then, if a method can be found to introduce this difference in processing priorities.

When entering a model, Transactions are given a Priority Level specified via their GENERATE Block's E Operand. When this was pointed out in Section 2.6, it was also stated that, "for reasons to be made clear later, it is convenient that various Transactions in a model may have distinctions made among one another with respect to their relative processing priority". In the problem at hand, just such a processing distinction is to be made.

Analogous to Case Study 2B, then, the Category 1 requests can be modeled in one segment, and Category 2 requests in another. The distinction in relative priorities is made by using a larger E Operand in the Category 2 GENERATE Block than in the Category 1 model segment. For example, Category 2 requests might enter the model through a GENERATE Block with an E Operand of 2; and Category 1 requests through a GENERATE Block with an E Operand of 1. Of course, the absolute Priority Levels chosen do not matter, as long as Transactions representing mechanics making Category 2 requests have a higher Priority Level than the other Transactions.

The way Priority Level influences a Transaction's processing priority is evident in terms of the Current Events Chain. Recall that, when a Transaction is transferred from the Future to the Current Events Chain, it takes a position on the Current Chain as the last member in its Priority Class. Furthermore, the higher the Priority Class, the closer to the front of the CEC a Transaction is. Because the GPSS Processor scans the Current Events Chain from front to back, it tries to move higher-priority Transactions forward in the model first. Later in the scan, it tries to move those with lower priorities. This means any mechanics waiting to make Category 2 requests (Priority Level 2) will be given the first opportunity to SEIZE, then mechanics waiting to make Category 1 requests (Priority Level 1) will have their chance to SEIZE. When the server has just become available and at least one mechanic in each category is waiting, a Category 2 mechanic will capture the server next simply because he is nearer the front of the CEC than Category 1 mechanics.

Model with No Priority Distinctions

Eliminating the priority distinctions from the model is a simple matter of having the GENERATE Block E Operands in the two major model segments be identical. The most convenient way to do this is to eliminate the E Operands from the GENERATE Blocks entirely; the result is that, by default, Category 1 and Category 2 requests have Priority Levels of zero.

The Statistic of Interest

The aggregate waiting time of mechanics is the statistic of primary interest in this problem. This statistic is available through the "AVERAGE CONTENTS" of the Queue in which the mechanics wait. Because the cost of having mechanics wait is independent of their request category, it is satisfactory to use only one GPSS Queue in modeling the system. The "AVERAGE CONTENTS" of this Queue can then be multiplied by 0.25¢ per second, or $9 per hour, or $72 per day, to get the cost per second, hour, or day, respectively, associated with waiting mechanics.

(3) Table of Definitions

Time Unit: 1 Second

GPSS Entity	Interpretation
Transactions	
Model Segment 1	Mechanics making Category 1 tool requests
Model Segment 2	Mechanics making Category 2 tool requests
Model Segment 3	A timer
Facilities	
CLERK	The tool crib clerk
Queues	
LINE	The Queue used to gather statistics on the combined waiting experience of mechanics making tool requests in both categories

Table 2C.1 Table of Definitions for Case Study 2C

(4) Block Diagram

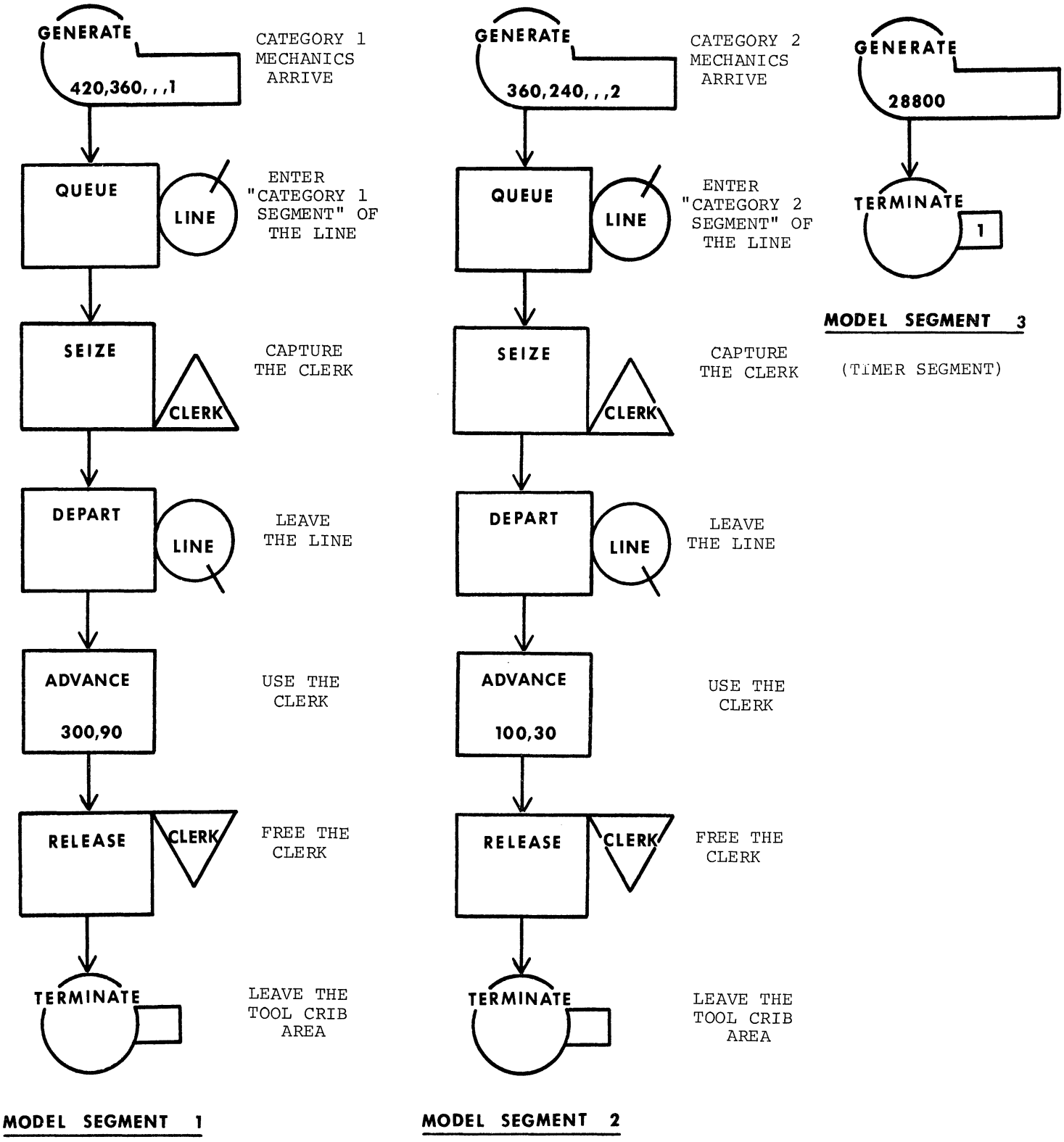


Figure 2C.3 Block Diagram for Case Study 2C (Priority Distinctions Made)

(5) Extended Program Listing

```
BLOCK                                                                    CARD
NUMBER   *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS               NUMBER
                SIMULATE                                                    1
         *                                                                  2
         *      MODEL SEGMENT 1                                             3
         *                                                                  4
1                GENERATE   420,360,,,1       CATEGORY 1 MECHANICS ARRIVE    5
2                QUEUE      LINE              ENTER "CATEGORY 1 SEGMENT" OF LINE  6
3                SEIZE      CLERK             CAPTURE THE CLERK              7
4                DEPART     LINE              LEAVE THE LINE                 8
5                ADVANCE    300,90            USE THE CLERK                  9
6                RELEASE    CLERK             FREE THE CLERK                10
7                TERMINATE                     LEAVE THE TOOL CRIB AREA      11
         *                                                                  12
         *      MODEL SEGMENT 2                                            13
         *                                                                  14
8                GENERATE   360,240,,,2       CATEGORY 2 MECHANICS ARRIVE   15
9                QUEUE      LINE              ENTER "CATEGORY 2 SEGMENT" OF LINE  16
10               SEIZE      CLERK             CAPTURE THE CLERK             17
11               DEPART     LINE              LEAVE THE LINE                18
12               ADVANCE    100,30            USE THE CLERK                 19
13               RELEASE    CLERK             FREE THE CLERK                20
14               TERMINATE                     LEAVE THE TOOL CRIB AREA      21
         *                                                                  22
         *      MODEL SEGMENT 3                                            23
         *                                                                  24
15               GENERATE   28800             TIMER ARRIVES AFTER 8 HOURS   25
16               TERMINATE  1                 SHUT OFF THE RUN              26
         *                                                                  27
         *      CONTROL CARDS                                              28
         *                                                                  29
                START      1                 START THE RUN                 30
                END        RETURN CONTROL TO OPERATING SYSTEM               31
```

Figure 2C.4 Program Listing for Case Study 2C (Priority Distinctions Made)

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
CLERK	.932	140	191.878	6	

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
LINE	3	.770	140	20	14.2	158.500	184.916		

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(a) Output When Priority Distinctions Are Made

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
CLERK	.959	142	194.605	1	

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
LINE	7	2.731	145	13	8.9	542.593	596.030		3

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(b) Output When No Priority Distinctions Are Made

Figure 2C.5 Selected Program Output for Case Study 2C

(7) Discussion

Model Implementation

The model shown in Figure 2C.4 was first submitted for running. Then the E Operands were eliminated from the GENERATE cards, and the resulting no-priority-distinction model was submitted for processing in a separate run. This approach is relatively awkward, requiring two separate runs. A method for accomplishing the processing in a single run will be described later in this chapter.

Program Output

As indicated, the Queue statistic of primary interest is the "AVERAGE CONTENTS". Figure 2C.5(b) shows that the average number of waiting mechanics when no priority distinctions are made is 2.731 for the 8-hour day simulated. When priority distinctions are made, this average decreases to 0.770, as shown in Figure 2C.5(a). Making a priority distinction, then, has led to "improved system performance" in the sense that the average number of waiting mechanics has been decreased by about 70%. Note that this difference is based on about 140 to 145 entries to the Queue ["TOTAL ENTRIES", Figure 2C.5(a) or (b)]. Due to randomness, there were 5 more arrivals at the tool crib during the no-priority-distinction simulation than during the priority-distinction run.

When no priority distinctions are made, the cost of waiting mechanics for the day simulated amounts to about $197 (2.731 times $72/day = $196.63). With priority distinctions in effect, the experienced cost was about $55 ($55.44) The difference in the experienced costs is then about $141 ($141.19).

Despite this difference in experienced costs, the duration of simulation used in this case study is so small that any conclusions drawn from the model output are tentative at best. After certain other features of GPSS have been introduced, this tool crib problem will be further investigated. In Case Study 3C, additional experimentation with the tool crib suggests that the expected daily savings resulting from a priority-distinction queue discipline is about $48, meaning that the savings indicated above is quite untypical.

2.29 A Third Example of the Use of Current and Future Events Chains

In Case Study 2C, a distinction is made among the Priority Levels of Transactions in two different segments of the model. This results in modeling a queue discipline described as "first-come, first-served, within Priority Class". Any doubts as to why this queue discipline is in effect as a consequence of controlling Transaction Priority Levels can be resolved by considering a numeric example showing how the GPSS Processor uses the Current and Future Events Chains to move the Figure 2C.4 model forward in simulated time. The initial details of such an example will now be presented.

Figure 2.34 is a repetition of the extended program listing for the Case Study 2C model.

```
BLOCK                                                                  CARD
NUMBER  *LOC  OPERATION  A,B,C,D,E,F,G        COMMENTS                 NUMBER
              SIMULATE                                                    1
        *                                                                 2
        *     MODEL SEGMENT 1                                             3
        *                                                                 4
1             GENERATE   420,360,,,1        CATEGORY 1 MECHANICS ARRIVE   5
2             QUEUE      LINE               ENTER "CATEGORY 1 SEGMENT" OF LINE  6
3             SEIZE      CLERK              CAPTURE THE CLERK             7
4             DEPART     LINE               LEAVE THE LINE                8
5             ADVANCE    300,90             USE THE CLERK                 9
6             RELEASE    CLERK              FREE THE CLERK               10
7             TERMINATE                      LEAVE THE TOOL CRIB AREA     11
        *                                                                12
        *     MODEL SEGMENT 2                                            13
        *                                                                14
8             GENERATE   360,240,,,2        CATEGORY 2 MECHANICS ARRIVE  15
9             QUEUE      LINE               ENTER "CATEGORY 2 SEGMENT" OF LINE  16
10            SEIZE      CLERK              CAPTURE THE CLERK            17
11            DEPART     LINE               LEAVE THE LINE               18
12            ADVANCE    100,30             USE THE CLERK                19
13            RELEASE    CLERK              FREE THE CLERK               20
14            TERMINATE                      LEAVE THE TOOL CRIB AREA     21
        *                                                                22
        *     MODEL SEGMENT 3                                            23
        *                                                                24
15            GENERATE   28800              TIMER ARRIVES AFTER 8 HOURS  25
16            TERMINATE  1                  SHUT OFF THE RUN             26
        *                                                                27
        *     CONTROL CARDS                                              28
        *                                                                29
              START      1                  START THE RUN                30
              END        RETURN CONTROL TO OPERATING SYSTEM              31
```

Figure 2.34 Repetition of Figure 2C.4

Using the Block Numbers in Figure 2.34, Table 2.19 displays the sequences of inter-arrival times and service times which will be assumed in effect at the various GENERATE and ADVANCE Blocks in the model. Finally, Figure 2.35 shows the history of Transaction residence on the Current and Future Events Chains.

Block Number	Block Type	System Aspect Simulated	Assumed Sequence Inter-Arrival Times or Service Times
1	GENERATE	Arrival of mechanics making Category 1 requests	392,223,426, . . .
5	ADVANCE	Servicing of Category 1 requests	309, . . .
8	GENERATE	Arrival of mechanics making Category 2 requests	403,279,352, . . .
12	ADVANCE	Servicing of Category 2 requests	Not Used in Example
15	GENERATE	Timer	28800 (Deterministic)

Table 2.19 Inter-Arrival Times and Service Times Used for the Third Chain Example

Line Number	Simulated Time	Current Events Chain Front of Chain ←	Future Events Chain Front of Chain ←
1	Before Input Phase	Empty	Empty
2	After Input Phase	Empty	[1,392,1,1] [2,403,2,8] [3,28800,0,15]
3	392	[1,ASAP,1,1]	[2,403,2,8] [3,28800,0,15]
4	392	Empty	[2,403,2,8] [4,615,1,1] [1,701,1,6] [3,28800,0,15]
5	403	[2,ASAP,2,8]	[4,615,1,1] [1,701,1,6] [3,28800,0,15]
6	403	[2,ASAP,2,10]	[4,615,1,1] [5,682,2,8] [1,701,1,6] [3,28800,0,15]
7	615	[2,ASAP,2,10] [4,ASAP,1,1]	[5,682,2,28] [1,701,1,6] [3,28800,0,15]
8	615	[2,ASAP,2,10] [4,ASAP,1,3]	[5,682,2,8] [1,701,1,6] [6,1041,1,1] [3,28800,0,15]
9	682	[2,ASAP,2,10] [5,ASAP,2,8] [4,ASAP,1,3]	[1,701,1,6] [6,1041,1,1] [3,28800,0,15]
10	682	[2,ASAP,2,10] [5,ASAP,2,10] [4,ASAP,1,3]	[1,701,1,6] [7,1034,2,8] [6,1041,1,1] [3,28800,0,15]

Figure 2.35 Chain History for the Third Example of the Use of Current and Future Events Chains

The details of Figure 2.35 will not be completely traced. Those interested can construct the chains as shown following the procedure already illustrated in the first and second chain examples. Of primary interest here are Lines 6, 8, and 10, showing the chain appearances at the completion of the Scan Phase after simulated times 403, 615, and 682, respectively. The significance of Transactions on the chains at those times will now be indicated.

The several Transactions on the chains at Line 6 have the interpretations shown in Table 2.20. In particular, Transaction 2 is waiting for the clerk, while

Chain	Transaction Number	Significance
CEC	2	Mechanic waiting to make the first Category 2 request
FEC	4	Second Category 1 mechanic, on his way to the tool crib
FEC	5	Second Category 2 mechanic, on his way to the tool crib
FEC	1	First Category 1 mechanic, having his request serviced by the tool crib clerk
FEC	3	The timer Transaction

Table 2.20 Significance of Transactions on Chains at Line 6, Figure 2.35

Transactions 4 and 5 represent mechanics on their way to the tool crib.

Then, at simulated time 615, the second Category 1 mechanic (Transaction 4) arrives at the tool crib. Note that, in being transferred from the Future to the Current Events Chain, Transaction 4 has been put behind Transaction 2 on the Line 7 CEC. This is consistent with the ordering property imposed on the Current Events Chain (incoming Transactions take up the last position in their Priority Class). Transaction 4 has a Priority Level of 1, whereas that of Transaction 2 is 2. In fact, even if the Priority Level of Transaction 4 were 2, it would be behind Transaction 2. At this point, then, the difference in Priority Levels has had no effect on the operation of the model. On the Current Events Chain after the Scan Phase (Line 8), Transactions 2 and 4 are waiting for the services of the tool crib clerk in "natural" order, based on their time of arrival at the crib.

Then, in the next Clock Update Phase, the clock is advanced to 682, the time of arrival of the second Category 2 mechanic (Transaction 5). Unhooking Transaction 5 from the front of the Future Events Chain at Line 8, the Processor merges it between Transactions 2 and 4 on the Current Events Chain (Line 9). In that position, it appears ahead of the lower-priority Transaction which was already resident on that chain, but behind the previous Priority Level 2 chain resident. The direct consequence of this ordering is that, eventually, Transaction 5 will capture the Facility ahead of Transaction 4, even though Transaction 4 has been waiting longer. Furthermore, if any additional Priority Level 2 Transactions are brought to the CEC before Transaction 4 has moved into Block 3 (SEIZE), they, too, will be merged ahead of it on the Current Events Chain, further delaying the time when the Facility will be made available to it.

Hence, the three elements of (1) Transaction Priority Level, (2) Ordering property on the Current Events Chain, and (3) Scan and re-scan of the CEC from front to back, combine to make first-come, first-served, within Priority Class the default queue discipline in GPSS.

2.30. Exercises

2.30.1 In the third example of the use of Current and Future Events Chains, assume that, after the Scan Phase at time 28752, these conditions are in effect:

(a) The tool crib clerk is idle.

(b) Transactions 2 is on the Future Events Chain, representing a mechanic scheduled to arrive at the tool crib at time 28800 (shutoff time).

(c) Except for Transactions 2 and 3 (the timer), no other Transactions in the model have a Move Time of 28800 or less.

Answer these questions.

(1) What is the composition of the Current Events Chain?

(2) What is the number of the Transaction at the front of the Future Events Chain?

(3) When the model shuts off, will the tool crib clerk be idle or in a a state of capture?

2.30.2 Build a model for Problem 2.27.7 (three types of arrivals at a tool crib). The server at the tool crib should use "shortest imminent operation" as a queue discipline. That is, he should serve next that request which is expected to require the smallest service time. Design the model so that it maintains separate Queue statistics for each of the three types of arrivals.

Simulate with the model until 16 Type 1 arrivals have received service at the crib. Compare and contrast the waiting experiences of the three arrival types.

Also compare and contrast the various waiting experiences with those determined for Problem 2.27.7, where queue discipline was first-come, first-served.

At what simulated time does the Problem 2.30.2 model shut off? When did the Problem 2.27.7 model shut off? How would you explain this difference?

2.30.3 In Case Study 2A, the barber works for 480 consecutive minutes, without taking a break. Show how to modify the model in that case study to include these features.

(a) Assume that the barber shop opens at 8:30 a.m. and closes at 5:00 p.m. The barber takes a 30-minute lunch break at 12:00 noon, or as soon thereafter as possible (i.e., if he is at an intermediate point in working on a customer when it becomes noon, he finishes with that customer before taking the 30-minute lunch break). Customers who arrive at the shop during the barber's break wait for him to return. What will be the barber's long-run utilization?

(b) Assume the shop opens at 8:00 a.m. and closes at 5:00 p.m. The barber takes these breaks at the indicated times, or as soon thereafter as possible.

(1) Coffee break, 10:00-10:15 a.m.

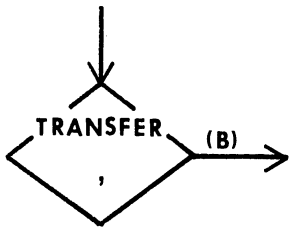
(2) Lunch break, 12:00-12:30 p.m.

(3) Coke break, 3:00-3:15 p.m.

Customers who arrive during any of the breaks choose to wait for the barber. After showing how to modify the case study 2A model to incorporate these changes, compute the barber's long-run utilization.

2.31. Routing Transactions to Non-Sequential Blocks

It is occasionally of interest to divert Transactions unconditionally to some non-sequential Block in a GPSS model. This can be accomplished by using the TRANSFER Block in unconditional transfer mode. As used in this mode, the TRANSFER Block is shown in Figure 2.36.



Operand	Significance	Default Result
A	Not used	---
B	Block Location to which Transactions move next	Error

Figure 2.36 The TRANSFER Block in Unconditional Transfer Mode

The A Operand is not employed in this TRANSFER Block usage. This means that a comma must be entered in card column 19 when the Block's punchcard image is prepared. This is suggested by the comma placed within the Block.

The B Operand names the Location occupied by the next Block the Transaction will attempt to enter. For brevity, whatever Block is in that Location will be termed the "B Block". The Block Location can be provided either symbolically or numerically. A symbolic Location Name is usually the most convenient to use. Otherwise, it is necessary for the analyst to determine the absolute Location Number that will be assigned to the "B Block" by the Processor.

In Figure 2.36, parentheses have been used around the B Operand. These parentheses appear only in the Block Diagram of the model. They are not part of the B Operand, and do not appear in the punchcard image of the TRANSFER Block.

In unconditional transfer mode, the TRANSFER Block cannot refuse entry to a Transaction. When a Transaction enters the Block, it immediately tries to move into the B Block. If that Block denies entry, the Transaction is left on the Current Events Chain. At each next scan of the Current Events Chain (whether due to a scan re-start within the current Scan Phase, or to initiation of the next Scan Phase), an attempt is again made to move the Transaction into the B Block. Eventually, the attempt will presumably be

successful.

It has been emphasized that the TRANSFER Block usage described above is termed "TRANSFER Block in unconditional transfer mode". There are alternative modes in which the TRANSFER Block can be used. Discussion of some of these alternatives will be taken up later.

Use of the TRANSFER Block in unconditional transfer mode is illustrated in Case Study 2D in the next section.

2.32. Case Study 2D: A One-Line, One-Server Queuing System with Feedback

(1) Statement of the Problem

The manufacture of a certain line of widgets involves a relatively lengthy assembly process, followed by a short firing time in an oven. Since ovens are expensive to maintain, several assemblers share a single oven, which holds only one widget at a time. An assembler cannot begin assembling a new widget until he has removed the old one from the oven.

This is the pattern followed, then, by each assembler:

(1) Assemble next widget.

(2) Wait, first-come, first-served, to use the oven.

(3) Use the oven.

(4) Return to (1).

Time and cost studies have been conducted to provide the information in Tables 2D.1 and 2D.2, respectively.

Operation	Time Required, Minutes
Assemble	30±5
Fire	8±2

Table 2D.1 Operation Times for Case Study 2D

Item	Cost Information
Assembler's Salary	$3.75 per hour
Oven Cost	$80 per 8-hour work day (independent of utilization)
Raw Material	$2 per widget
Value of Finished Widgets	$7 per widget

Table 2D.2 Financial Data for Case Study 2D

Build a GPSS model of this manufacturing process. Use the model to determine the optimal number of assemblers to be assigned to an oven. The optimal number is understood in this context to be the one maximizing profit. Base the determination on simulations equivalent to 40-hours of simulated time. Assume there are no discontinuities within a working day, or in moving between consecutive 8-hour work days.

(2) Approach Taken in Building the Model

A good approach to use in building many GPSS models is this:

(1) First, identify the constraints in the system to be modeled.

(2) Then, make a decision about which GPSS entities to use to simulate these constraints in the model itself.

In the current problem, there are two constraints. First, there is only one oven involved. Second, there is some fixed number of assemblers who are in the system. It is natural to choose a Facility to simulate the oven. It is equally natural to let a Transaction represent an assembler. Then, even as the assemblers can be thought of as "circulating" through the system as they move repeatedly through the assemble-fire cycle, the Transactions can circulate through the GPSS model of the system.

In the system itself, after each assembler takes a fired widget from the oven, he "goes back" to begin his next assembly. In the model of the system, after a Transaction has released the Facility simulating the oven, it can be "routed back" via a TRANSFER Block to the ADVANCE Block where the next assembly process is simulated.

As to constraining the total number of Transactions which circulate through the model, the D Operand at the GENERATE Block provides a convenient method for fixing this total number.

Ultimately, to compute the profit associated with a given number of assemblers, it will be necessary to know how many finished widgets they produced during the course of a simulation. The number of releases of the oven Facility conveniently provides this count.

(3) Table of Definitions

Time Unit: 1 Minute

GPSS Entity	Interpretation
Transactions	
Model Segment 1	Assemblers
Model Segment 2	The timer
Facilities	
OVEN	The oven

Table 2D.3 Table of Definitions for Case Study 2D

(4) Block Diagram

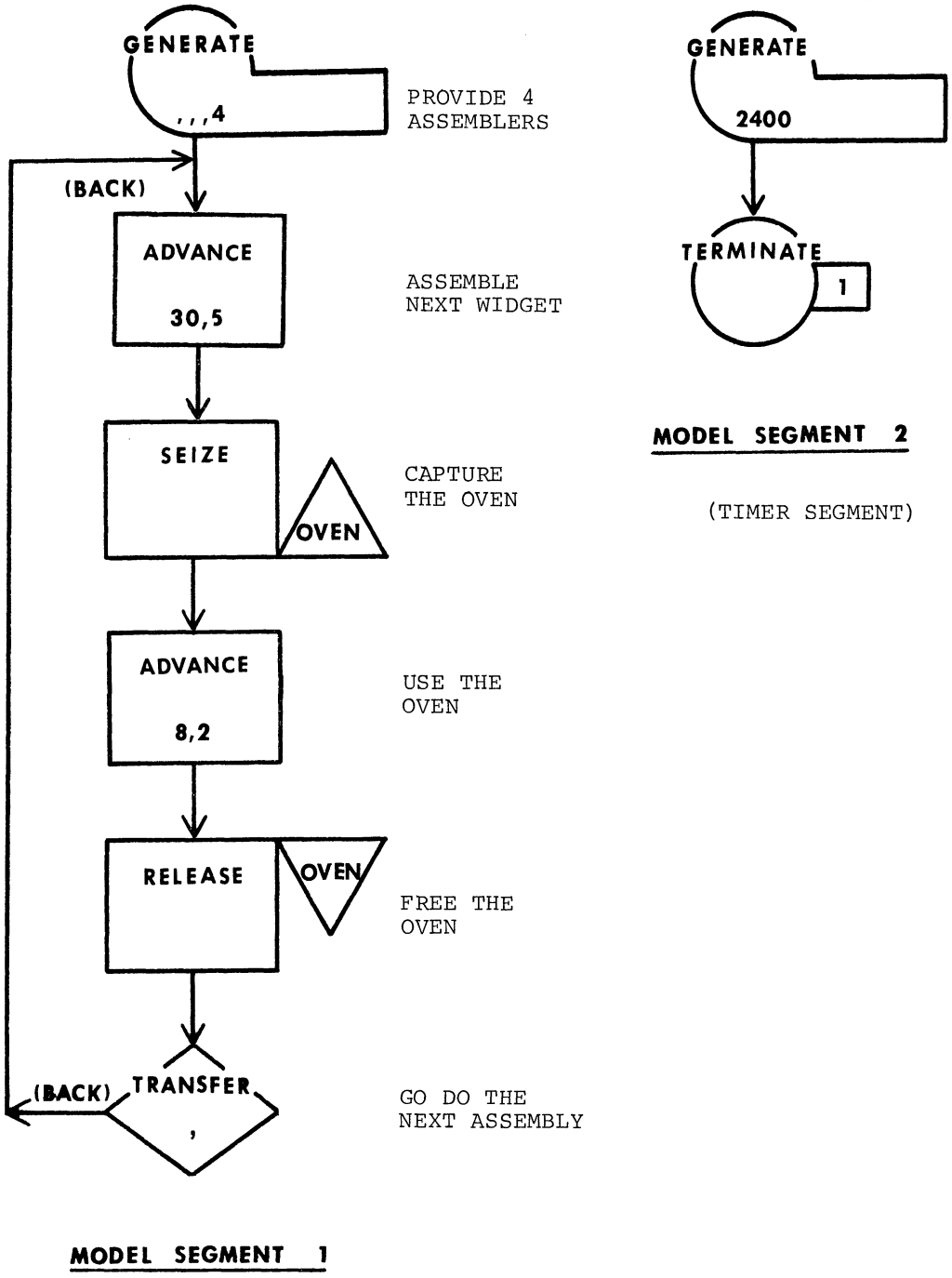


Figure 2D.1 Block Diagram for Case Study 2D (4 Assemblers Used)

(5) Extended Program Listing

```
BLOCK                                                                CARD
NUMBER   *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS          NUMBER
                SIMULATE                                                1
         *                                                              2
         *      MODEL SEGMENT 1                                         3
         *                                                              4
1                GENERATE   ,,,4          PROVIDE 4 ASSEMBLERS       5
2         BACK   ADVANCE    30,5          ASSEMBLE NEXT WIDGET       6
3                SEIZE      OVEN          CAPTURE THE OVEN           7
4                ADVANCE    8,2           USE THE OVEN               8
5                RELEASE    OVEN          FREE THE OVEN              9
6                TRANSFER   ,BACK         GO DO THE NEXT ASSEMBLY   10
         *                                                             11
         *      MODEL SEGMENT 2                                        12
         *                                                             13
7                GENERATE   2400          TIMER ARRIVES AFTER 5 DAYS 14
8                TERMINATE  1             SHUT OFF THE RUN           15
         *                                                             16
         *      CONTROL CARDS                                          17
         *                                                             18
                START      1             START THE RUN              19
                END        RETURN CONTROL TO OPERATING SYSTEM         20
```

Figure 2D.2 Extended Source Listing for Case Study 2D (4 Assemblers Used)

(6) Program Output

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
OVEN	.785	236	7.991	5	

(a) Oven Statistics When 4 Assemblers Are Used

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
OVEN	.952	280	8.160	5	

(b) Oven Statistics When 5 Assemblers Are Used

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
OVEN	.976	574	8.162	10	

(c) Oven Statistics When 6 Assemblers Are Used

Figure 2D.3 Selected Program Output for Case Study 2D

(7) Discussion

Model Logic

This is the first model in which there has been occasion to give a symbolic name to a Block Location. In Figure 2D.1, the Location of the "ADVANCE 30,5" Block has been given the name BACK. In Figure 2D.2, this symbolic name appears in the Location Field (under the label *LOC) on the card image of the ADVANCE Block. In the Block Number column in that figure, it is seen that the symbolic location BACK is equivalent to numeric location 2.

In Figure 2D.1, although assemblers must wait for the oven, the SEIZE Block is not sandwiched between a QUEUE-DEPART Block pair. No Queue statistics are needed for the problem as stated. As a result, no provisions are made to gather waiting line data in the model.

Assemblers waiting to use the oven are represented by Transactions on the Current Events Chain, waiting to move into Block 3 (SEIZE) "as soon as possible". From the point of view of "Block Counts", these Transactions contribute to the "Current Count" at Block 2 (ADVANCE). This is the case even though their computed "holding time" at the ADVANCE Block has already elapsed.

Model Implementation

The three assembler:oven ratios investigated were studied by making three separate computer runs. Only the D Operand in the Segment 1 GENERATE Block was changed from run to run, to provide for the changing assembler: oven ratio. In Section 2.33, a GPSS Control Card will be introduced which makes it possible to study a variety of assembler: oven ratios in sequence, with a single computer run.

Program Output

The cost of the oven for the 40-hour week simulated is $400. Each assembler earns $150 per week. The number of widgets finished during the 40-hour week were 235, 279, and 294 for the cases of 4, 5, and 6 assemblers, respectively. (This "finished widget" count excludes the widget still in the oven, if any, at the end of the simulation.) The corresponding return-after-costs for the three cases is easily computed to be $175, $245, and $170, respectively. Hence, the optimal number of assemblers per oven is 5.

This problem involves a simple tradeoff between the cost of waiting assemblers, and the cost of an idle oven. With 4 assemblers, assembler waiting time (not determined by the model) is relatively low, but oven utilization is also low [0.784, Figure 2D.3(a)]. With 5 assemblers, assembler waiting time increases, but so does oven utilization [0.942, Figure 2D.3(b)]. Hence, the salary of the additional assembler is more than offset by the increased oven utilization. With 6 assemblers, however, assembler waiting time increases even more, but the oven utilization is not affected that greatly [0.988, Figure 2D.3(c)]. The modest increase in oven utilization with 6 assemblers is not enough to offset the cost of the additional assembler.

2.33. GPSS Control Cards: The CLEAR Card

In Case Study 2D, having built a model for the system to be simulated, it was necessary to make a series of sequential runs with slightly modified versions of the model. The model modification involved changing the D Operand in the Segment 1 GENERATE Block. This, in turn, constituted a re-configuration of the system, in the sense that it changed the number of assemblers using a single oven. The need and desire to re-configure models, then make more computer runs, often arises in simulation modeling. In fact, one of the principle uses of simulation is to measure the behavior of alternative systems that might be used to accomplish a given objective.

It would clearly be convenient if the analyst, instead of submitting a series of batch jobs to perform the change-and-run steps, could have these steps performed in a single batch session. This would have the following implications.

(1) After a given simulation shuts off and the system statistics have been outputted, the existing model would have to be susceptible to modification. For example, it would have to be possible to re-define the Operands in effect at user-selected Blocks in the model.

(2) After modification of the model, it would have to be made ready to run again. This would entail three things:

   (a) All model statistics would have to be set back to zero.

   (b) All Transactions that were in the model at the end of the previous simulation would have to be removed. This means that the chains would have to be emptied, with Transactions on them at the time of the last shutoff being returned to the latent pool.

   (c) What was previously described as the "model input phase" would have to be performed. That is, the Processor would have to examine the re-configured model, looking for GENERATE Blocks. For each GENERATE Block found, it would fetch a Transaction from the latent pool and hook it onto the Future Events Chain accordingly.

(3) Finally, after re-configuring the model and making it ready to run again, the simulation would have to be re-initiated. The Processor would have to initialize the Termination Counter, then begin execution of the Clock Update Phase, setting the simulation clock to the Move Time of the Transaction at the front of the Future Events Chain, etc. etc.

Each of the above steps can be accomplished in GPSS. The method for carrying this out will now be described.

Suppose that, having inputted a model and come to a START Card, the Processor has initiated a simulation. Eventually, the Termination Counter is decremented to zero, the simulation shuts off, and various output is produced. The model with which the simulation has just been performed is left as it was when the simulation shut off. The Processor now looks at the next card in the deck to find its next instruction.[2r] Assume the next card is the image of one or more Blocks already in the existing model. For example, suppose that a GENERATE Block is in the model at the Location named KEY, and that immediately after the first START Card, the analyst has placed the card "KEY GENERATE 15,5,,30". Finding this new definition of the Block in the Location KEY, the Processor replaces the Block previously at that Location with this newly-defined

---

[2r] In the models presented so far, the next card has always been the END Card. This card signals the Processor to return control to the operating system.

Location occupant. This means the Block in the Location KEY, although still a GENERATE Block, now has A, B, C, and D Operands of 15, 5, Default Value (No Offset), and 30, respectively. It is this ability to re-define the Operands of Blocks occupying already-existing Locations in a model which makes possible the model modification described as Step (1) above.[2s]

After the START Card, then, the analyst may include one or more cards which re-define the Blocks occupying selected Locations in the model.[2t] Acting upon these cards one-by-one, the Processor always goes to the next card in the deck to find out what to do next. Suppose that, after the last Block re-definition card, the analyst has placed a CLEAR Card in the deck. The CLEAR Card is a GPSS Control Card which performs step (2) above. It consists of the word CLEAR, entered in the Operation Field on a punchcard. There is no Block equivalent for it. Whenever the Processor encounters a CLEAR Card, it sets model statistics back to zero (including both the Relative and Absolute Clocks), puts all Transactions back in the latent pool, and performs the "input phase" with the model in its current state of definition.

Having accomplished this, the Processor goes to the next card for its next instruction. Assume that a START Card has been placed after the CLEAR Card in the deck. This is a signal that the Processor is to perform a simulation with the model. The Termination Counter is initialized to the value supplied as the A Operand on that START Card, and the Processor then goes into execution of the Clock Update Phase, then the Scan Phase, etc. Eventually, the Termination Counter is again decremented to zero, the simulation shuts off, and system statistics are outputted. The Processor then returns to the deck for its next instruction, and so on.

As previously indicated, the processing in Case Study 2D was performed in a series of three consecutive batch runs. With the CLEAR Card, it is now possible to accomplish the same three simulations in a single batch session. Figure 2.37(a) shows the coding sheet prepared for the corresponding job. Figure 2.37(b) displays the extended program listing produced by the Processor when the job is run. Note these features of Figure 2.37(b).

(1) The GENERATE Block for assemblers (Block 1) occupies the symbolic Location KEY.

(2) After the first START Card (Card 19), the GENERATE Block occupying the Location KEY has been re-defined. Immediately after that, the Processor has printed the message, MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD. (There is no "Card Number" for this message, because there is no corresponding card in the job deck.) Despite this message, the Processor does not consider the multiple use of a Block Location Name to be an error. It outputs the message only as a warning, in case the analyst unintentionally assigned the same Location Name to two separate Blocks in the model. Following the inserted

---

(2s) GENERATE Blocks can only be replaced by other GENERATE Blocks. Except for this restriction, however, the newly-defined Block does not have to be of the same type as the Block it is replacing. For example, a QUEUE Block can be replaced with an ADVANCE Block, if this suits the purposes of the analyst.

(2t) It is also possible to extend the currently existing model by placing, between START Cards, a series of punchcards describing one or more self-contained model segments which were not previously included as part of the overall model.

LOCATION	OPERATION	A,B,C,D,E,F →
	SIMULATE	
*		
*	MODEL SEGMENT 1	
*		
KEY	GENERATE	,,,,4 PROVIDE 4 ASSEMBLERS
BACK	ADVANCE	30,,5 ASSEMBLE NEXT WIDGET
	SEIZE	OVEN CAPTURE THE OVEN
	ADVANCE	8,,2 USE THE OVEN
	RELEASE	OVEN FREE THE OVEN
	TRANSFER	,BACK GO DO THE NEXT ASSEMBLY
*		
*	MODEL SEGMENT 2	
*		
	GENERATE	2400 TIMER ARRIVES AFTER 5 DAYS
	TERMINATE	1 SHUT OFF THE RUN
*		
*	CONTROL CARDS AND BLOCK REDEFINITIONS	
*		
	START	START THE 1ST RUN
KEY	GENERATE	,,,,5 RE-CONFIGURE FOR 2ND RUN
	CLEAR	CLEAR FOR 2ND RUN
	START	1 START THE 2ND RUN
KEY	GENERATE	,,,,6 RE-CONFIGURE FOR 3RD RUN
	CLEAR	CLEAR FOR 3RD RUN
	START	1 START THE 3RD RUN
	END	RETURN CONTROL TO OPERATING SYSTEM

(a) Completed Coding Sheet for Model

```
BLOCK                                                                  CARD
NUMBER   *LOC    OPERATION  A,B,C,D,E,F,G          COMMENTS           NUMBER
                 SIMULATE                                                 1
         *                                                                2
         *       MODEL SEGMENT 1                                          3
         *                                                                4
1         KEY    GENERATE   ,,,4          PROVIDE 4 ASSEMBLERS          5
2         BACK   ADVANCE    30,5          ASSEMBLE NEXT WIDGET          6
3                SEIZE      OVEN          CAPTURE THE OVEN              7
4                ADVANCE    8,2           USE THE OVEN                  8
5                RELEASE    OVEN          FREE THE OVEN                 9
6                TRANSFER   ,BACK         GO DO THE NEXT ASSEMBLY       10
         *                                                               11
         *       MODEL SEGMENT 2                                         12
         *                                                               13
7                GENERATE   2400          TIMER ARRIVES AFTER 5 DAYS    14
8                TERMINATE  1             SHUT OFF THE RUN              15
         *                                                               16
         *       CONTROL CARDS AND BLOCK RE-DEFINITIONS                  17
         *                                                               18
                 START      1             START THE 1ST RUN             19
1         KEY    GENERATE   ,,,5          RE-CONFIGURE FOR 2ND RUN      20
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
                 CLEAR                    CLEAR FOR 2ND RUN             21
                 START      1             START THE 2ND RUN             22
1         KEY    GENERATE   ,,,6          RE-CONFIGURE FOR 3RD RUN      23
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
                 CLEAR                     CLEAR FOR 3RD RUN            24
                 START      1             START THE 3RD RUN             25
                 END        RETURN CONTROL TO OPERATING SYSTEM           26
```

(b) Extended Program Listing

Figure 2.37 A Model for Simulating Three Alternative Case Study 2D Configurations in a Single Batch Run

warning message comes the CLEAR Card (Card 21), the next START Card (Card 22), and so on.

Realize that the re-definition of the Block at Location KEY is not effective until after the simulation started by the first START Card has been performed. The Processor first reads in the entire model, producing the extended program listing. Then, after more internal processing, the model is executed. Nevertheless, it is while the model is being read, and before the simulation (or simulations) begin, that the warning messages about multiple card definitions are produced.

Figure 2.38 shows the output resulting from the simulation with the Figure 2.37 model. The key feature of Figure 2.38 is that, for the 5 and 6 assembler configurations, the results differ from those shown in Figure 2D.3(b) and (c). Why does this turn out to be the case? When three separate batch runs were used, the random number stream for each configuration always began at the same point. In the case of one batch run, the random number stream for each next configuration begins at whatever point it had reached by the end of the previous simulation. It is precisely because a different sequence of random numbers is in effect that the statistics for the 5 and 6 assembler configurations differ from those shown in Case Study 2D.

It is evident, then, that the CLEAR Card does not cause "the random number stream" to be restored to its initial state.

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
OVEN	.785	236	7.991	5	

(a) Oven Statistics When 4 Assemblers Are Used

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
OVEN	.942	281	8.053	6	

(b) Oven Statistics When 5 Assemblers Are Used

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
OVEN	.989	296	8.020	1	

(c) Oven Statistics When 6 Assemblers Are Used

Figure 2.38 Selected Program Output Resulting When the Figure 2.37 Deck Was Run

2.34. Exercises

2.34.1 Compute the return-after-costs for the Case Study 2D data shown in Figure 2.38. Is the conclusion still that 5 assemblers is the optimal number to use?

2.34.2 In a particular machine shop, a single machine is used to polish castings. The steps required to polish a casting are shown below, where the times required are indicated, in minutes, after each step.

(1) Fetch raw casting from a storage area (12±3).

(2) Load raw casting on the polishing machine (10±4).

(3) Carry out polishing phase 1 (80±20).

(4) Re-position the casting on the machine for additional polishing (15±7).

(5) Carry out polishing phase 2 (110±30).

(6) Unload the finished casting from the machine (10±4).

(7) Store the finished casting (12±3), then return to step (1).

The castings are too heavy to be handled by the operator of the polishing machine. He requires the services of an overhead crane to assist in the handling process. In particular, the overhead crane is needed at Steps 1, 2, 4, 6, and 7 above.

There is only one overhead crane available. The polishing machine operator does not have the exclusive use of this crane. It is also on call to perform service at other machines in the shop. "Other calls" for the crane occur every 39±10 minutes. The time required for the crane to service one of these other calls is 25±10 minutes.

Build a model of the system as described. Design the model to collect data about the time the polishing machine operator spends waiting for the crane. Distinguish between time spent waiting for the crane to perform step (4), and time spent waiting to perform step (6). (Assume that when the crane has been captured for step (6), it is not then released until after step (2) has again been performed.) Also gather waiting time information relative to "other calls" on the crane.

Run the model for 400 hours of simulated time. Compare and contrast the waiting experience at the three indicated points in the model for each of these two alternative queue disciplines.

(1) First-come, first-served

(2) Polishing machine operator has highest priority for use of the crane.

What must be true for the "priority-distinction" case to "make a difference" in the operation of the system? How likely is it that this condition will frequently be in effect?

Did you use a Facility to model the polishing machine? Why, or why not? If you did use a Facility, consider how your model would perform if the SEIZE-RELEASE Block pair were removed from it. Would your model still be "true" to the system as described?

2.34.3 In a particular model, the simulation begins at midnight and continues through a series of consecutive 24-hour days. Transactions are to be brought into the model each day of simulated time at 1 a.m., 4 a.m., 10 a.m., and 2 p.m. Whenever a Transaction enters the model, it is to be routed to a "QUEUE STACK" Block. Under the following alternative assumptions, show a Block Diagram segment which brings Transactions into the model at the proper times.

(a) Transactions are to move into the "QUEUE STACK" Block exactly "on time".

(b) Transactions do not always enter the model exactly on time, but are subject to a delay of 10±10 minutes, uniformly distributed. Any delays which do occur, however, are independent of each other. For example, the time of arrival of the "1 a.m. Transaction" on the third day in no way influences the time of arrival of the "4 a.m. Transaction" that day, or of the "1 a.m. Transaction" the next day, etc.

A possible interpretation for this situation is that Transactions simulate planes arriving at an airport. When planes arrive, they join a "stack" of planes awaiting clearance to land. Planes often do not arrive "on time"; hence, part (b) is considerably more realistic than part (a).

2.35. Entities to Simulate Parallel Servers: Storages

Two or more servers often work side-by-side, performing a similar service. Such servers can either be people, or things. Here are some examples of two or more people in the category of side-by-side, or parallel, servers.

(a) barbers
(b) supermarket checkout girls
(c) beauty shop operators
(d) clerks at a tool crib
(e) ticket-takers at the theatre

Here are some examples of things which perform a similar service, and which might be available in quantities of two or more.

(a) tugs for berthing and de-berthing ships at a harbor
(b) overhead cranes for moving heavy castings from molds to machines
(c) parking places in a parking lot
(d) a supply of a particular type of spare part, sitting on a shelf
(e) card punches in a computing center

A Facility can be used in GPSS to simulate a single server. Two or more side-by-side servers could be simulated in a GPSS model, then, with two or more Facilities located "side-by-side", i.e., in parallel. Indeed, it is sometimes necessary to use parallel Facilities to model parallel servers. This is the approach usually taken when the individual servers are heterogeneous, that is, are characterized by different properties, such as performing service at different rates. Many times, however, parallel servers are assumed to be homogeneous. This means, roughly speaking, that such

servers share a common set of properties. For example, the rate at which checkout girls serve customers in a supermarket may not depend on the particular checkout girl involved.

GPSS provides a special entity for simulating homogeneous parallel servers. The name "Storage" is used for this entity. There can be many different Storages in a model. With 64K bytes of memory, for example, the normal number of Storages available in a model is 35.

The analyst supplies names to distinguish among the various Storages he uses. The naming convention used for Facilities and Queues also applies to Storages. Names can be numeric or symbolic. When numeric, they must be positive, whole numbers. When symbolic, names are composed of from 3 to 5 alphanumeric characters, with the restrictions that the first 3 be alphabetic.

The number of servers which each particular Storage simulates must be defined in the model by the analyst. In this sense, one speaks of the capacity of a Storage. The method for defining Storage capacity and using the Storage entity is described in the next section.

2.36. Using One or More Parallel Servers: The ENTER Block and the LEAVE Block

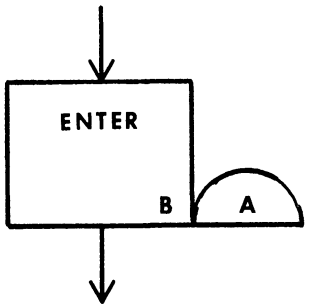
The approach to simulating the use of one or more parallel servers with a Storage is similar to that for simulating use of a single server with a Facility. The entities which place demands on Storages for service are Transactions. Assume a Transaction is to engage one or more of the servers being simulated with a Storage. The Transaction does this by moving (or attempting to move) into a particular Block related to the Storage. The Block has these features.

(1) If the one or more servers needed are not currently available, the Transaction is denied entry to the Block. (It is reasonable that a Transaction may require the simultaneous services of two or more servers. A ship may need two or more tugs, for example, to pull it into a berth.)

(2) If servers in the required number are available, the Transaction is permitted to enter the Block. This causes the underlying Block subroutine to be executed. As a consequence, the status of the Storage is updated.

Whereas a Facility is either in use or not, the status of Storages is somewhat more complicated. The extent to which a Storage is in use must be considered. The number of parallel servers who are currently engaged is termed the content of the Storage. The number of servers not currently engaged is the remaining capacity. Of course, the sum of current content and remaining capacity equals the number of parallel servers the Storage simulates.

The Block which has the features indicated above is the ENTER Block. It is shown with its A and B Operands in Figure 2.39. The A Operand is the name of a Storage

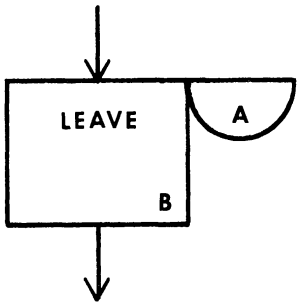


Operand	Significance	Default Value or Result
A	The name (numeric or symbolic) of a Storage	Error
B	The number of servers to be engaged	1

Figure 2.39 The ENTER Block and Its A and B Operands

simulating the parallel servers a Transaction intends to use. The B Operand indicates the number of those servers required by the Transaction. The default value for the B Operand is 1, indicating that only one of the servers is needed.

Movement of a Transaction into an ENTER Block, then, simulates engaging one or more parallel servers. In complementary fashion, movement of a Transaction into another particular Block simulates disengaging one or more parallel servers. The Block which accomplishes this is the LEAVE Block. It is shown with its A and B Operands in Figure 2.40. When a Transaction moves into the LEAVE Block, the current content of the Storage named by the A Operand is decreased by an amount equal to the B Operand. The LEAVE



Operand	Significance	Default Value or Result
A	The name (numeric or symbolic) of a Storage	Error
B	The number of servers to be disengaged	1

Figure 2.40 The LEAVE Block and Its A and B Operands

never denies entry to a Transaction. An error condition results if execution of the LEAVE Block subroutine causes the current content of the referenced Storage to become negative. If this happens, an error message is outputted and the simulation is stopped.

If the current content of a Storage is greater than zero, any Transaction in a model can decrease the content by moving into a LEAVE Block which references the Storage. An error results only if the content is thereby made negative. This is in sharp contrast to the situation with Facilities. Recall that if a Facility is in use, only the Transaction which is the user of record can release it.

The above paragraph indicates that Transactions can "leave" Storages they never "entered". This points out the need to avoid thinking of a Storage as a "place in which things are stored". Unfortunately, such terms as "storage", "enter", "leave", "current content", and "remaining capacity" lead initially to the view of a Storage as a place. The Storage entity, however, is entirely abstract. There is no restriction that a Storage cannot be "left" by a Transaction unless that same Transaction previously "entered" it. If a Transaction ever does leave a Storage it did not enter, model logic usually requires that some other Transaction, in complementary fashion, enters that same Storage, and never leaves it.

The ENTER and LEAVE Blocks are of the type whose execution later causes the scan of the Current Events Chain to be re-started. Assume that one of the Blocks a Transaction moves through before finally stopping is an ENTER and/or a LEAVE Block. Then, rather than processing the sequential Transaction on the Current Events Chain as its next step, the Processor re-starts the scan of the Current Events Chain.

The capacity of the various Storages in a model is defined by use of one or more Storage Capacity Definition Cards. Two forms are available for these cards. The first form requires that a separate card be used for each Storage. Table 2.21 shows the card format. Like most other cards, the Storage Capacity Card is divided into the Location, Operation, and Operands Fields. Entered in the Location Field is the number or symbolic name of the Storage. The Operation Field contains the word STORAGE. The A Operand specifies the capacity of the Storage. Two examples are shown in Figure 2.41, where the

Punchcard Field	Information Supplied in the Field
Location	The name (numeric or symbolic) of a Storage
Operation	Literally, the word STORAGE
Operands	
A	The capacity of the Storage

Table 2.21 Card Format for Single Storage Capacity Definition

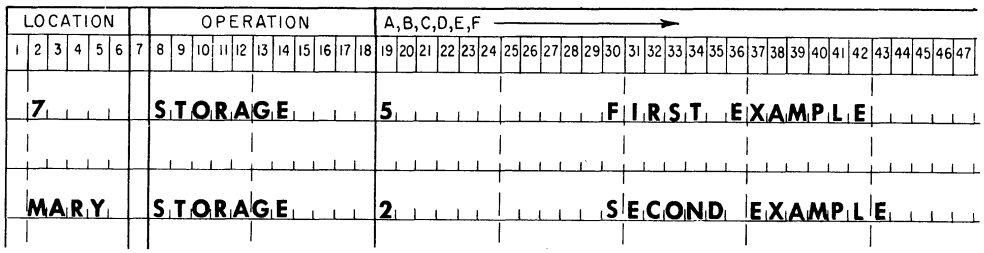


Figure 2.41 Examples Showing Use of the Table 2.21 Card Format

Storages 7 and MARY are defined to have capacities of 5 and 2, respectively.

The second form permits the analyst to define the capacity of two or more Storages with a single card. The card format is shown in Figure 2.42. As indicated,

Location	Operation	Operands
Not Used	STORAGE	$Sname_1,c_1/Sname_2,c_2/ \ldots /Sname_i,c_,/ \ldots /Sname_n,c_n$

Figure 2.42 Card Format for Multiple Storage Capacity Definitions

the Location Field is not used. The word STORAGE is entered in the Operation Field. The "basic unit" of information in the Operands Field is of the form "$Sname_i,c_i$", where $name_i$ is the numeric or symbolic name of the i-th Storage whose capacity is being defined, and c_i is the capacity. When symbolic names are used, a dollar sign ($) must be interposed between S and the symbolic name itself. Examples of "basic units" are "S5,8", "S$TUGS,3", and "S21,5". Slashes (/) are used to separate consecutive basic units. The first blank column encountered by the Processor in the Operands Field is interpreted as an "end of card" character. No information (except for comments) can be entered beyond card column 71. More than one of the Figure 2.42 STORAGE Cards can be used, if necessary.

Figure 2.43 provides several examples of multiple Storage capacity definitions. As indicated in the first example, it is not necessary that the numbers of the Storages involved be in ascending order on the STORAGE Card. The second example shows a mixture of Storages named numerically and symbolically. The third example reveals that Storages 4, 5, and 6 are each to have a capacity of 5. When Storages whose numbers form an uninterrupted sequence of integers are all to have the same capacity, the corresponding entries in the STORAGE Card can be compacted to the form "Si-Sk,c", where i and k are the smallest and largest numbers, respectively, in the range of Storage numbers, and c is the common capacity. The fourth example in Figure 2.43 repeats the third example,

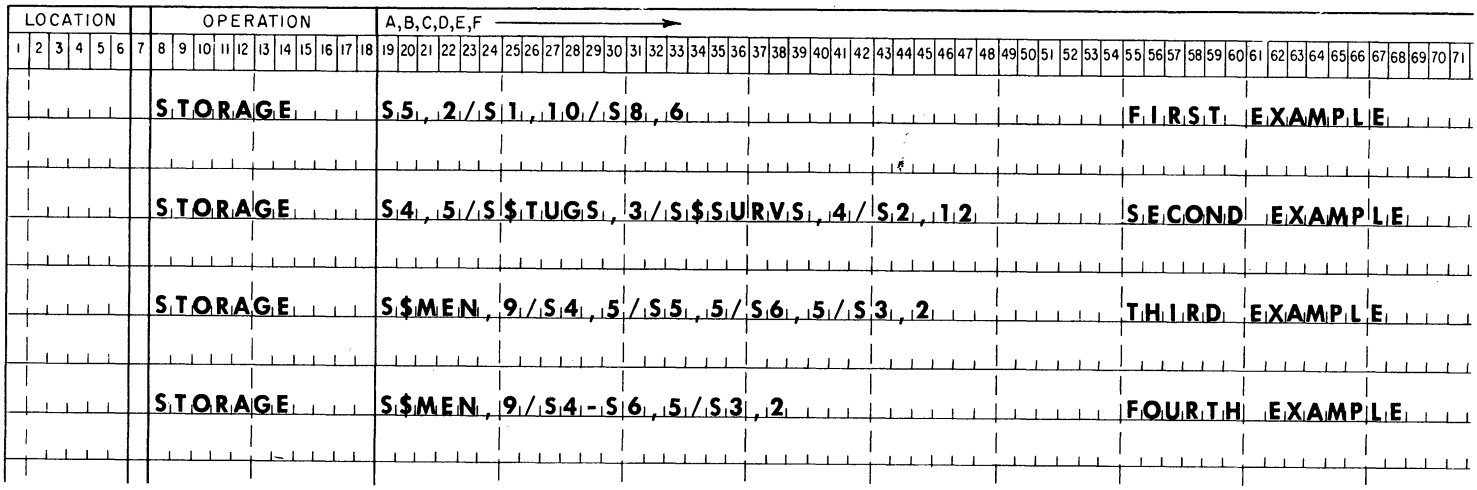


Figure 2.43 Examples Showing Use of the Figure 2.42 Card Format

making use of this compact form.

Although it is not strictly necessary, the practice in this book will be to place the Storage Capacity Definitions at an early point in the card deck, after the SIMULATE Card but before the first card which is a Block image.

If the analyst forgets to supply a capacity definition for one or more Storages in a model, the Processor will assume a capacity of $2^{31}-1$ (i.e., 2,147,483,647).

One possibility for re-configuring a model, before continuing to simulate with it in the context of a single batch run, is to re-define one or more Storage capacities. This can be done by including in the card deck, after the START Card for the "preceding simulation" and before the START Card for the "next simulation", the new capacity definitions. The next case study illustrates this procedure (see Figure 2E.2).

It is natural to wonder what difference there is between a Facility and a Storage with a capacity of 1. There are two differences.

(1) A Facility can only be released by the Transaction which is the seizer of record.

(2) A Facility can be pre-empted, whereas a Storage cannot.

Pre-emption involves removing the current seizer from a Facility, so the Facility can immediately respond to a more urgent need for the service it simulates. Pre-emption is an important concept in the design of computing systems, as well as in some other systems. Simulation of pre-empts in GPSS is discussed in Chapter 6.

2.37. One-Line, Multiple-Server Queuing Systems

Probably the most-frequently occurring pattern for modeling the use of Storages is that shown in Figure 2.44. In addition to the ENTER-ADVANCE-LEAVE sequence in the figure, the ENTER Block has been sandwiched between a QUEUE-DEPART Block pair, to gather information about the waiting experience ahead of the ENTER Block. Of course, the QUEUE-DEPART Block pair is optional.

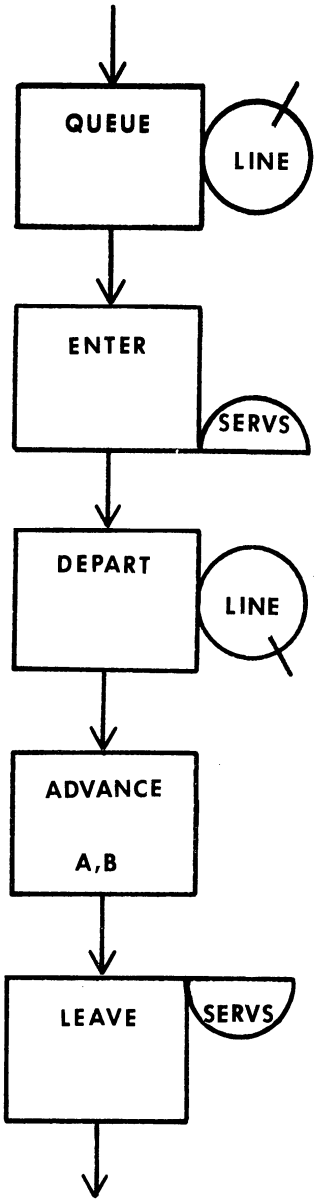


Figure 2.44 A Frequently-Occurring Pattern for Modeling with Storages in GPSS

Two aspects of the queuing system modeled with the Block sequence in Figure 2.44 merit discussion. The first involves the way a user selects a server; the second involves the way a server selects a user.

Quite often, when parallel servers are involved, a separate waiting line forms ahead of each server. When a would-be user arrives he surveys the situation, then decides which line to join. In this sense, the user discriminates in his selection of a server. Such discrimination is not possible when parallel servers are simulated with a Storage. The servers a Storage represents have no individual identity; there is consequently no way to associate a separate waiting line with each server. In essence, then, a would-be user waiting at a Storage necessarily takes the attitude, "when my turn comes, I'll take whichever server is available". This is equivalent to having only one waiting line ahead of the parallel servers. When a would-be user arrives, he simply joins the line. After reaching the head of the line, he goes to whichever server is free

next. This situation is illustrated in Figure 2.45.

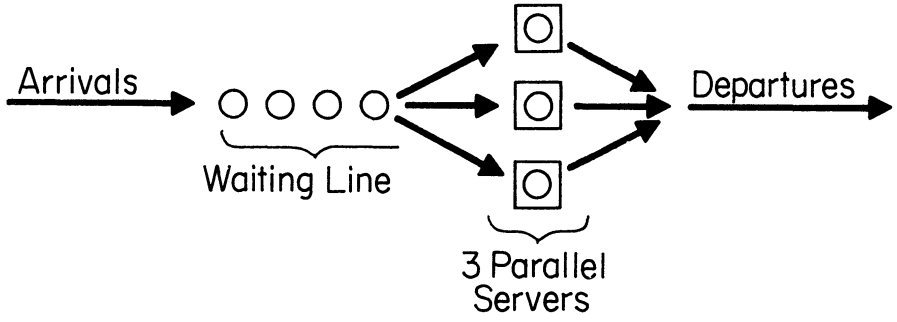


Figure 2.45 Illustration of a One-Line, Multiple-Server Queuing System

Interestingly enough, although the users cannot discriminate in Figure 2.44, the servers can. That is, the servers can regard the "one" waiting line as being composed of a variety of priority classes. When a server becomes available, he then selects a user by employing a "first-come, first-served, within priority class" queue discipline. The servers a Storage simulates practice such discrimination in GPSS "automatically" if the Transactions waiting for them have different Priority Levels. This is because Transactions waiting to move into the Figure 2.44 ENTER Block are arranged on the Current Events Chain in order of decreasing Priority Level. When a LEAVE Block is executed, thereby freeing one or more servers, the scan of the Current Events Chain is restarted. The available server is then engaged by the highest-priority Transaction which has been waiting the longest. This situation is illustrated in Figure 2.46. As the figure suggests, would-be users do not simply "go to the back of the line" when they

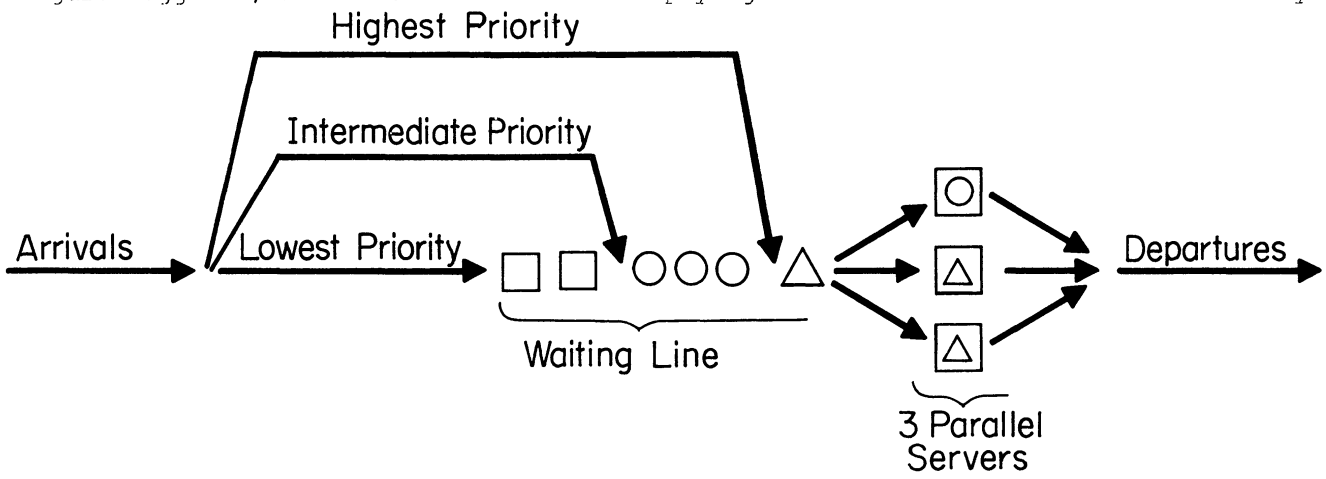


Figure 2.46 A One-Line, Multiple-Server Queuing System with Priority Distinctions

arrive; instead, they "merge into the line as the last member in their priority class". Figure 2.46, then, is simply a repetition of Figure 2.45, except that distinctions are made among user priorities.

The one-line, multiple-server scheme is frequently used in real situations. This tends especially to be true when the items waiting for service are inanimate and, for all practical purposes, identical. This would be the case, for example, for parts moving

from work station to work station on a production line. Even though the work stations might be manned by two or more people working in parallel, work would very likely be fed to them from a single line.

The one-line, multiple-server queuing system is rather frequently used, too, when those waiting for service are people, not things. It is used in some banks, for example, where it is called the "Quickline" system. A person entering such a bank, rather than selecting a teller's window and joining the line at that window, goes to the back of a single line which feeds all windows. This approach makes it impossible to get "stuck" behind a customer with an unusually high service time requirement (customers opening new accounts, for example, or buying and signing traveler's checks). The average time spent waiting for service is lowered as a consequence.

Quite often, when parallel servers are involved, a separate waiting line does form ahead of each server. Modeling this multiple-line, multiple-server queuing system is substantially more complicated than modeling the one-line, multiple-server scheme. A queue selection criterion must be introduced, for example. When a new arrival comes, this criterion is then used by the arrival to determine which waiting line to joing. The possibility of line jumping should also be allowed. When a line is moving too slowly, those toward the end of the line may choose to move to another line. Modeling multiple-line, multiple-server queuing systems in GPSS is deferred until Chapter 4.

2.38. Case Study 2E: A Problem in Production Management

(1) Statement of the Problem

In a certain garment factory, 50 sewing machines are operated 8 hours a day, 5 days a week. Each of the machines is subject to random failure. Whenever a machine fails, it is replaced with a backup machine immediately, or as soon as one becomes available. Meantime, the failed machine is sent to an in-house repair shop, where it is repaired and then assumes the status of an available backup machine.

The four stages in this closed machine cycle are shown in Figure 2E.1 for a

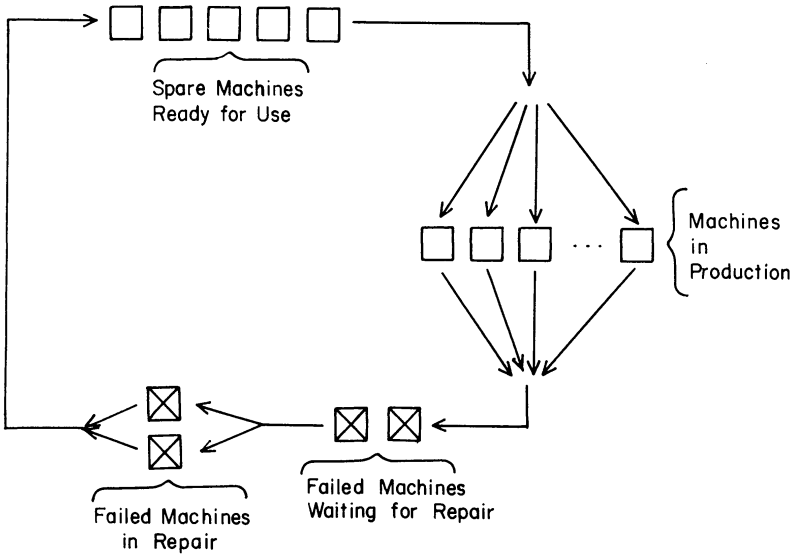


Figure 2E.1 Graphic Description of the Production Problem

hypothetical situation. In the figure, an "open square" is a machine which is either in use, or is in backup status. An "X-ed out square" is a machine which is either in repair, or is waiting for repair. Two machines are shown in repair, implying that there are two repairmen. In total, 59 machines are in the system shown (2 in repair; 2 waiting for repair; 50 in production; and 5 available for backup purposes).

Management wants to know how many repairmen to hire for the repair shop, and how many backup machines to rent to supplement the 50 machines it owns. The objective is to minimize system costs. Wages for repairmen are $3.75 per hour, and backup machines are available on a long-term rental basis at a cost of $30 per day. The hourly cost penalty for having fewer than 50 machines in production is estimated at $20 per machine. This penalty reflects lost production.

Based on past experience, the time required to repair a failed machine is 7 ± 3 hours, uniformly distributed. When a machine is put into use, its running time before failure is 157 ± 25 hours, uniformly distributed. The time required to remove a failed machine from its production station and replace it with a backup machine is negligibly small.

No distinctions are made either among repairmen, or among machines cycling through the system. That is, the distribution of repair time is independent of the particular repair man involved. And, the distribution of running time between failures is identical for both the owned and the rented machines.

The charge for rental of backup machines is independent of machine usage. For this reason, no attempt is made to maximize the number of owned machines in production at any given time. The implication is that a machine is taken out of production only when it fails.

Build a GPSS model for this sytem, then use it to investigate the cost consequences of the various "hire, rent" alternatives available to management. Use the model to determine the "hire, rent" alternative which minimizes the average daily cost of running the system.

(2) An Approach Leading to Solution

As suggested in Case Study 2D, it is oftentimes useful to approach the modeling process by first identifying the constraints in a system, then deciding on various GPSS entites to simulate these constraints. There are three constraints in the Figure 2D.1 system:

(1) The number of repairmen

(2) The maximum number of sewing machines which are to be in production at any one time

(3) The total number of sewing machines cycling through the system

It is convenient to choose GPSS Storages to simulate the first two constraints, and Transactions to simulate the third constraint. The reasoning behind such choices, roughly speaking, is that the "repairmen" and the "in-production machines" are stationary, occupying fixed points as suggested in Figure 2D.1. On the other hand, the sewing machines themselves are "dynamic", moving from point to point as they circulate through the system.

To consider further the feasibility of this approach, consider the history of a particular machine as it makes one complete cycle through the Figure 2D.1 system. Suppose that the machine is in "backup status", and that 50 machines are already in production. Then the Storage NOWON (now-on, for the number of machines now in use) is full, and the backup machine is "held in place" because it cannot enter that Storage. Eventually, the machine in backup status is given the opportunity to go into production. The Transaction simulating it determines this when, after repeated attempts to enter the Storage NOWON, an attempt is successful. Moving through the ENTER Block to ADVANCE, the Transaction then simulates running time of that machine before it fails.

Upon failure, the Transaction leaves the Storage NOWON, thereby permitting another backup machine to go into production. The Transaction then waits (if necessary) to enter the Storage MEN (for repairmen). Entering that Storage, it engages a repairman while repairtime is simulated. Eventually leaving that Storage, it frees the repairman so he can begin repair on another failed machine. Meanwhile, the Transaction cycles back to the point in the model where it again begins to attempt to enter the Storage NOWON.

The total number of machines cycling through the system equals the 50 machines owned, plus the number of machines rented for backup purposes. This number of Transactions can be created at the start of the simulation by use of the Limit Count on the GENERATE Block. The only other thing needed in the model is a Timer.

It might be noted that the Storage NOWON is not really used here to simulate 50 "servers". In a sense, it simulates the 50 "positions" available for occupation by up to 50 sewing-machine operators.

This problem, although inherently complicated, can be modeled then with relative ease in GPSS. A FORTRAN model for the same system is considerably more complicated.[2u]

(3) Table of Definitions

Time Unit: 1 Hour

GPSS Entity	Interpretation
Transactions	
Model Segment 1	Sewing Machines
Model Segment 2	The timer
Storages	
MEN	The side-by-side repairmen
NOWON	The Storage whose capacity (50) is the maximum number of sewing machines ever to be in production simultaneously

Table 2E.1 Table of Definitions for Case Study 2E

[2u] A FORTRAN model of this system is documented in "Three Computer Models for Probability Applications," by T.J. Schriber, FORTRAN Applications in Business Administration, Vol. II (The University of Michigan, 1971), pp. 349-422. [Available only from Ulrich's Books, Inc., Ann Arbor, Michigan, 48104.]

(4) Block Diagram

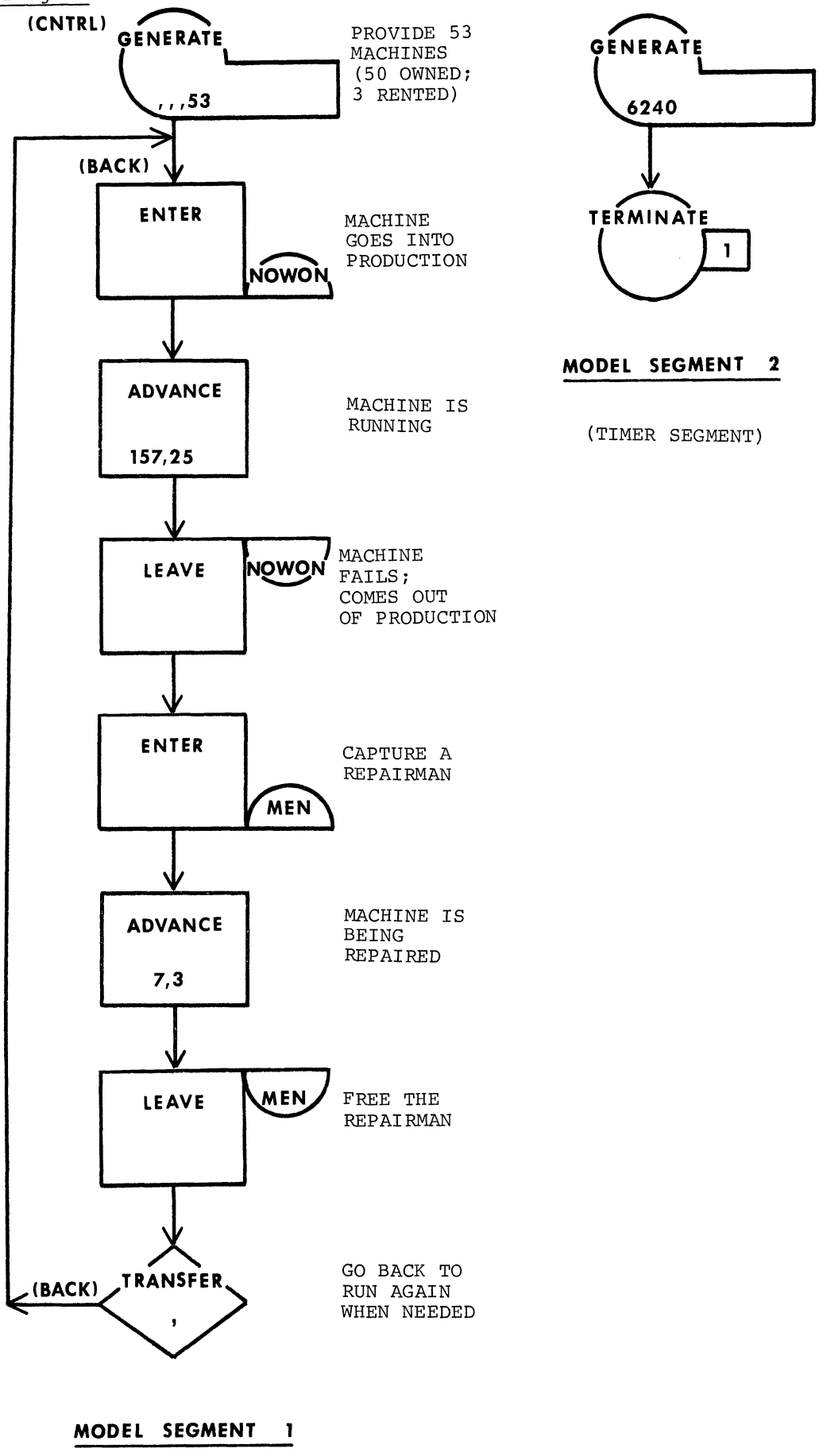


Figure 2E.2 Block Diagram for Case Study 5 (4 Backup Machines Rented)

```
BLOCK                                                                    CARD
NUMBER   *LOC    OPERATION  A,B,C,D,E,F,G          COMMENTS              NUMBER
                 SIMULATE                                                    1
         *                                                                   2
         *       DEFINE STORAGE CAPACITIES                                   3
         *                                                                   4
                 STORAGE    S$MEN,3/S$NOWON,50    3 MEN; MAX OF 50 MACH'S RUNNING  5
         *                                                                   6
         *       MODEL SEGMENT 1                                             7
         *                                                                   8
1         CNTRL GENERATE   ,,,53       PROVIDE 53 MACHINES (50 OWNED;  3 RENTED)  9
2         BACK  ENTER      NOWON       MACHINE GOES INTO PRODUCTION          10
3               ADVANCE    157,25      MACHINE IS RUNNING                    11
4               LEAVE      NOWON       MACHINE FAILS;  COMES OUT OF PRODUCTION  12
5               ENTER      MEN         CAPTURE A REPAIRMAN                   13
6               ADVANCE    7,3         MACHINE IS BEING REPAIRED             14
7               LEAVE      MEN         FREE THE REPAIRMAN                    15
8               TRANSFER   ,BACK       GO BACK TO RUN AGAIN WHEN NEEDED      16
         *                                                                   17
         *       MODEL SEGMENT 2                                             18
         *                                                                   19
9               GENERATE   6240         TIMER COMES AFTER 3 YEARS  40-HOUR WEEKS)  20
10              TERMINATE  1            SHUT OFF THE RUN                      21
         *                                                                   22
         *       CONTROL CARDS, BLOCK RE-DEFINITIONS, AND STORAGE CAPACITY CHANGES  23
         *                                                                   24
                 START      1            START THE 1ST RUN                     25
1         CNTRL GENERATE   ,,,54       SET RENTED MACHINES = 4 FOR 2ND RUN    26
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
                 CLEAR                    CLEAR FOR 2ND RUN                  27
                 START      1            START THE 2ND RUN                     28
1         CNTRL GENERATE   ,,,55       SET RENTED MACHINES = 5 FOR 3RD RUN    29
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
                 CLEAR                    CLEAR FOR 3RD RUN                  30
                 START      1            START THE 3RD RUN                     31
                 STORAGE    S$MEN,4      SET REPAIRMEN HIRED = 4 FOR 4TH RUN   32
1         CNTRL GENERATE   ,,,53       SET RENTED MACHINES = 3 FOR 4TH RUN    33
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
                 CLEAR                    CLEAR FOR 4TH RUN                  34
                 START      1            START THE 4TH RUN                     35
1         CNTRL GENERATE   ,,,54       SET RENTED MACHINES = 4 FOR 5TH RUN    36
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
                 CLEAR                    CLEAR FOR 5TH RUN                  37
                 START      1            START THE 5TH RUN                     38
1         CNTRL GENERATE   ,,,55       SET RENTED MACHINES = 5 FOR 6TH RUN    39
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
                 CLEAR                    CLEAR FOR 6TH RUN                  40
                 START      1            START THE 6TH RUN                     41
                 STORAGE    S$MEN,5      SET REPAIRMEN HIRED = 5 FOR 7TH RUN   42
1         CNTRL GENERATE   ,,,53       SET RENTED MACHINES = 3 FOR 7TH RUN    43
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
                 CLEAR                    CLEAR FOR 7TH RUN                  44
                 START      1            START THE 7TH RUN                     45
1         CNTRL GENERATE   ,,,54       SET RENTED MACHINES = 4 FOR 8TH RUN    46
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
                 CLEAR                   CLEAR FOR 8TH RUN                   47
                 START      1            START THE 8TH RUN                     48
1         CNTRL GENERATE   ,,,55       SET RENTED MACHINES = 5 FOR 9TH RUN    49
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
                 CLEAR                    CLEAR FOR 9TH RUN                  50
                 START      1            START THE 9TH RUN                     51
                 END        RETURN CONTROL TO OPERATING SYSTEM                52
```

Figure 2E.3 Extended Program Listing for Case Study 2E

(6) Program Output

STORAGE	CAPACITY	AVERAGE CONTENTS	AVERAGE UTILIZATION	ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENTS	MAXIMUM CONTENTS
MEN	3	2.185	.728	1924	7.087		3
NOWON	50	49.182	.983	1974	155.471	50	50

(a) Storage Statistics for "3 Hired, 3 Rented" Configuration

STORAGE	CAPACITY	AVERAGE CONTENTS	AVERAGE UTILIZATION	ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENTS	MAXIMUM CONTENTS
MEN	3	2.180	.726	1934	7.036	3	3
NOWON	50	49.486	.989	1984	155.643	50	50

(b) Storage Statistics for "3 Hired, 4 Rented" Configuration

STORAGE	CAPACITY	AVERAGE CONTENTS	AVERAGE UTILIZATION	ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENTS	MAXIMUM CONTENTS
MEN	3	2.184	.728	1951	6.986	3	3
NOWON	50	49.629	.992	2003	154.611	49	50

(c) Storage Statistics for "3 Hired, 5 Rented" Configuration

Figure 2E.4 Selected Program Output for Case Study 2E

(7) Discussion

Strategy for Use of the Model

The model is to be used to estimate the hire/rent combination which minimizes the average daily cost of running the factory. Some thought should be given, then, to the way the cost is expected to behave as various hire/rent combinations are considered. For a fixed number of repairmen, average daily cost will vary with "machines rented" as shown in Figure 2E.3. With too few rented machines, costs are high because of the lost-production penalty. With too many rented machines, costs are also high because the rental fee is being paid for more backup machines than are necessary. Somewhere between these two extremes, the cost can be expected to pass through a minimum.

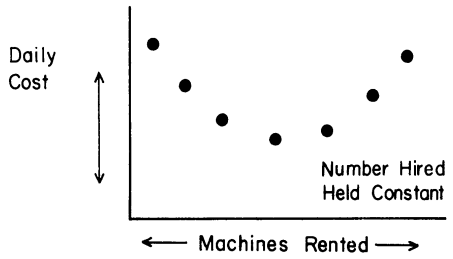


Figure 2E.3 Dependence of Daily Cost on Number of Machines Rented

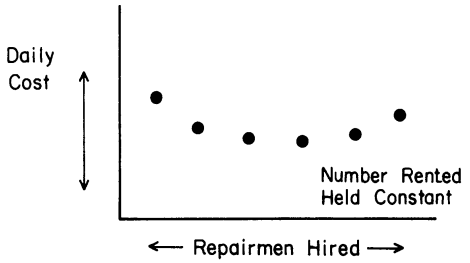


Figure 2E.4 Dependence of Daily Cost on Number of Repairmen Hired

Similarly, for a given number of rented machines, the influence of "repairmen hired" will follow the pattern in Figure 2E.4. With too few repairmen, high costs are experienced because of the lost-production penalty. With too many repairmen, costs are high because salaries must be paid, even though the utilization of the repairmen is low. It is reasonable to expect that cost goes through a minimum between these two extremes.

A three-dimensional perspective of the situation is suggested in Figure 2E.5.

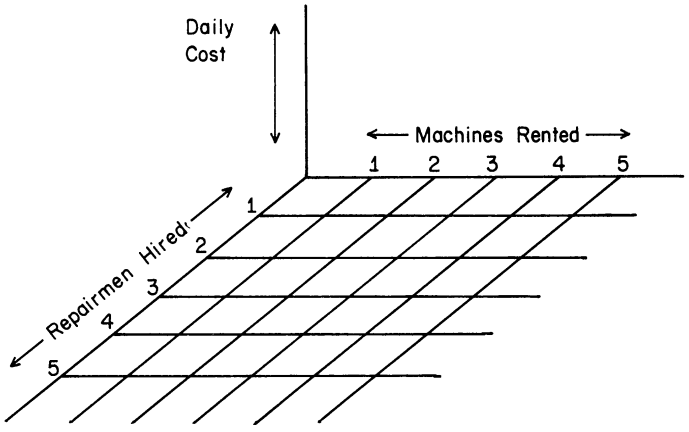


Figure 2E.5 A Three Dimensional Perspective of the Daily Cost Surface

Note that the mesh points in the grid laid down in the Figure 2E.5 "hired/rented" plane represent feasible (i.e., integer-valued) choices of values for the decision-variable

pair. Located above each of these mesh points, in the third dimension, is the expected daily cost corresponding to that particular hire/rent alternative (these points are not shown in Figure 2E.5). The collection of all such cost points forms a "cost surface" in the third dimension.[2v] Figures 2E.3 and 2E.4 are then simply contours through this cost surface.

It is easy to argue that the cost surface has a single low point. The search for the optimal hire/rent combination, then, is essentially a search for this lowest point. This search might be conducted in a variety of ways. Consider two of the possibilities.

(1) A "univariate searching strategy" might be employed. Briefly, this strategy involves moving through the decision-variable space from low point to low point on successive contours until all more promising directions of movement have been exhausted. Further details about this method are available elsewhere.[2w] This method is especially useful if it can be conducted by interactive use of the model from a terminal, or if it is made a part of the model itself. Neither of these possibilities will be further explored here.

(2) The model can be submitted for a run in which a simulation is performed for each of a series of methodically-placed hire/rent combinations. A relatively "coarse" grid can be investigated at first, by spacing the combinations relatively far apart. Examination of the output then usually reveals which region holds promise for further investigation. In the next run, then, a simulation is performed for hire/rent combinations which are placed closer together in this smaller, promising region. If necessary, this process can be repeated, with the search being restricted to increasingly smaller regions. Eventually, then, an estimate of the optimal combination can be made.

The second method will be followed here. In this simple problem, however, it is not necessary to start with a coarse grid, because simple computations make it possible to estimate approximately where the low-cost point lies. Notice first of all that the daily costs of a repairman, a backup machine, and an out-of-production machine are \$30, \$30, and \$160, respectively. The cost due to lost production is disproportionately high, meaning that the optimal hire/rent combination will be one tending to keep lost production time low.

Now consider conditions under which there is little lost production. From the data, average machine lifetime is 157 hours, and average repair time is 7 hours. Suppose a hypothetically "ideal" system is one in which no waiting occurs. That is, in such an ideal system, failed machines never have to wait before repair beings on them; and machines coming out of repair go back into service immediately. Each machine then has an average cycle time in the system of 164 hours. Machine utilization in this ideal system is about 0.95 [157/164 $\doteq$ 0.95]. If 50 machines are to be in use at all times (to eliminate lost production costs), a total of 52.6 machines must then cycle through the system [(52.6)(0.95) $\doteq$ 50]. With 50 machines in use, 2.6 machines would then be in repair. This, in turn, indicates that 2.6 repairmen would be required.

[2v] The points, rather than forming a "surface", are simply a collection of discrete values with no continuum joining them. It is nevertheless convenient to use the term "surface".

[2w] For example, see the source in Footnote 2u.

The above considerations are based on a hypothetically ideal situation. Of course, because of randomness, machines will experience some waiting, typically, at the two points where waiting can occur. Furthermore, it is not possible to rent 2.6 machines, or hire 2.6 repairmen. As a result, it seems reasonable to consider renting 3 or 4 or 5 backup machines, and hiring 3 or 4 or 5 repairmen. This suggests simulating with the nine combinations resulting from use of 3, 4, or 5 machines, and 3, 4, or 5 repairmen. As will be seen below, when the model was run with these nine configurations, the "4 hired, 4 rented" combination produced the lowest estimate of average daily cost.

Model Implementation

The Figure 2E.1 model was used to simulate with the nine indicated configurations in a single batch session. The deck makeup required to accomplish this is shown in Figure 2E.2. Referring to that figure, note that card number 5 sets the capacities of the Storages MEN and NOWON at 3 and 50, respectively, corresponding to "3 hired" and "as many as 50 machines in production simultaneously". Card 9 is the image of the GENERATE Block through which machine-Transactions enter the model. The D Operand is 53, which corresponds to the case of 3 rented machines (the D Operand equals the sum of machines owned and machines rented). The GENERATE Block's Location has been symbolically named CNTRL (for "control"), so that the Block can be re-defined between START Cards when necessary to reflect changes in the number of rented machines. After the first START Card, cards 26, 27, and 28 respectively re-configure the model for the case of 4 rented machines, clear the model of Transactions and statistics from the first simulation, and start the simulation for the "3 hired, 4 rented" combination. Cards 29, 30, and 31 repeat this pattern, resulting in a simulation for a "3 hired, 5 rented" combination. Card 32,then re-defines the capacity of the Storage MEN at 4. Card 33 modifies the GENERATE Block CNTRL to represent the "3 rented" situation. The model is then cleared with card 34, and card 35 starts the simulation for the "4 hired, 3 rented" combination. Continuing in this fashion, the remaining "hired, rented" combinations of "4,4", "4,5", "5,3", "5,4", and "5,5" are then run in that sequence.

Program Output

Figure 2E.6 shows model-outputted Storage statistics for the first three configurations studied. For each Storage, seven different pieces of information are printed out. The labels over this information are self-explanatory and will not be further described. Scanning Figure 2E.6 from (a) to (c), note that the AVERAGE UTILIZATION of the Storage NOWON moves from 0.983 to 0.989 to 0.992. Also note that at the time of the Figure 2E.4(a) output, there were no machines in repair (CURRENT CONTENTS of the Storage MEN is "blank"), and the factory was running at full capacity (CURRENT CONTENTS of the Storage NOWON is 50). Compare and contrast these values with the corresponding information in parts (b) and (c) of that figure.

Table 2E.2 summarizes the utilization of the Storage NOWON as a function of the nine different "hired, rented" alternatives studied with the Figure 2E.3 model. The observable trends are to be expected. For a given number of repairmen hired, utilization rises as more machines are rented. Similarly, for a given number of machines rented, utilization rises as more repairmen are hired.

		Machines Rented		
		3	4	5
Repairmen Hired	3	0.983	0.989	0.992
	4	0.989	0.993	0.995
	5	0.991	0.993	0.997

Table 2E.2 Utilization of the Storage NOWON as a Function of Various "Hired, Rented" Combinations

Table 2E.3 summarizes the costs for the various hire/rent combinations

		Machines Rented		
		3	4	5
Repairmen Hired	3	180	210	240
	4	210	240	270
	5	240	270	300

(a) Fixed Costs, $/day

		Machines Rented		
		3	4	5
Repairmen Hired	3	136	88	64
	4	88	56	40
	5	72	56	24

(b) Lost Production Costs, $/day

		Machines Rented		
		3	4	5
Repairmen Hired	3	316	298	304
	4	298	296	310
	5	312	326	324

(c) Total Costs, $/day

Table 2E.3 Daily Costs for the Various "Hired, Rented" Combinations Investigated

investigated. In part (a), the fixed costs for salaries and machine rentals are shown. Part (b) indicates the "variable costs", which are easily computed from the utilization of the Storage NOWON, and the lost-production penalty. Part (c) displays the sum of the fixed and variable costs. Examination of part (c) indicates that, based on information produced by these simulations, a "4 hired, 4 rented" combination minimizes the overall costs. Bear in mind, however, that the utilizations in Table 2E.2 are only estimates of the true, long-run utilizations for the indicated hire/rent combinations. This, in turn, means the costs in part (c) of Table 2E.3 are only estimates. If the variability in the estimator is high for the sample size used, the results could be somewhat misleading. For example, it could be that the true cost associated with a "4 hired, 3 rented" combination is lower than that for "4 hired, 4 rented". This topic is considered further in the next section.

2.39. GPSS Control Cards: The RESET Card

Initial model conditions may vary markedly from those in effect when a state of steady operation has been reached.(2x) For example, consider Case Study 2E. Under typical operating conditions, some of the in-production sewing machines will be nearing their time of failure, whereas others will have just recently been put into use, and so on. Hence, the "remaining running time before failure" of in-production machines will be spread at random across the range from zero (or close to it) to the maximum feasible

(2x) Some systems never do reach a state of steady operation. A good example of such a system is a bank. Between opening at 9:30 a.m. and closing at 3 p.m., conditions in a bank are strongly dependent on the time of day. Discussion in this section is limited to systems in which conditions do not exhibit such time dependency.

lifetime.

At the start of the simulation in the Figure 2E.2 model, however, the model behaves as though all 50 machines in production have just been turned on. Because the lifetime distribution of each sewing machine is 157±25 hours, none of these machines can fail before simulated time 132. This means that, initially, the "remaining running times" of in-production machines are very untypical. Yet, the various "hire, rent" consequences in that system should be studied only after the model reaches typical operating conditions; otherwise, misleading conclusions may be drawn from the simulation.

Consider the problem, then, of guaranteeing that the value or values of pertinent statistics are "typical". The statistic of interest in Case Study 2E is utilization of the Storage NOWON. (As explained in the case study discussion, variable cost in the problem depends on this statistic; variable cost, in turn, is critical in choosing the optimal operating condition.) Three approaches can be taken in estimating the typical value of this statistic.

(1) Design the model so that operating conditions are typical initially. Then simulate with this model and begin observing the statistic immediately.

(2) Run the model for such a long time that any untypical statistics gathered during the early part of the simulation are swamped out by the multitude of typical statistics gathered later in the simulation. That is, make the untypical observations such a small percentage of the total that their numeric influence is insignificant.

(3) Follow this sequence.

   (a) Run the model until typical operating conditions have come about.

   (b) Toss out statistical observations made up to that point, without otherwise changing the state of the model.

   (c) Continue the simulation, in the process gathering statistics which are not biased by the atypical observations accumulated previously.

The first approach requires of the analyst that he (a) knows what typical operating conditions are, and (b) makes special arrangements in the model to have these conditions be in effect at the outset of the simulation. In models of complex systems, the first requirement can rarely be met. Even if it can, it may be extremely troublesome to meet the second requirement. It is left as an exercise in the next section to show how this first approach can be taken with case study 2E.

The second approach suffers from the major disadvantage that long simulations may be required before steady state readings outweigh the early, biased observations. The cost of making long simulations with models of complex systems may be prohibitively high, making this approach undesirable.

The third approach, then, appears to be the one offering most hope. This approach can be put into perspective by comparing it to the steps involved in re-configuring a model at intermediate points during a run, so that a series of system alternatives can be simulated with a single submit. Here are those steps.

(1) Shut off the model.

(2) Re-define selected Blocks in the model, re-define Storage capacities, etc.

(3) "Clear" the model. Clearing the model involves the three things accomplished by use of a CLEAR Card, namely

   (a) Set statistics back to zero.
   (b) Empty the chains.
   (c) Perform the input phase.

(4) Re-start the model. Re-starting the model is accomplished by use of the START Card.

Relative to the above scheme, the third approach consists exactly of steps (1), (3a), and (4). That is, if steps (2), (3b), and (3c) in the above scheme are not performed, the effect is one of simply tossing out statistical observations, then continuing the simulation without otherwise having changed the status of the model. We already know how to perform steps (1) and (4). What is needed is a Control Card which will perform step (3a), i.e., will set statistics back to zero but, in contrast with the CLEAR Card, will not empty the chains and perform the input phase. The RESET Card is the GPSS Control Card which has this effect. It consists of the word RESET, entered in the Operation Field on a punchcard. There is no Block equivalent for it. When the Processor encounters a RESET Card, it sets model statistics back to zero and then, as usual, proceeds to the next card in the deck. If the next card is a START Card, the run is re-started, and accumulation of statistics during the next simulated time interval begins. With the RESET Card, then, GPSS provides the analyst with a convenient tool for gathering statistics under conditions of typical model operation.

There are several statistical items which the RESET Card does not cause to be restored to their initial values.

(1) "The random number stream" is not returned to its initial state. When the simulation is turned on again, the random number sequence begins at whatever point it had reached by the end of the preceding simulation.

(2) The Current Count at each Block is set equal to the number of Transactions which are at that Block. Total Block Counts are all set to zero. In the next set of Block counts produced, the Total Block Counts are then interpretable as the number of Transactions which entered the various Blocks during the most recent simulation.

(3) The Absolute Clock is not set back to zero, even though the Relative Clock is. The following interpretations can consequently be given to the values of the Relative and Absolute Clocks. The Absolute Clock measures the simulated time elapsed since the model was last cleared (or since it was first turned on if it has not been cleared). If no RESET Cards have been used, the Absolute and Relative Clock readings are identical, meaning the Relative Clock provides no additional information. If the readings are not identical, the Relative Clock measures the simulated time elapsed since the model was last reset.

At the user's option, a Selective RESET Card can be used. This makes it possible to specify certain entities (for example, certain Facilities, and/or Queues, and/or Storages) for which statistics are not to be set back to zero as part of the resetting operation. Because use of this option in fundamental GPSS modeling is rare, its details will not be discussed in this book.

It has been implied that, if a model is run long enough, typical operating conditions will come about. The question, of course, is how long must a run be for this to happen. There are no pat answers to this question. Consequently, a frequent practice

is to use the model itself, in experimental fashion, to estimate the duration of simulation required to reach steady state. The RESET Card can be conveniently used for this experimentation.

As an example of such RESET Card use, Figure 2.47 shows the extended source listing

```
,OCK                                                                    CARD
MBER  *LOC   OPERATION  A,B,C,D,E,F,G            COMMENTS              NUMBER
      *      SIMULATE                                                      1
      *                                                                    2
      *      DEFINE STORAGE CAPACITIES                                     3
      *                                                                    4
             STORAGE    S$MEN,4/S$NOWON,50    4 REPAIRMEN; UP TO 50 IN-PROD'N  5
      *                                                                    6
      *      MODEL SEGMENT 1                                               7
      *                                                                    8
             GENERATE   ,,,54         PROVIDE 54 MACHINES                 9
       BACK  ENTER      NOWON         MACHINE GOES INTO PRODUCTION        10
             ADVANCE    157,25        MACHINE IS RUNNING                  11
             LEAVE      NOWON         MACHINE FAILS;  COMES OUT OF PRODUCTION 12
             ENTER      MEN           CAPTURE A REPAIRMAN                 13
             ADVANCE    7,3           MACHINE IS BEING REPAIRED           14
             LEAVE      MEN           FREE THE REPAIRMAN                  15
             TRANSFER   ,BACK         GO BACK TO RUN AGAIN WHEN NEEDED    16
      *                                                                    17
      *      MODEL SEGMENT 2                                               18
      *                                                                    19
             GENERATE   40            TIMER ARRIVES EVERY 40 HOURS        20
             TERMINATE  1             SHUT OFF THE RUN                    21
      *                                                                    22
      *      CONTROL CARDS                                                 23
      *                                                                    24
             START      1             START RUN FOR WEEK 1                25
             RESET                     RESET FOR WEEK 2                   26
             START      1             START RUN FOR WEEK 2                27
             RESET                     RESET FOR WEEK 3                   28
             START      1             START RUN FOR WEEK 3                29
             RESET                     RESET FOR WEEK 4                   30
             START      1             START RUN FOR WEEK 4                31
               .                          .                               .
               .                          .                               .
               .                          .                               .
            RESET                     RESET FOR WEEK 24                  70
            START      1             START RUN FOR WEEK 24               71
            RESET                     RESET FOR WEEK 25                  72
            START      1             START RUN FOR WEEK 25               73
            END        RETURN CONTROL TO OPERATING SYSTEM                 74
```

Figure 2.47 A Model to Experimentally Estimate When Steady State is Reached

for a job which investigates behavior of the utilization statistic for the Storage NOWON in Case Study 2E, for a "4 hired, 4 rented" configuration. Cards 25, 26, and 27 show the typical "START-RESET-START" pattern corresponding to the respective steps of (1) shutting off the model, (3a) setting statistics back to zero, and (4) re-starting the model, as described above. The simulation time span between "resets" is one work week (i.e., 40 hours; note that 40 is the card 11 GENERATE Block A Operand). The number of "RESET-START" card pairs included in the model corresponds to a total simulation span of 25 work weeks. (Note that the "RESET-START" card pairs between cards 32 and 69 have been deleted in Figure 2.47).

Output produced from the Figure 2.47 model run is summarized in Table 2.22. Column 2 shows the utilization of the Storage NOWON experienced during the corresponding simulated week as indicated in Column 1. Column 3 shows the hand-computed Storage utilization on an accumulated basis, i.e., as though the model had been run the corresponding number of weeks without setting statistics back to zero at the end of each week. If the second

Week Spanned by Time Interval	Storage Utilization During the Interval	Storage Utilization Accumulated During the Entire Simulation
1	0.974	0.974
2	1.000	0.987
3	1.000	0.991
4	0.905	0.967
5	0.632	0.891
6	0.901	0.877
7	1.000	0.895
8	0.987	0.908
9	0.984	0.916
10	0.998	0.915
11	0.999	0.921
12	0.998	0.927
13	0.987	0.933
14	0.991	0.937
15	1.000	0.940
16	1.000	0.944
17	0.975	0.947
18	1.000	0.947
19	1.000	0.950
20	0.999	0.952
21	0.988	0.953
22	0.992	0.955
23	1.000	0.957
24	0.998	0.958
25	0.991	0.960

Table 2.22 Behavior of a Critical Statistic As the Case Study 2E Model Moves from Initial Conditions Toward Steady State Operation

approach described earlier for attempting to reach steady state were taken ("run the model long enough to swamp out the early, biased statistics"), observations would correspond to those in column 3. Inspection of column 2 suggests that typical operating conditions are reached after about 10 weeks of simulated time. As shown in column 3, the accumulated statistic is still far from having reached a typical value, even at the end of the 25th week. Hence, the realtive economy of the "reset approach" vs. the "swamping approach" is evident in Table 2.22.

Some comment on the entries in column 2 is in order. The lifetime distribution of each sewing machine is 157±25 hours. The smallest possible running time before failure is 132 hours. The model behaves as though all 50 machines in production have just been turned on at the beginning of the simulation. None of these 50 machines can fail, then, during the first 3 weeks. The utilization of the Storage NOWON should then be 1.0 at the end of each of the first 3 weeks. This is the case at the end of weeks 2

and 3, but not at the end of week 1. Why is there this aberration in the first week's statistic? It is because, as pointed out previously, the earliest Move Time in a model is time 1. The machines are consequently not turned on in the model until the end of the first hour (the Transactions are not moved into the ADVANCE Block, Block 3, until simulated time 1). But GPSS computes statistics as though the simulation starts at time 0. At time 40, then, the machines have only been on for 39 time units. This results in a utilization of 39 divided by 40, or 0.975. (The value 0.974 appears in the output due to truncation error.)

The utilization during week 5 is also of interest. Early in week 5, many of the machines turned on at the beginning of the first week fail (that is, they fail between time 160 and 200). There are not enough backup machines available to immediately take their place, and the 4 repairmen require time to work on the failed machines. The result is that the number of in-production machines drops dramatically, and the utilization figure falls to 0.632. It is only eventually, then, about by the end of week 10, that a consistent utilization pattern comes into effect.

The example just presented in detail illustrates use of the RESET Card to estimate when a model reaches steady state operation. The next logical usage mode for the RESET Card involves its use just after steady state has been reached, before the simulation is continued under the steady state conditions. In terms of Case Study 2E, for example, and conclusions reached from Table 2.22, the model would be reset after a 10-week simulation. But now another question arises. How long a run is required under steady state conditions for the measured statistics to be typical? Even within steady state, statistics fluctuate about their expected values. If the steady state run is too short, certain measurements may not be typical. A series of consecutive START Cards could be used after the RESET at steady state to experimentally estimate the steady state run required for pertinent statistics to stabilize.

Consider the effect of having consecutive START Cards in a job deck (without intermediate RESET or CLEAR Cards). Suppose that, having inputted a model and come to a START Card, the Processor has initiated a simulation. Eventually, the Termination Counter is decremented to zero, the simulation shuts off, and various output is produced. If the very next card the Processor encounters is another START Card, the simulation is simply re-initiated. That is, the Termination Counter is initialized with the next START Card's A Operand, and the Processor then goes into execution of the Clock Update Phase, etc. Of course, Transactions resident on the chains are exactly as they were at the time of the preceding shutoff. Furthermore, accumulated statistical information remains undisturbed, and is simply augmented by the continuing observations being made. In essence, then, inclusion of two or more consecutive START Cards provides the analyst with snapshots of accumulated system statistics as a simulation proceeds. By use of consecutive START Cards, "a" simulation is effectively composed of a series of consecutive simulations. At the end of each "next" simulation, system statistics are outputted, and the run then proceeds where it had left off.

Rather than performing such a START-Card experiment here, it will now simply be assumed that a 20-week simulation during steady state is satisfactory for the Case Study

2E model. Figure 2.48 shows the extended source listing for a job to study all combinations of 3, 4, and 5 repairmen, and 3, 4, and 5 rented machines, with the "RESET after

```
BLOCK                                                                    CARD
NUMBER   *LOC    OPERATION  A,B,C,D,E,F,G            COMMENTS            NUMBER
                 SIMULATE                                                    1
         *                                                                   2
         *       DEFINE STORAGE CAPACITIES                                   3
                 STORAGE    S$MEN,3/S$NOWON,50    3 MEN; MAX OF 50 MACH'S RUNNING  4
         *                                                                   5
         *       MODEL SEGMENT 1                                             6
         *                                                                   7
1         CNTRL  GENERATE   ,,,53       PROVIDE 53 MACHINES (50 OWNED; 3 RENTED)  8
2         BACK   ENTER      NOWON       MACHINE GOES INTO PRODUCTION        9
3                ADVANCE    157,25      MACHINE IS RUNNING                  10
4                LEAVE      NOWON       MACHINE FAILS;  COMES OUT OF PRODUCTION  11
5                ENTER      MEN         CAPTURE A REPAIRMAN                 12
6                ADVANCE    7,3         MACHINE IS BEING REPAIRED           13
7                LEAVE      MEN         FREE THE REPAIRMAN                  14
8                TRANSFER   ,BACK       GO BACK TO RUN AGAIN WHEN NEEDED    15
         *                                                                   16
         *       MODEL SEGMENT 2                                             17
         *                                                                   18
9                GENERATE   400         TIMER ARRIVES EVERY 10 WEEKS        19
10               TERMINATE  1           DECREMENT TERMINATION COUNTER       20
         *                                                                   21
         *       CONTROL CARDS, BLOCK RE-DEFINITIONS, AND STORAGE CAPACITY CHANGES  22
         *                                                                   23
                 START      1           START 1ST INITIALIZATION RUN        24
                 RESET                   RESET FOR 1ST STEADY-STATE RUN      25
                 START      2           START 1ST STEADY-STATE RUN          26
1         CNTRL  GENERATE   ,,,54       SET RENTED MACHINES = 4 FOR 2ND RUN  27
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
                 CLEAR                   CLEAR FOR 2ND INITIALIZATION RUN    28
                 START      1           START 2ND INITIALIZATION RUN        29
                 RESET                   RESET FOR 2ND STEADY-STATE RUN      30
                 START      2           START 2ND STEADY-STATE RUN          31
                   .                       .                                 .
                   .                       .                                 .
                   .                       .                                 .
1         CNTRL  GENERATE   ,,,55       SET RENTED MACHINES = 5 FOR 9TH RUN  64
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
                 CLEAR                   CLEAR FOR 9TH INITIALIZATION RUN    65
                 START      1           START 9TH INITIALIZATION RUN        66
                 RESET                   RESET FOR 9TH STEADY-STATE RUN      67
                 START      2           START 9TH STEADY-STATE RUN          68
                 END        RETURN CONTROL TO OPERATING SYSTEM               69
```

Figure 2.48 A Model Using RESET and CLEAR Cards to Investigate 9 Alternatives Under Steady State Conditions

reaching steady state" feature included. (Cards 32 through 63 have been deleted from the figure for simplicity.) Note that, after each RESET, the next START Card has 2 entered as the A Operand. Hence, two appearances of the timer-Transaction are required before any particular steady state run shuts off. This means that the steady state simulation interva is 20 weeks, not 10. Also note how, at the conclusion of each steady state run, the model is re-configured, then cleared (with the CLEAR Card), before the START-RESET-START pattern is then repeated for the next case.

The Storage utilizations resulting when the Figure 2.48 job was run are shown in Table 2.23(a). The average daily costs corresponding to these utilizations appear in Table 2.23(b). Note that the "4 hired, 4 rented" configuration looks even better now

		Machines Rented		
		3	4	5
Repairmen Hired	3	0.986	0.989	0.995
	4	0.989	0.995	0.995
	5	0.990	0.992	0.998

(a) Storage Utilizations

		Machines Rented		
		3	4	5
Repairmen Hired	3	292	298	280
	4	298	280	310
	5	320	334	316

(b) Total Costs, $/Day

Table 2.23 Pertinent Results Corresponding to the Figure 2.48 Model

than it did in Table 2E.3. Also note that each configuration studied involved only 30 weeks of simulated time. In contrast, 156 weeks were used for each configuration in Case Study 2E. The longer simulation was required in Case Study 2E to overcome the utilization statistic bias introduced during model startup. Of course, using the RESET Card to eliminate that bias is preferable to simply "simulating for a long time". The results are better defined, and execution time required for the run is substantially reduced. In this instance, the Figure 2.48 model took only about 25% as much computer time as the model in Case Study 2E.(2y)

In summary, in addition to introducing the RESET Card, this section brings out the fact that when a model has been built, the analyst's work "has just begun". The analyst's next task usually requires experimenting with the model to determine (a) how long to simulate to reach steady state, and (b) how long to simulate, given that steady state has been reached. Furthermore, these "how long" determinations will generally depend on the particular statistic or statistics of interest, and on the model configuration in question. Case Study 2E was simple because there was only one statistic of interest, a Storage utilization. In addition, it was implicitly assumed that the "how long" answers determined for a "4 hired, 4 rented" combination are valid for all other combinations investigated. Although that assumption is reasonably valid here, this is not true in general. Hence, the "how long" answers may have to be determined more than one time in a given problem.

It is beyond the scope of this book to dwell in further detail on the statistical aspects of simulation. It is similarly outside of the scope here to elaborate further on searching strategies which might be employed in the use of simulation models, as briefly discussed in the last section. These two important topics should be recognized, however, as a sine qua non of intelligent simulation.

(2y) On the University of Michigan's 360/67 multiprocessing system, CPU time for the Figure 2E.2 model was 73 seconds, whereas CPU time for the Figure 2.48 model was 18 seconds.

2.40. Exercises

2.40.1 Compare and contrast the effect of using the RESET and CLEAR Control Cards in GPSS modeling. Include a discussion of the differences, if any, in the effects of RESET and CLEAR on (a) Block Counts, (b) Relative and Absolute Clock values, and (c) the "random number stream".

2.40.2 (a) In Case Study 2D, compute the fixed costs, dollars per day, corresponding to the 3-hired, 3-rented configuration. Check your result against Table 2E.3(a).

(b) In Case Study 2D, compute the lost production cost, dollars per day, from data available from the 3-hired, 3-rented simulation. Check your results against Table 2E.3(b).

2.40.3 As discussed at the beginning of Section 2.39, initial model conditions in Case Study 2D vary markedly from those in effect when a state of steady operation has been reached. Show how to modify the model in that case study so that operating conditions are typical initially. In particular, at the beginning of the simulation, the lifetime of each of the 50 in-production machines should be determined by sampling from a "remaining lifetime distribution" hypothesized to be uniformly distributed over the interval from 0 to 182 hours. Any machines subsequently put into production should have their lifetime determined by sampling from the 157±25 distribution.

2.40.4 In a particular model, the implicit time unit is 1 minute. Each 480 time units, the Termination Counter is decremented by 1. Compare and contrast the nature of the statistics printed out for the alternative Control Card sequences shown below.

(a)
```
START     1
RESET
START     1
```

(b)
```
START     1
CLEAR
START     1
```

(c)
```
START     1
START     1
```

2.40.5 In a particular model, the implicit time unit is 1 minute. Each 240 time units, the Termination Counter is decremented by 1. Discuss the nature of the statistics printed out for the Control Card sequence shown in Figure P2.40.5. Include in your discussion a listing of the Relative and Absolute Clock values as they appear in the 5 sets of information which are printed out.

```
START     4
CLEAR
START     4
CLEAR
START     1
RESET
START     1
RESET
START     2
```

Figure P2.40.5

2.40.6 (a) Customers arrive at a two-man barber shop every 9±4 minutes. Each of the two barbers takes 16±4 minutes to provide service to a customer. Build a GPSS model to gather statistics for the line in which the customers wait. Use the model to simulate for one work day. Assume the barber shop opens at 9 a.m. and closes at 5 p.m. The barbers taken no breaks during the day.

(b) Modify the model for (a) to include the condition that the shop opens at 8:30 a.m. and closes at 5 p.m. Each barber takes a 30-minute lunch break at 12:00 noon, or as soon thereafter as possible (i.e., if a barber is at an intermediate point in working on a customer when it becomes noon, he finishes with the customer before taking the 30-minute lunch break). Customers who arrive at the shop during the lunch break wait for the barbers to return. What will be the long-run utilization of the barbers?

(c) In the model for (b), it is likely that the barbers' lunch breaks will overlap, i.e., that both barbers will be out at the same time for at least a part of their lunch break. Show how to modify the model so that the barbers take a staggered lunch break. That is, the second barber does not leave for his break until after the first barber has returned. Assume the second barber leaves "as soon as he can" after the first barber returns. If he is idle, he goes out immediately; if he is busy, he finishes working on his current customer, and then leaves. What will the influence of staggered lunch breaks be on the average customer waiting time?

2.40.7 In figure 2.48, the RESET Card is used to zero out statistics in the Case Study 2E model, before continuing the run under steady state conditions. Having reached steady state, a question then arises as to how much longer to run for the measured statistics of interest to be typical. The uncertainty arises because, even within steady state, statistics fluctuate about their expected values.

Show a deck for the Case Study 2E model for the "4 hired, 4 rented" configuration in which, after resetting after a 10-week simulation, aggregate statistics are printed out at the end of each next week for 20 consecutive weeks. Submit the deck for running. What is your conclusion

about the length of steady-state run required to gather meaningful steady-state information? Note that, to produce the results shown in Table 2.23, it was arbitrarily assumed that a 20-week steady state simulation for each configuration would be satisfactory.

2.40.8 In Table 2.22, the behavior of the utilization of the Storage NOWON in Case Study 2E is shown for a "4 hired, 4 rented" configuration. It was qualitatively concluded from the information in column 2 in that table that "eventually, about by the end of week 10, a consistent utilization pattern comes into effect". This conclusion for the "4 hired, 4 rented" configuration was then tacitly assumed to be valid for all configurations investigated in the Figure 2.48 job deck.

In practice, the length of run required to reach steady state may depend on the system configuration being investigated. Build and run a model to produce output similar to that shown in column 2 in Table 2.23, but for a "10 hired, 10 rented" configuration. Does the run have to be longer, shorter, or about the same as for the "4 hired, 4 rented" configuration before a consistent utilization pattern comes about?

2.40.9 In the discussion of program output in Case Study 2C, it was pointed out that "the duration of the simulation used in this case study is so small that any conclusions drawn from the model output must be tentative at best". Construct and carry out a plan to further investigate the two alternative queue disciplines proposed in that model. As part of your plan, make use of the RESET Card to estimate how long each of the two alternative models must be run for steady state conditions to be reached. Then simulate under steady state conditions with the objective of drawing firm conclusions about the relative merits of the two queue disciplines. Compare the resulting daily cost difference with the difference of about $141 tentatively established in Case Study 2C. As manager of the plant involved, which of the two disciplines would you have implemented at the tool crib?

2.40.10 In Case Study 2D (widget manufacture), a Transaction represented a worker cycling repeatedly through the assemble-fire-assemble-fire, etc., pattern. An alternative, yet equally valid, GPSS model can be built for the Case Study 2D problem by using a Storage to simulate the assemblers. In this alternative approach, a Transaction is thought of as representing the material required for a widget. Coming from the GENERATE Block as "raw material", the material is transformed by assembly and firing as the corresponding Transaction moves through the model, Block by Block.

Figure P2.40.10 shows the punchcard image of this alternative model. Study the model, then answer these questions.

(1) What is the inter-arrival time of Transactions at the main segment GENERATE Block? (Note that there are no Operands at the GENERATE Block.)

```
OPERATION     OPERANDS      COMMENTS
SIMULATE
STORAGE       S$GUYS,4
GENERATE                    CREATE RAW MATERIAL
ENTER         GUYS          ENGAGE AN ASSEMBLER
ADVANCE       30,5          ASSEMBLY TIME ELAPSES
SEIZE         OVEN          CAPTURE THE OVEN
ADVANCE       8,2           FIRING TIME ELAPSES
RELEASE       OVEN          RELEASE THE OVEN
LEAVE         GUYS          FREE THE ASSEMBLER
TERMINATE                   FINISHED WIDGET LEAVES THE SHOP
GENERATE      480           CREATE THE TIMER
TERMINATE     1             SHUT OFF THE RUN
START         1             START THE SIMULATION
RESET
START         5
STORAGE       S$GUYS,5
CLEAR
START         1
RESET
START         5
STORAGE       S$GUYS,6
CLEAR
START         1
RESET
START         5
END
```

Figure P2.40.10

(2) At what simulated time will a Transaction first attempt to move into the ENTER Block?

(3) For the first configuration studied, how many consecutive Transactions will move into the ENTER Block at simulated time 1, before a Transaction is finally denied entry there?

(4) What must happen in the model before the first Transaction not permitted to move into the ENTER Block is finally permitted to make that move? When will his successor then arrive at the GENERATE Block and attempt to move on in the model? What will the seccessor's initial experience be?

(5) Would the logic of the model be subverted if a QUEUE-DEPART Block pair were placed around the ENTER Block? Why or why not?

(6) Why, rather than having the LEAVE Block immediately follow the "ADVANCE 30,5", does it not appear in the model until after the RELEASE Block? What would be the implications, in terms of the system being modeled, of placing the LEAVE Block between the ADVANCE and SEIZE Blocks?

(7) How many different "configurations" will be simulated when the Figure P2.40.10 job is run? Explain.

(8) How long a simulation takes place with each configuration to eliminate the bias in statistics due to startup conditions?

(9) What is the duration of the steady state simulation for each configuration?

(10) Discuss the implications of removing the two CLEAR Cards from the model.

(11) Note that, as processing of the job proceeds, the capacity of the Storage GUYS is increased from configuration to configuration. What problem could result if the capacity were decreased from configuration to configuration? Could this problem arise if the CLEAR Cards were placed in the deck ahead of the cards re-defining the capacity of the Storage?

2.40.11 In Case Study 2D, it is implicitly assumed that all the widgets manufactured can be sold. The availability of only one oven as a manufacturing constraint means that, in the long run, not more than 60 widgets can be produced in an 8-hour day. (Because the firing time for each widget is 8±2 minutes, an oven utilization of 1.0 would correspond to a manufacturing rate of 60 widgets per working day.)

Suppose marketing studies indicate that demand for widgets will support a manufacturing rate of 275 widgets per working day. Additional ovens are available at the indicated utilization-independent cost of $10 per hour.

Build a model which reflects the availability of more than one oven. Then use the model to determine the number of assemblers and ovens which maximizes daily profit.

In the optimal "assembler, oven" configuration, it may not be true that all of the latent demand will be satisfied. Some demand necessarily goes unsatisfied if the optimal configuration produces widgets at a long-run average daily rate less than 275.

On the other hand, the optimal configuration may be capable of producing widgets at an average daily rate greater than 275. In this event, it is not necessary to produce widgets that cannot be sold. Assemblers may "stop working early" from time to time, to avoid wasting the $2 raw material cost in the manufacture of widgets that would go unsold. Note that it is not necessary for your model to have assemblers "stop working early" under certain conditions. If a given configuration produces, at full capacity, 283 widgets per day, for example, you can compute "profit-after-cost" as though only exactly 275 had been made. It must be assumed, of course, that even if assemblers are occasionally told by management to stop work early, they remain on the payroll for the full 8-hour day.

2.40.12 (a) Build a model to represent the queuing system shown in Figure P2.40.6. Arrivals occur at Station 1 every 120±30 seconds. The times required to perform service at Stations 1 and 2 are 40±10 and 110±25 seconds, respectively. Design the model to measure the waiting line behavior ahead of Stations 1 and 2. Assume that there is unlimited space

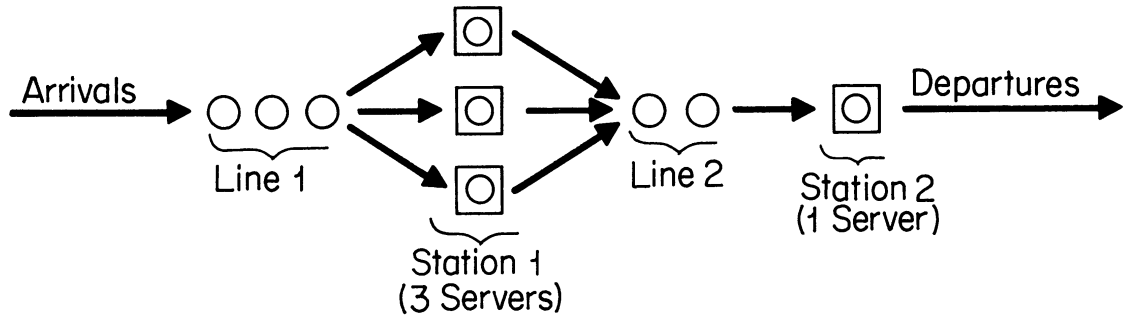


Figure P2.40.12

between the two stations, so that there is no maximum size beyond which waiting line 2 entries cannot be accommodated.

Experiment with the model to determine how long a simulation must be for the average Queue contents at each of the two waiting lines to stabilize. Then run the model in the steady state to measure what the steady state average Queue content is at the two points of waiting.

(b) Assume that, due to space limitations, the number of units in waiting line 2 cannot exceed 2. The Station 1 servers cannot begin service on the next unit until the preceding unit has been placed in waiting line 2. Modify your model in (a) to take this constraint into account. Then use the model to measure the average and maximum Queue content at waiting line 1 under steady state conditions. Compare the results with those obtained in (a).

(c) In part (b), suppose that the true utilization of the Station 1 servers is defined as the fraction of the time they spend performing service. They are not considered to be performing service when they are waiting to place the preceding unit in waiting line 2. Show how to modify part (b) so that it measures the true utilization of the Station 1 servers.

2.40.13 Ships of two types arrive at a harbor, where they unload their cargo. There are two tugs which service the harbor. Type 1 ships, which are small, require the use of only one of these tugs to dock and undock. Type 2 ships, which are larger, require the use of both tugs to dock and undock. Because of their size differences, the two ship types also unload at different berths, and have different unloading time requirements. Appropriate data appear in Table T2.40.9

(a) Build a model of the system, designing it to estimate the aggregate waiting time of each ship type at the harbor. Aggregate waiting time consists of time spent waiting for any reason, i.e., waiting for a berth, and/or waiting for a tug. A ship waiting to dock is not

	Ship Type	
	1	2
Inter-arrival Time, Minutes	130±30	390±60
Docking Time, Minutes	30±7	45±12
Number of Unloading Berths Available	6	3
Unloading Time, Hours	12±2	18±4
Undocking Time, Minutes	20±5	35±10

Table T2.40.9

to capture a tug until a berth is available. (What might happen if a ship could capture a tug before a berth became available for the ship?) Furthermore, a Type 2 ship is not to capture a tug until both tugs are available.

Use the model to estimate the requested aggregate waiting time statistics under conditions of steady state operation.

(b) If it costs $350 and $500 per hour to have ships of Type 1 and 2, respectively, wait, and the cost of providing a tug is $250 per day, can addition of a third tug at the harbor be justified?

(c) Assuming the waiting-ship cost stated in (b), at what estimated berth cost per day could addition of another berth for Type 1 ships be justified? At what daily cost could addition of another berth for Type 2 ships be justified?

2.40.14 There are seven booths at a toll point on a turnpike. Each open booth can handle a car in 15±3 seconds. Suppose that at 3 o'clock on a particular afternoon, there are four open booths and no cars waiting to move through the toll point. The mean hourly rate at which cars arrive at the toll point increases during the late afternoon, then decreases again early in the evening as shown in Table T2.40.10.

Time Span:	3:00-4:00	4:00-4:30	4:30-5:00	5:00-5:30	5:30-6:00	6:00-6:30	6:30-7:00
Arrival Rate, Cars/Hour:	1,000	1,250	1,600	2,000	1,800	1,300	1,000

Table T2.40.10

The variation in the inter-arrival times for each time span is plus or minus 25% of the mean, and is uniformly distributed.

To compensate for the traffic increase during the rush hours, a fifth booth is opened at 4:30, and the two remaining booths are opened at 5:00.

Model this situation, then run the model to estimate the maximum and average number of cars waiting at the toll point for each of the various time spans appearing in Table T2.40.10. Do you think it would be more realistic to model this situation with a one-line, multiple-server queuing system, or a multiple-line, multiple-server system? Which of these two queuing systems does your model correspond to?

2.41. Obtaining Printout During a Simulation

At the analyst's option, printouts of model statistics can be obtained during a simulation to supplement those produced at the end of a run. Complete sets of intermediate statistics are available through use of the C Operand on the START Card. Selected sets of statistics can be obtained by making use of the PRINT Block. Details are described below.

2.41.1 The Snap Interval Option

In addition to the Termination Counter, GPSS maintains a Snap Interval Counter. Its initial value is specified by the START Card's C Operand. When the Processor reads a START Card, it initializes the Termination Counter and Snap Interval Counter with the A and C Operand values, respectively. Each time the Termination Counter is decremented during the simulation, the Snap Interval Counter (SIC) is decremented by a like amount. When the SIC has been decremented to zero, the standard model output is produced. The SIC is then automatically re-initialized to the C-Operand value, and the simulation continues. When the SIC has again been decremented to zero, the standard model output is again outputted, the SIC is again re-initialized, etc. As a result, a series of "snapshots" of statistical information can be obtained as a simulation proceeds.

Figure 2.49 shows two examples of START Cards with the optional C Operand

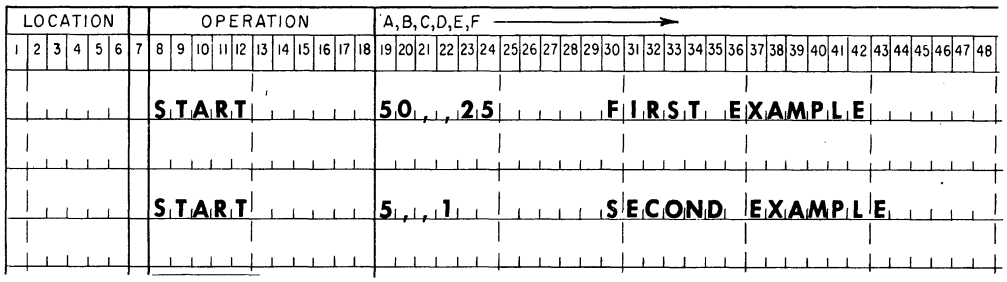


Figure 2.49 Examples of Snap Interval Use

used. In the first example, a complete set of output is produced after the Termination Counter (and Snap Interval Counter) has been decremented by 25. The SIC is then restored to 25 in value, and the simulation continues. When the Termination Counter (and Snap Interval Counter) has been decremented by another 25, the simulation shuts off and, as usual, a complete set of output is produced.

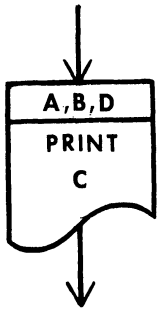
In the second example, a complete set of statistics is outputted each time the Termination Counter (and Snap Interval Counter) is decremented by 1. By the time the simulation shuts off, five sets of output (including the one at the end of the run) have been produced. Note that an identical effect could be

achieved by placing the card "START 1" in the model five consecutive times. The primary value of the Snap Interval Counter, then, is to cut down on the number of START Cards otherwise required to achieve the same effect.

2.41.2 The PRINT Block

Each time a Transaction moves through a PRINT Block, the GPSS Processor outputs the standard statistical information for a specified type of entity. For example, the usual statistical information for Facilities, or Storages, or Queues can be printed out. It is also possible to be selective with respect to the particular entity members for which the information is outputted. Instead of printing the statistics for all the Facilities in a model, for example, it can be specified that only the information for Facility 4 is to be outputted, or for Facilities 4, 5, and 6, and so on.

The PRINT Block and its various Operands are shown in Figure 2.50. The C



Operand	Significance	Default Value or Result
C	A mnemonic indicating the entity class of interent; F, S, and Q indicate Facilities, Storages, and Queues, respectively	Error
A and B	The smallest and largest numbers, respectively, of entity members for which information is to be outputted	Information will be printed for all the
D	Optional Operand; termed a Paging Indicator; if any alphabetic character is used as the D Operand, printing occurs without skipping to the top of a new page first	Printing occurs at the top of a new page

Figure 2.50 The PRINT Block and Its A, B, C, and D Operands

Operand indicates the entity class of interest. The A and B Operands specify the range of entity numbers within that class for which information is to be printed. If the analyst defaults on these two Operands, the Processor prints information for all the entity members. Printing normally occurs beginning at the top of a new page. If any alphabetic character is entered as the PRINT Block's D Operand, the skip to the top of the next page does not occur.

The punchcard images corresponding to several PRINT Blocks are shown in Figure 2.51. Each time a Transaction enters the Block whose image is shown in the first example, the current statistical information for Storages 4 through 8 will be printed. It is not considered an error if one or more of the Storages in this range do not exist.

LOCATION	OPERATION	A,B,C,D,E,F →
1 2 3 4 5 6	7 8 9 10 11 12 13 14 15 16 17 18	19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
	PRINT	4,,8,,S FIRST EXAMPLE
	PRINT	5,,5,,F,,D SECOND EXAMPLE
	PRINT	,,,Q THIRD EXAMPLE

Figure 2.51 Examples of the PRINT Block

Whenever a Transaction enters the second-example Block, the Processor prints out the current statistics for Facility 5 (i.e., Facilities in the range from "5 to 5, inclusive"). The printout takes place without skipping to the top of a new page.

A Transaction that enters the Block in the third example causes the Processor to output statistical information for all the Queues in a model. This is a result of defaulting on the A and B Operands. Note how commas are entered in card columns 19 and 20.

Figure 2.52 shows a complete Block Diagram segment which could be used to

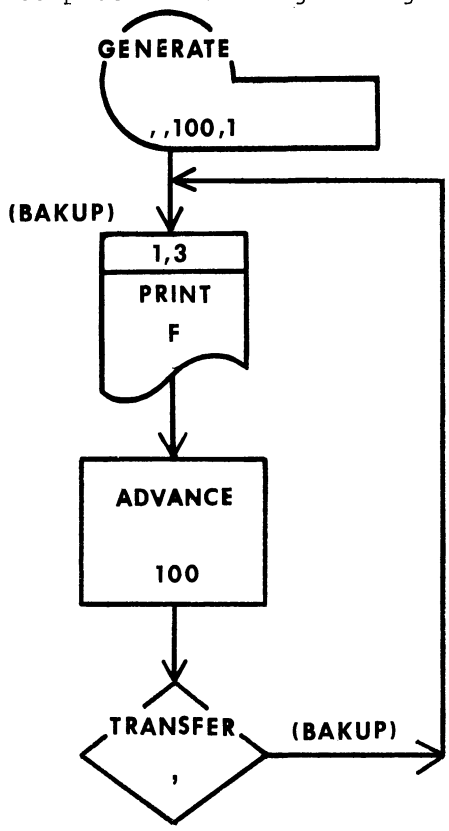


Figure 2.52 A Self-Contained Model Segment to Periodically Output Statistics for Selected Facilities

print out the statistics for Facilities 1, 2, and 3 in a model every 100 time units. A single Transaction circulates through the segment. The Transaction enters the segment through the GENERATE Block at time 100, and immediately prints out the Facility statistics. (It is understood that use of the Facilities is simulated in another part of the overall model.) It then goes into an ADVANCE Block for 100 time units. Exiting that Block at time 200, it unconditionally transfers back to the PRINT Block, prints out the Facility statistics again, and so on. Segments like this can be "appended" to models to provide selected statistical output as a simulation proceeds.

The PRINT Block can also be used to output the Relative and Absolute Clock readings, Block Counts, and the Current and Future Events Chains. The PRINT Block's C Operand (i.e., Field C mnemonic) for each of these possibilities is shown in Table 2.24. In each of these cases, no "range of entity members"

PRINT Block C Operand	Information Printed
C	Relative and Absolute Clock
N	Total Block Counts (All Blocks)
W	Current Block Counts (All Blocks)
B	Current and Total Block Counts (All Blocks)
MOV	Current Events Chain
FUT	Future Events Chain

Table 2.24 Partial List of Available PRINT Block C Operands

is involved. As a result, the A and B Operands are not used. A Transaction entering the Block "PRINT ,,MOV", for example, would cause the Current Events Chain to be printed out. As another example, the Processor will print out the Relative and Absolute Clock readings each time a Transaction enters the Block "PRINT ,,C".

Many of the additional GPSS entities not yet discussed can also be printed out with the PRINT Block. The C Operands which apply to these various possibilities are summarized in Appendix D. As further language entities are introduced, the way the PRINT Block can be used to output corresponding information will be mentioned.

The experienced user who makes use of the PRINT Block in a simulation may actually prefer suppressing the standard model output at the end of a run. This can be done by supplying NP (for No Printing) as the B Operand on the START Card. Use of this option is not especially recommended when one is first gaining practice with the GPSS language.

2.42 GPSS Control Cards: The JOB Card

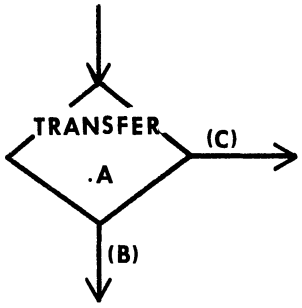
The analyst sometimes wants to simulate with two or more completely unrelated models in a single batch job. This can be done with use of the JOB Card. When the Processor finds a JOB Card in the deck, it completely destroys its internal image of

both the model and all associated simulations which went before the JOB Card. This means that all statistics are restored to their initial status, including "the random number stream"; all Transactions are returned to the latent pool; and all Block images are destroyed. The Processor then acts on the next model in the deck as though it were the first model, or the only model.

The format for the JOB Card is identical to that for the other Control Cards. The card consists simply of the word JOB, entered in the Operation Field.

2.43 Random Transfer of Transactions to Either One of Two Blocks

It is sometimes of interest to have the next Block entered by a Transaction be chosen at random from among two possibilities. This can be accomplished by using the TRANSFER Block in statistical transfer mode. Specifications for the Block as used in this mode are shown in Figure 2.53.



Operand	Significance	Default Value or Result
A	Fraction of the time that entering Transactions will transfer to the "C Block"	Error
B	A Block Location ("B Block")	Sequential Block
C	Another Block Location ("C Block")	Error

Figure 2.53 The TRANSFER Block in Statistical Transfer Mode

Note that, as shown within the TRANSFER Block itself in Figure 2.53, the first character in the A Operand is a decimal point. This means that, in the punchcard image of the Block, a decimal must be entered in card column 19. The rest of the A Operand is then interpreted as the decimal fraction of the time that Transactions entering the TRANSFER Block are to randomly move next to the Block Location specified with the C Operand (i.e., the "C Block"). The rest of the time, Transactions move next to the Block at the Location specified with the B Operand (the "B Block"). Not more than three digits can be used in specifying the A Operand.

As an example, consider Figure 2.54, where the punchcard image of a TRANSFER Block

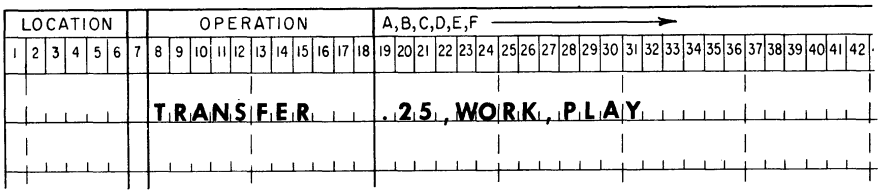


Figure 2.54 A First Example of the TRANSFER Block in Statistical Transfer Mode

operating in statistical transfer mode is shown. Transactions entering that Block will transfer about 25% of the time to the Block at the Location named PLAY (the "C Block"). The rest of the time (about 75%), they will transfer to the Block at the Location named WORK (the "B Block"). It is important to realize that each choice of a next Block is independent of the previous choices. For example, it is distinctly possible that 5 consecutive Transactions entering the Figure 2.54 TRANSFER Block all go to the Block at WORK as their next Block. It is only in the long run that about 75% of them will go to WORK next.

Figure 2.55 shows the punchcard image of another TRANSFER Block operating in

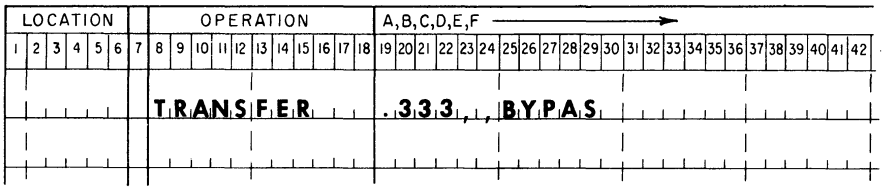


Figure 2.55 A Second Example of the TRANSFER Block in Statistical Transfer Mode

statistical transfer mode. No "B Block" has been specified. In such a case, the "B Block" is understood to be the sequential model Block. Transactions entering the Figure 2.55 Block will transfer, then, about 33% of the time to the Block at BYPAS. The rest of the time, they will move to the sequential Block, i.e., to the Block whose image would be on the punchcard following that shown in Figure 2.55. In this sense, Transactions entering the Figure 2.51 Block are sometimes said to "fall through" to the sequential Block.

In Figure 2.55, there would be nothing wrong in naming the Location occupied by the sequential Block, then supplying this Location Name as the B Operand at the TRANSFER Block itself. The B Operand default option simply spares the analyst the trouble of supplying a Location Name in this situation.

In Figure 2.53, note that parentheses have been used around the B and C Operands. These parentheses appear only in the Block Diagram of the model. They are not part of the B and C Operands, and do not appear in the punched-card image of the Block.

In statistical transfer mode, the TRANSFER Block cannot refuse entry to a Transaction. When a Transaction enters the Block, its choice of next Block is determined once and for all. It then tries to move into that chosen Block. If that Block denies entry, the Transaction is left on the Current Events Chain. At each next scan of the Current Events Chain (whether due to a scan re-start within the current Scan Phase, or to initiation of the next Scan Phase), the attempt to move the Transaction into the previously-chosen Block is again made. Eventually, the attempt will presumably be successful.

Use of the TRANSFER Block in statistical transfer mode is illustrated in Case Study 2F in the next section.

2.44 Case Study 2F: Inspection Station on a Production Line

(1) Statement of the Problem

Assembled television sets move through a series of testing stations in the final stage of their production. At the last of these stations, the vertical control setting on the sets is tested. If the setting is found to be functioning improperly, the offending set is routed to an adjustment station, where the setting is modified. After adjustment, the television set is sent back to the last inspection station, where the setting is again inspected. Television sets passing the final inspection phase, whether the first time or after one or more routings through the adjustment station, pass on to a packing area.

The situation described is pictured in Figure 2F.1, where "circles" represent

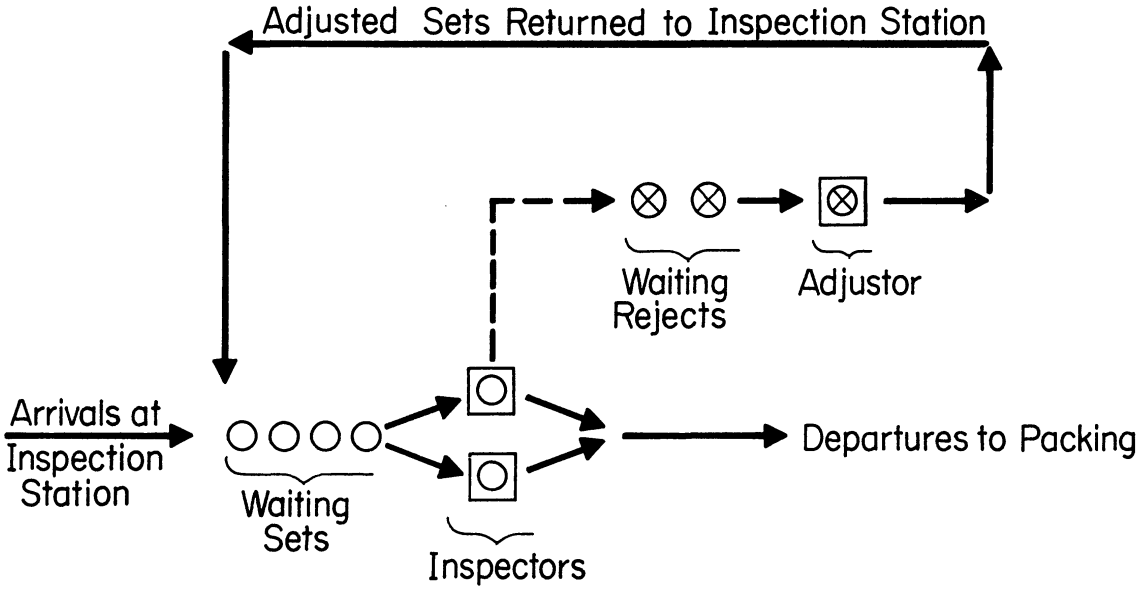


Figure 2F.1 Flow Schematic for Case Study 2F

television sets. "Open circles" are sets waiting for final inspection, whereas "circled x's" are sets whose vertical control setting is improper, and which are either being serviced at the adjustment station, or are waiting for service there.

Television sets arrive at the final inspection station from the previous station every 5.5 ± 2 minutes. Two inspectors work side-by-side at the final inspection station. The time required to inspect a set is 9 ± 3 minutes. About 85% of the sets pass inspection and continue on to the packing department. The other 15% are routed to the adjustment station, which is manned by a single worker. Adjustment of the vertical control setting requires 30 ± 10 minutes.

Design a GPSS model to simulate the operation of this segment of the production line. Use the model to estimate how much staging space must be provided ahead of the inspection station, and ahead of the adjustment station. Staging space is the space occupied by work (in this case, television sets) waiting for service to begin.

(2) Approach Taken in Building the Model

This model is easily constructed as a single sequence of Blocks. Television-set-Transactions move through the usual QUEUE-ENTER-DEPART-ADVANCE-LEAVE sequence simulating the inspection station. From the LEAVE, they enter a TRANSFER Block in statistical transfer mode. From here, some 85% "fall through" to a TERMINATE Block. The remaining 15% take the non-sequential exit to a QUEUE-SEIZE-DEPART-ADVANCE-RELEASE sequence simulating the adjustment station. After the RELEASE, they unconditionally transfer back to the QUEUE Block associated with the inspection station. The MAXIMUM CONTENT statistic for the two Queues can be interpreted directly as the staging space required ahead of the inspection and adjustment stations, respectively.

(3) Table of Definitions

Time Unit: 0.1 Minutes

GPSS Entity	Interpretation
Transactions	
Model Segment 1	Television sets
Model Segment 2	A timer
Facilities	
FIXER	Adjustment station worker
Queues	
AREA1	Inspection station waiting line
AREA2	Adjustment station waiting line
Storages	
TEST	Inspection station workers

Table 2F.1 Table of Definitions for Case Study 2F

(4) Block Diagram

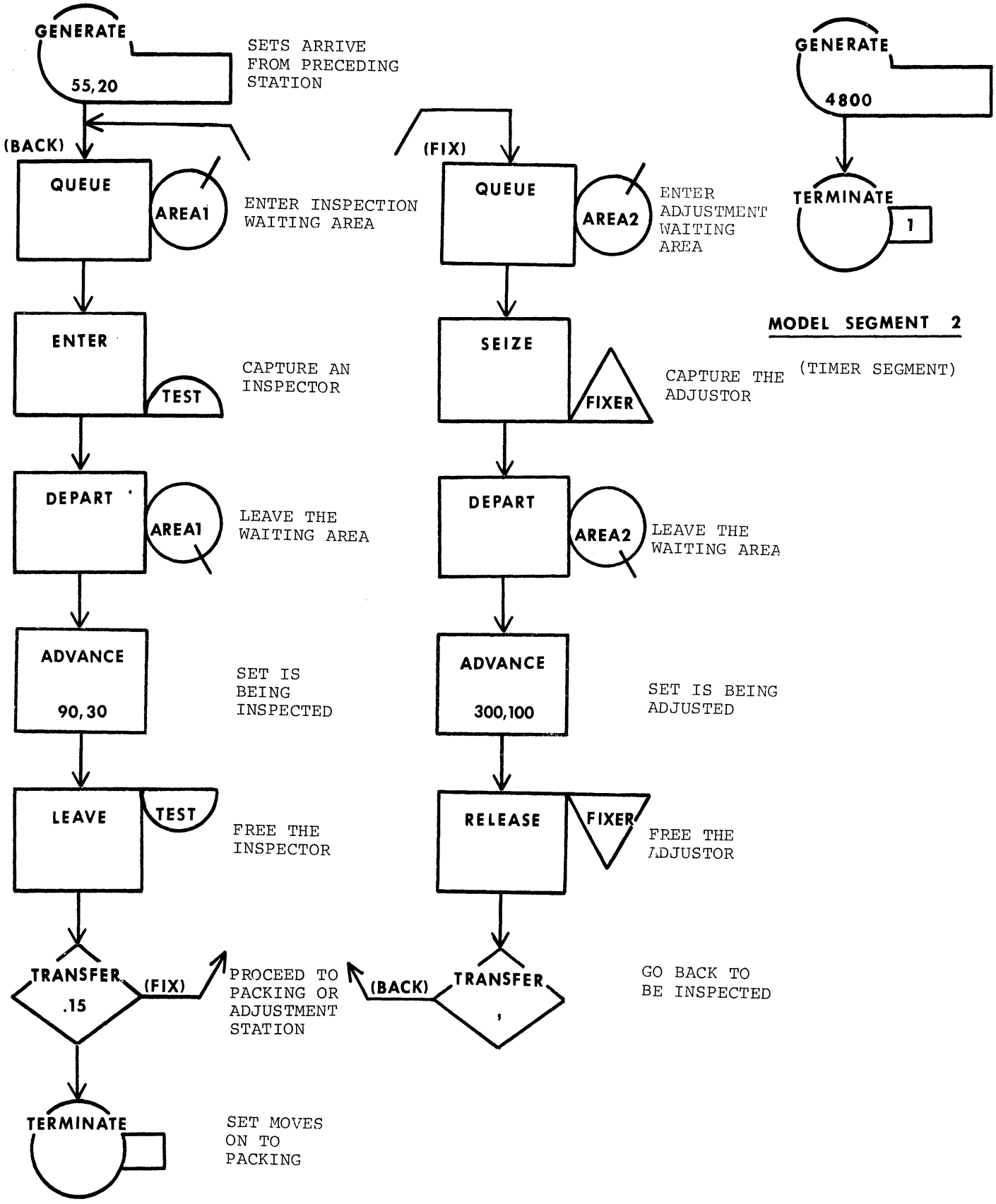


Figure 2F.2 Block Diagram for Case Study 2F

(5) Extended Program Listing

```
BLOCK                                                                    CARD
NUMBER   *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS               NUMBER
                SIMULATE                                                    1
         *                                                                  2
         *      DEFINE STORAGE CAPACITY                                     3
         *                                                                  4
                STORAGE    S$TEST,2    TWO WORKERS AT LAST INSPECTION STATION 5
         *                                                                  6
         *      MODEL SEGMENT 1                                             7
         *                                                                  8
1                GENERATE   55,20       SETS ARRIVE FROM PRECEDING STATION  9
2         BACK   QUEUE      AREA1       ENTER INSPECTION WAITING AREA      10
3                ENTER      TEST        CAPTURE AN INSPECTOR               11
4                DEPART     AREA1       LEAVE THE WAITING AREA             12
5                ADVANCE    90,30       SET IS BEING INSPECTED             13
6                LEAVE      TEST        FREE THE INSPECTOR                 14
7                TRANSFER   .15,,FIX    PROCEED TO PACKING OR ADJUSTMENT STATION 15
8                TERMINATE              SET MOVES ON TO PACKING            16
9         FIX    QUEUE      AREA2       ENTER ADJUSTMENT WAITING AREA      17
10               SEIZE      FIXER       CAPTURE THE ADJUSTOR               18
11               DEPART     AREA2       LEAVE THE WAITING AREA             19
12               ADVANCE    300,100     SET IS BEING ADJUSTED              20
13               RELEASE    FIXER       FREE THE ADJUSTOR                  21
14               TRANSFER   ,BACK       GO BACK TO BE INSPECTED            22
         *                                                                  23
         *      MODEL SEGMENT 2                                             24
         *                                                                  25
15               GENERATE   4800        TIMER ARRIVES AT END OF EACH DAY   26
16               TERMINATE  1           PROVIDE SNAP OUTPUT OR SHUT OFF THE RUN 27
         *                                                                  28
         *      CONTROL CARDS                                               29
         *                                                                  30
                START      5,,1        START THE RUN                      31
                END        RETURN CONTROL TO OPERATING SYSTEM             32
```

Figure 2F.3 Extended Program Listing for Case Study 2F

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
AREA1	3	.580	101	23	22.7	27.594	35.730		2
AREA2	5	2.152	17	2	11.7	607.764	688.799		1

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(a) Queue Statistics After 1 Day

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
AREA1	3	.610	198	46	23.2	29.580	38.532		
AREA2	5	1.400	28	5	17.8	480.035	584.391		

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(b) Queue Statistics After 2 Days

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
AREA1	3	.560	296	85	28.7	27.253	38.232		1
AREA2	5	1.566	42	6	14.2	537.238	626.777		

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(c) Queue Statistics After 3 Days

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
AREA1	3	.586	393	111	28.2	28.664	39.946		
AREA2	5	1.263	53	11	20.7	457.867	577.785		1

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(d) Queue Statistics After 4 Days

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
AREA1	3	.546	493	147	29.8	26.618	37.927		
AREA2	5	1.434	70	11	15.7	491.828	583.525		2

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(e) Queue Statistics After 5 Days

Figure 2F.4 Selected Program Output for Case Study 2F

(7) Discussion

Model Implementation

The model was run for 5 consecutive 8-hour working days, with aggregate statistics measured from the start of the simulation being printed out at the end of each day. In the extended program listing in Figure 2F.3, note how this effect is accomplished with the timer-Transaction GENERATE Block (card 26) and the START Card (card 31). Through the A and C Operands on the START Card, the Termination Counter and Snap Interval Counter are initialized with values of 5 and 1, respectively. A timer-Transaction enters the model every 480 simulated minutes and immediately terminates, decrementing both of these counters by 1. For the first four timer-Transactions, this causes the Processor to (a) print out the standard model statistics, because the Snap Interval Counter has been decremented to zero, (b) restore the Snap Interval Counter's value to 1, and (c) continue the simulation, because the Termination Counter is not yet zero. Finally, when the fifth timer-Transaction enters the model and terminates, the simulation shuts off and the standard printout is produced for the fifth and last time.

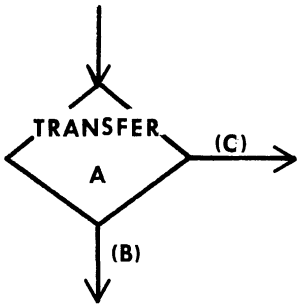
Program Output

Referring to Figure 2F.4(e), the maximum number of television sets in the staging areas ahead of the inspection station (AREA1 Queue) and adjustment station (AREA2 Queue) did not exceed 3 and 5, respectively, for the 5 simulated days. These maximum contents had already been realized by the end of day 1 [Figure 2F.4(a)], suggesting that these statistics reached stable values during the simulation. In contrast, note the variation in the values of AVERAGE TIME/TRANS for the Queue AREA2 throughout the simulation .

On day 1, 17.2% of the television sets going through inspection were routed to the adjustment station [as shown in Figure 2F.4(a), of 101 TOTAL ENTRIES to the AREA1 Queue, 99 went through inspection and 2 are still in the Queue; of the 99 going through inspection, 17 were sent to the AREA2 Queue]. During the 5 days, Figure 2F.4(e) information reveals that 14.2% of the inspections resulted in sets being sent on for adjustment.

2.45 Conditional Transfer of Transactions to One of Two Blocks

When the TRANSFER Block is used in statistical transfer mode, a Transaction's choice of next Block is independent of conditions currently in effect there. If the next Block chosen denies entry, the Transaction simply waits until permission to enter is granted. In an alternative mode, the TRANSFER Block can be used to send a Transaction to whichever one of two Blocks will first accept it. When applied this way, the TRANSFER Block is said to be used in BOTH mode. Specifications for this mode are shown in Figure 2.56.



Operand	Significance	Default Value or Result
A	Literally, the word BOTH	Error
B	A Block Location ("B Block")	Sequential Block
C	Another Block Location ("C Block")	Error

Figure 2.56 The TRANSFER Block in BOTH Mode

The A Operand is literally the word BOTH. The B and C Operands are the names of two Block Locations in the model. A Transaction entering the TRANSFER Block in BOTH mode immediately attempts to move into the "B Block" (i.e., the Block at the Location whose name is supplied as the B Operand). If entry is denied, it then attempts to move into the "C Block". If that Block also denies entry, the Transaction is left on the Current Events Chain. At each next scan of the Current Events Chain, the Processor again attempts to move the Transaction into the B Block and, if entry is still denied there, into the C Block. Eventually, then, the Transaction moves into whichever one of the two Blocks will first accept it. Note that the B Block is tested first, then (if necessary) the C Block is tested. When both Blocks are simultaneously willing to accept the Transaction, then, it is the B Block which the Transaction enters.

As is true for the TRANSFER Block in statistical transfer mode, the analyst can default on the B Operand at his option. When this is done, it is assumed that the B Block is the sequential Block in the model.

As an example of BOTH mode use, assume that there are only three chairs in the waiting area of a barber shop. Customers arrive at the shop every 14±5 minutes, but are willing to wait only if there is a chair for them to sit in. Otherwise, they leave and do not return later. Letting the Storage SEATS represent the seating capacity in the shop, Figure 2.57 shows a Block Diagram for a model to simulate the shop's operation. Customers entering the model first check to see if a seat is available in the

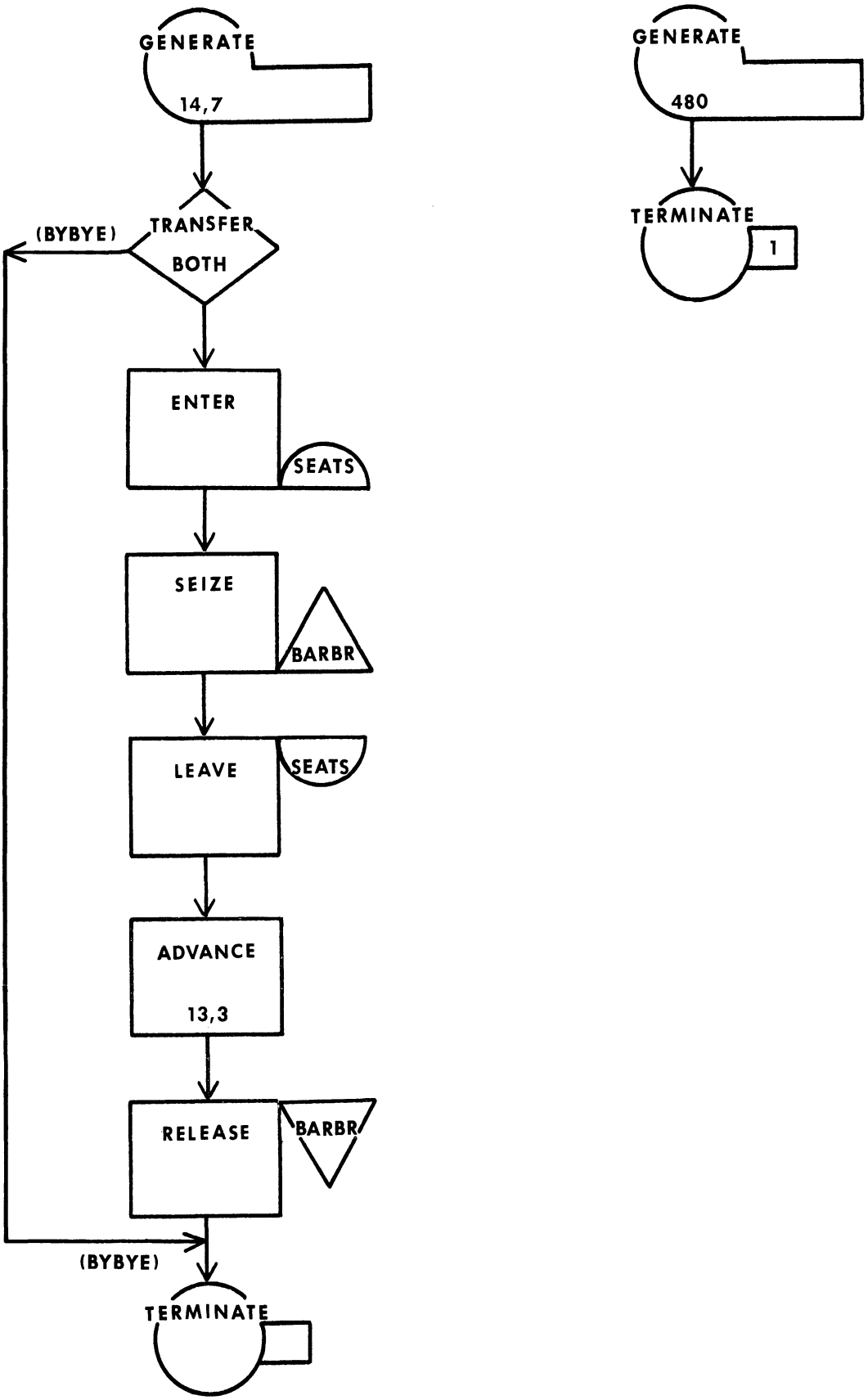


Figure 2.57 An Example of TRANSFER Block BOTH Mode Use

waiting area. If it is, they stay; otherwise, they leave. Note that, in this example, the TRANSFER Block's "C Block" can never deny entry. Also note that no Location Name has been given to the ENTER Block. This is a situation in which it is convenient to default on the TRANSFER Block's B Operand.

2.46. Exercises

2.46.1 In Exercise 2.40.7, Case Study 2E as to be run for a "4 hired, 4 rented" configuration with resetting after a 10-week simulation, followed by collection of aggregate statistics at the end of each week for the next 20 weeks. Show how to do this by taking advantage of the C Operand on the START Card, as contrasted with using a series of consecutive START Cards in the model.

2.46.2 Show the Operands for PRINT Blocks which, when executed, will cause these printouts to occur.

(a) Current Events Chain
(b) Future Events Chain
(c) Information for all the Storages in a model
(d) Information for Facilities 3 through 7
(e) Information for Queue 5

2.46.3 (a) The Current Events Chain is to be printed out every 50 simulated time units, beginning at time 50. Show a self-contained Block Diagram segment which will have this effect.

(b) Repeat (a), only assume that the printouts are to begin at time 25.

2.46.4 In Case Study 2F, the staging space required ahead of the adjustment station depends not only on the average service time at the inspection and adjustment stations, but also on the amount of variation possible in this average.

(a) Re-run the Case Study 2F model so that inspection and adjustment always require exactly 9 and 30 minutes, respectively. What is the most staging space required under these circumstances during a 5-day period?

(b) Now re-run Case Study 2F under the assumption that inspection time is 9±3 minutes, as before, whereas adjustment of the vertical control setting is now 30±20 minutes. What is the most staging space required under these circumstances during a 5-day period?

2.46.5 Show how to modify the model for Case Study 2F so that not more than 4 television sets can simultaneously wait for service at the adjustment station. A worker at the inspection station cannot begin the next inspection until the preceding unit has been sent on to packing or has been transferred to the staging space ahead of the adjustment station. Run the resulting model and compare the behavior of the inspection station's waiting line with the behavior observed in Case Study 2F.

2.46.6 In Case Study 2F, it is unrealistically assumed that, even after it has gone through the adjustment station at least one time, the probability a television set will require further adjustment is still 0.15. Show how to modify the model in that case study under the assumption that after a visit to the adjustment station, the probability is only 0.03 that a set will have to be adjusted again.

Remaining exercises in this section all pertain to the Figure 2.57 model. Except when indicated, the questions are not otherwise related to each other.

2.46.7 No explicit Queue is included in Figure 2.57. Show how to include such a Queue to gather statistics on the waiting line which forms ahead of the barber.

2.46.8 Discuss the possibility of interpreting the model-produced statistics for the Storage SEATS as applying to the waiting line which forms ahead of the barber. In light of this discussion, what additional information, if any, would be acquired by using an explicit Queue in the model as suggested in 2.46.7?

2.46.9 Describe how output from the Figure 2.57 model can be used to determine how many customers "left without waiting" during the course of the simulation.

2.46.10 Show how to modify the model so there are <u>two</u> barbers who work in the shop. Service time for the first barber is still 13±3 minutes. Service time for the second barber is 15±4 minutes. Change the customer inter-arrival time to 7±3 minutes, but retain the condition that customers who cannot find a chair to sit in immediately leave the shop. (Hint: Use two Facilities "in parallel" to simulate the two barbers.)

2.46.11 For the conditions described in 2.43.10, discuss the possibility of using a Storage with a capacity of 2 to simulate the two barbers.

2.46.12 Expand the Figure 2.57 model to include a shoeshine boy. After their haircut is finished, 25% of the customers <u>consider the possibility</u> of getting a shoeshine. Among them, only 20% are <u>willing to wait</u> for a shoeshine; if the boy is busy at the time, the others who consider getting a shine do not bother to wait.

2.46.13 Describe how output from the model in 2.43.12 can be interpreted to determine how many customers considered getting a shoeshine during the simulation, but went without the shine because the boy was busy at the time.

2.46.14 Expand the Figure 2.57 model to include the condition that 40% of the customers leaving the shop because no chair is available come back after 15±5 minutes to try again. Those who are not successful on the second try do not return.

2.46.15 Modify the Figure 2.57 model to introduce this variation on customer behavior. About 20% of the customers arriving at the shop remain <u>only if</u> the barber is immediately available. The others remain as long as they can at least get a seat in the waiting area.

CHAPTER 3

Sampling from Probability Distributions in GPSS

3.1. Introduction

In Chapter 2, ten GPSS Blocks were introduced and for one of them, the TRANSFER, four usage modes were described. A variety of models can be built with these ten Blocks, as the Chapter 2 cases and exercises indicate. A source of discontent with the Chapter 2 models, however, is that all inter-arrival and service times are assumed to be uniformly distributed. Making this assumption, the beginning analyst can concentrate on relatively more important aspects of model-building. Enough is now known about GPSS modeling, though, so that the next topic of interest is the representation and use of non-uniform distributions in the language.

The analyst must use Functions to describe non-uniform distributions. Fortunately, this does not mean the modeling concepts developed in Chapter 2 must be discarded. After modest modification, all models built in that chapter are still valid when non-uniform inter-arrival and service times are in effect. The modification involves two steps.

(1) Functions describing each of the pertinent non-uniform distributions must be defined.

(2) At the various GENERATE and ADVANCE Blocks, the A and/or B Operands describing uniform distributions must be replaced with references to the appropriate Functions.

When these two steps have been performed, the analyst has converted previously "unrealistic" models to ones which simulate systems under considerably more realistic conditions. This chapter discusses how these steps are accomplished.

3.2. The GPSS Uniform Random Number Generators

In general, the "source" of randomness in a simulation model is one or more functions which, when called, return a value drawn at random from the population uniformly distributed over the 0-1 interval. Recall how this was the case in Chapter 1, where a uniform random number function was introduced and then used to sample from non-uniform distributions, as well as from more general uniform populations. Drawing a value from a distribution, then, generally involves two steps.

(1) First, a value is drawn from the population uniformly distributed on the 0-1 interval.

(2) Then, this result is converted, or mapped, into an equivalent value from the population of interest.

GPSS also uses values drawn at random from the uniform 0-1 population to sample from non-uniform distributions, as well as from more general uniform populations. There are eight distinct sources of uniform random numbers in GPSS. These random number sources are a pre-defined part of the Processor itself. Each source can be thought of as a Function. The pre-defined names of these random number Functions are RN1, RN2, RN3, RN4, RN5, RN6, RN7, and RN8.

Each of the eight GPSS random number generators returns six-digit values drawn from random number streams which are identical to each other. This fact is illustrated in Table 3.1, where the values of RNj, j = 1, 2, 3, ···, 8, are shown for the first ten calls on each of the random number functions.[3a] The numbers themselves in Table 3.1

Number of the Call	Function Called: RN1	RN2	RN3	RN4	RN5	RN6	RN7	RN8
1	.000573	.000573	.000573	.000573	.000573	.000573	.000573	.000573
2	.510675	.510675	.510675	.510675	.510675	.510675	.510675	.510675
3	.870337	.870337	.870337	.870337	.870337	.870337	.870337	.870337
4	.999177	.999177	.999177	.999177	.999177	.999177	.999177	.999177
5	.778871	.778871	.778871	.778871	.778871	.778871	.778871	.778871
6	.194160	.194160	.194160	.194160	.194160	.194160	.194160	.194160
7	.790719	.790719	.790719	.790719	.790719	.790719	.790719	.790719
8	.014667	.014667	.014667	.014667	.014667	.014667	.014667	.014667
9	.043340	.043340	.043340	.043340	.043340	.043340	.043340	.043340
10	.645420	.645420	.645420	.645420	.645420	.645420	.645420	.645420

Table 3.1 Values Returned by the Various Random Number Functions in GPSS When the Functions are Called the First Ten Times

are not of consequence. What is of interest is that entries across any particular row are identical. Hence, the eighth call on RN7 produces the same random value as does, say, the eighth call on RN2. Similarly, the third value returned by RN8 is identical to the third value returned by RN3, and so on.

As suggested in Table 3.1, the uniform random numbers in GPSS are six digit values. The smallest possible value is 0.000000. The largest is 0.999999. Hence, $0 \leqslant RNj < 1$, for $j = 1, 2, 3, \cdots, 8$.

When the analyst defines a probability distribution, he must specify which of the eight GPSS random number sources is to be used as the starting point (argument) when a sample is drawn from that distribution. Each time model conditions require sampling from the analyst-defined distribution the Processor first goes to the indicated random number source, drawing the next value from the corresponding sequence of random numbers. The analyst-supplied description of the distribution of interest is then used to convert this "uniform" value into an equivalent value from that distribution.

3.3. GPSS Sampling from Uniform Distributions at GENERATE and ADVANCE Blocks

In Chapter 2, it was implied that the GPSS Processor "knows how" to sample from uniform distributions. Each time a Chapter 2 model was run, it was frequently necessary to sample from uniform distributions with means and spreads specified by the A and B Operands at GENERATE and ADVANCE Blocks. To draw such samples, the Processor must have access to a source of random numbers uniformly distributed over the 0-1 interval. The Processor uses RN1 as this source.[3b]

[3a] The entries in Table 3.1 were produced by Version I, Modification Level 2, of GPSS/360 Because of occasional changes in the random number generation scheme, the values may vary from implementation to implementation. Exercise 5.7.10 involves building a GPSS model which will call and display the first 10 values produced by the RNj, for j taking on values from 1 to 8.

[3b] The Processor also needs a random number each time it executes the TRANSFER Block in statistical transfer mode. It uses RN1 in this context, too.

Consider how the Processor uses RN1 to sample from uniform distributions. Figure 3.1 shows the extended program listing for the tool crib model previously shown in

```
BLOCK                                                                  CARD
NUMBER   *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS             NUMBER
                SIMULATE                                                 1
         *                                                               2
         *      MODEL SEGMENT 1                                          3
         *                                                               4
1                GENERATE   420,360,,,1        CATEGORY 1 MECHANICS ARRIVE  5
2                QUEUE      LINE               ENTER "CATEGORY 1 SEGMENT" OF LINE  6
3                SEIZE      CLERK              CAPTURE THE CLERK            7
4                DEPART     LINE               LEAVE THE LINE               8
5                ADVANCE    300,90             USE THE CLERK                9
6                RELEASE    CLERK              FREE THE CLERK              10
7                TERMINATE                      LEAVE THE TOOL CRIB AREA    11
         *                                                              12
         *      MODEL SEGMENT 2                                         13
         *                                                              14
8                GENERATE   360,240,,,2        CATEGORY 2 MECHANICS ARRIVE 15
9                QUEUE      LINE               ENTER "CATEGORY 2 SEGMENT" OF LINE  16
10               SEIZE      CLERK              CAPTURE THE CLERK           17
11               DEPART     LINE               LEAVE THE LINE              18
12               ADVANCE    100,30             USE THE CLERK               19
13               RELEASE    CLERK              FREE THE CLERK              20
14               TERMINATE                      LEAVE THE TOOL CRIB AREA    21
         *                                                              22
         *      MODEL SEGMENT 3                                         23
         *                                                              24
15               GENERATE   28800              TIMER ARRIVES AFTER 8 HOURS 25
16               TERMINATE  1                  SHUT OFF THE RUN            26
         *                                                              27
         *      CONTROL CARDS                                           28
         *                                                              29
                START      1                  START THE RUN               30
                END        RETURN CONTROL TO OPERATING SYSTEM          31
```

Figure 3.1 A Repetition of Figure 2C.4

Figure 2C.4. Suppose the Figure 3.1 model is being run. During the input phase, the Processor first reads the "GENERATE 420,360" card. A sample must then be drawn from the 420±360 distribution to pre-schedule arrival of the first Transaction at this Block. The Processor draws the sample by first calling RN1, with 0.000573 returned as the value, per Table 3.1.(3c) This value is then converted to 60.(3d) Transaction 1 is fetched from the latent pool and hooked onto the Future Events Chain, scheduled to move to Block

(3c) When RN1 is called, two 32-bit numbers are multiplied to form a 64-bit product. A subset of these 64 binary digits is then used as the next "random number." Consecutive calls on RN1 produce the sequence of random numbers shown in Table 3.1, unless the initial multiplier value is modified with use of the control card discussed in Section 3.18. As it happens, when RN1 is called to support the GENERATE and ADVANCE Block operations explained in Section 3.3, the subset of the 64 bit product used by the Processor to form the needed "random number" differs from the subset used when RN1 is called in any other context. This results in values differing from those shown in Table 3.1. For simplicity, however, the Section 3.3 computations assume that the RN1 values in Table 3.1 are those used for the "internal operations" being described. It is the procedure explained in Section 3.3 which is important, not the numbers.

(3d) The value drawn from the uniform distribution A±B is computed as the integer portion of A-B+RN1(2B+1). In this case, the value is 420-360+0.000573(720+1), or 60.4+. The integer portion is simply 60.

1 at time 60. Later in the input phase, the Processor reads the "GENERATE 360,240" card. To pre-schedule the first arrival at that Block, RN1 is called and returns a value of 0.510675, per Table 3.1. (Note that this is the Processor's second call on RN1.) This value is then converted to 365. Transaction 2 is fetched from the latent pool and hooked onto the Future Events Chain, scheduled to move to Block 8 at time 365. Finally, at Block 15, the Processor reads the "Generate 28800" card. Because the inter-arrival time at this GENERATE Block is deterministic, no random number is required. Without calling RN1, the Processor immediately fetches Transaction 3 from the latent pool and hooks it onto the Future Events Chain in proper fashion.

Then, with the input phase complete, the first Clock Update Phase begins. The clock is advanced to 60, and Transaction 1 is moved to the Current Events Chain. The scan of the CEC begins, and Transaction 1 is moved to and from its GENERATE Block. In pre-scheduling the next arrival at that GENERATE Block, the Processor again calls RN1, with 0.870337 returned as the value. (This is the third call on RN1 so far.) This value is then converted to 687. Transaction 4 is fetched from the latent pool and hooked onto the Future Events Chain accordingly. The movement of Transaction 1 is resumed, and it flows to the ADVANCE Block (Block 5). Sampling from the 300±90 distribution is consequently required. The Processor calls RN1, which returns a value of 0.999177. (This is the fourth call on RN1.) This value is then converted to 390. The 390 is added to the clock reading (60), producing 450. Transaction 1 is then hooked onto the Future Events Chain, scheduled to move to Block 6 at time 450. And so on.

Two conclusions are to be drawn from the steps traced out above.

(1) One and the same random number source is being referenced from multiple points in a model. In the example, RN1 is being used at two GENERATE Blocks, and at two ADVANCE Blocks. Each Block uses only a subset of the values drawn from the random number stream. The subset used depends on the chronological order in which the various Block subroutines are executed as Transactions move into those Blocks.

(2) Suppose the analyst specifies RN1 as the source of random numbers for sampling from some distribution he defines. In doing so, he may be sharing RN1 with the Processor. This will be true if the Processor is forced to do its own sampling from uniform distributions at one or more GENERATE and/or ADVANCE Blocks, or if it requires random numbers for the TRANSFER Block used in statistical transfer model.

3.4. The GPSS Method for Sampling from Discrete, Non-Uniform Distributions

The analyst usually wants the Processor to sample from non-uniform distributions to determine inter-arrival times, and service times. Some scheme must then be followed to convert a draw from the uniform 0-1 population into a draw from the non-uniform distribution of interest. The scheme followed to sample from discrete distributions will now be described.

A random variable is said to be discrete when it can take on only a finite number of different values. For example, when a die is tossed, the random variable defined as the number of points turned up can take on only the values 1, 2, 3, 4, 5, or 6. Or, if a paycheck is known to be more than $75 but less than $80, the random

variable defined as the amount of the paycheck can take on only the values 75.01, 75.02, 75.03, ···, 79.98, and 79.99. In the first example, the random variable is discrete and integer-valued. In the second example, the random variable is also discrete, even though it is not integer-valued.

When the GPSS Processor samples from discrete distributions, it draws a random number from a uniform 0-1 source, then uses this value to perform a table-lookup on the cumulative distribution for the population of interest. Assume, for example, that a random variable can take on the values 2, 5, 8, 9, and 12, with corresponding relative frequencies of 0.15, 0.20, 0.25, 0.22, and 0.18. This information is shown in Table 3.2,

Value of Random Variable	Relative Frequency	Cumulative Frequency	Portion of 0-1 Range Spanned	Interval
2	0.15	0.15	0.0 to 0.15	1
5	0.20	0.35	0.15+, to 0.35	2
8	0.25	0.60	0.35+, to 0.60	3
9	0.22	0.82	0.60+, to 0.82	4
12	0.18	1.00	0.82+, to 1.0	5

Table 3.2 An Example of a Discrete, Integer-Valued Random Variable

where the values of the random variable and their relative frequencies appear in the first and second columns, respectively. The third column shows the corresponding cumulative distribution, which spans the range of values from 0 to 1. The interval within the 0-1 range corresponding to each value of the random variable is shown in column 4. For example, for a random variable value of 2, the range of cumulative values from 0.0, up to and including 0.15, is spanned. This is designated as Interval 1 in column 5. For a random variable value of 5, the applicable cumulative values range from anything greater than 0.15 (i.e., 0.15+) up to and including 0.35. This range is designated as Interval 2 in column 5. And so on.

Now suppose that a sample is to be drawn from the Table 3.2 distribution. First, a value is drawn from a uniform 0-1 source. Assume that the value is 0.523664. Then, by table-lookup, it is established that 0.523664 falls into Interval 3. The indicated value assumed by the random variable is consequently the third value or, in this case, 8 (third entry, column 1).

Given appropriate information about a distribution, the GPSS Processor automatically performs the table-lookup procedure when a sample must be drawn from the distribution. The information the Processor must have includes (a) the source of uniform 0-1 random numbers, (b) the values which the random variable can assume, and (c) their cumulative frequency of occurrence. The analyst supplies this information by defining a discrete GPSS Function. Definition of discrete GPSS Functions is described in the next section.

3.5. Defining Discrete GPSS Functions

The user must supply this information to define a Discrete GPSS Function:[3e]

(1) A name must be given to the Function. The naming convention used for Facilities, Queues, and Storages also applies to Functions. Names can be numeric or symbolic. When numeric,

[3e] The normal quantities of Functions which the user can define at the 64, 128, and 256K levels are 20, 50, and 200, respectively.

they must be positive, whole numbers. When symbolic, names are composed of from 3 to 5 alphanumeric characters, with the restriction that the first 3 be alphabetic. Examples of valid and invalid names are shown in Table 3.3

Valid	Invalid
JOE	-5
7	HI
JOE23	MP2LO
SWICH	8BETA

Table 3.3 Examples of Valid and Invalid Names for Functions

(2) The Function's argument must be specified. The argument names the source of random numbers to be used in sampling from the distribution described by the Function. The argument will consequently be RNj, where j is 1, 2, 3, 4, 5, 6, 7, or 8. The choice of the particular random number source is up to the user.

(3) The number of different values that can be assumed by the discretely-distributed random variable must be specified.

(4) The values themselves, and their corresponding cumulative frequencies, must be provided.

The format in which the first three pieces of the above-indicated information is supplied on a punchcard is shown in Table 3.4. The Function's name is entered in the Location Field, and the word FUNCTION is punched in the Operation Field. The A

Punchcard Field	Information Supplied in Field
Location	Name (numeric or symbolic) of the Function
Operation	Literally, the word FUNCTION
Operands	
A	RNj, where j = 1, 2, 3, 4, 5, 6, 7, or 8
B	Dn, where n is the number of different values the random variable can assume

Table 3.4 Specifications for First Card in Function Definition

Operand indicates the source of 0-1 random numbers to be used in evaluating the Function. The B Operand consists of the character D (for discrete), and an integer indicating how many different values the random variable can assume.

Then, the random variable values, and their corresponding cumulative frequencies, are provided on one or more subsequent punchcards, termed Function follower cards. The format used for these cards is shown in Figure 3.2.[3f] The "basic unit" of information on a Function follwer card is X_i, Y_i, where X_i is the i-th cumulative probability value, and Y_i is the corresponding value of the random variable. The first and second entries in each "basic unit" are separated by a comma. Consecutive basic units are separated by a slash (/). The basic units must be ordered so that the cumulative frequencies form a strictly ascending sequence. Basic units must be punched in consecutive card columns, beginning in column 1 and not extending beyond column 72. If necessary or convenient,

---

[3f] An alternative format for the X_i, Y_i pairs is to punch them in consecutive 6-column fields. When this approach is followed, no "commas" or "slashes" are used. The "comma-slash" procedure is convenient, however, and is the only one followed in this book.

Card Columns	Information Entered
1 forward, but not beyond 72	$X_1,Y_1/X_2,Y_2/X_3,Y_3/ \ldots /X_i,Y_i/ \ldots /X_n,Y_n$

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

$$X_1 \quad X_2 \quad X_3 \quad \ldots \quad X_i \quad \ldots \quad X_n$$

Figure 3.2 Format for Function Follower Cards

two or more Function follower cards can be used. During the input phase, the first blank column encountered on a Function follower card causes the Processor to ignore the rest of the card. If a pair has not yet been found for each random variable value, it is expected that additional pairs will be found on the next card, starting in column 1. Any basic unit begun on a card must be completed on that same card. The examples which follow will help clarify additional questions about formatting these cards.

A GPSS Function for sampling from the distribution described in Table 3.2 is shown in Figure 3.3. The Function has been symbolically named KATHY. RN4 has been chosen

```
LOCATION  | OPERATION | A,B,C,D,E,F ------>
KATHY     | FUNCTION  | RN4,D5
.15,2/    | .35,5/.6,8/.82,9/1,12
```

Figure 3.3 Example of a Function for Sampling from the Table 3.2 Distribution

as the source of uniform 0-1 random numbers. The random variable can take on 5 different values. The cumulative frequencies, and the corresponding five random variable values, are entered as five matched pairs on the next card. The cumulative values are in ascending order.

When presenting pairs of points describing a distribution, it is optional whether decimal points are punched when the data are whole numbers. For example, the first pair in Figure 3.3 could have been punched as ".15,2.". Or, the last pair could have been punched as "1.,12", or as "1.,12.".

The number of pairs placed on each card is optional, as long as the pairs do not extend beyond card column 72. Figure 3.4 shows an alternative way in which the Figure 3.3

```
LOCATION  | OPERATION | A,B,C,D,E,F ------>
KATHY     | FUNCTION  | RN4,D5
.15,2
.35,5/.6,8
.82,9/1,12
```

Figure 3.4 An Alternative Format for the Figure 3.3 Example

example could have been punched. It is usually convenient, of course, to put as many pairs as possible on each card.

Note that the last pair on a card is not followed by a slash, nor is the first

pair on the next card followed by a slash. The slash is required only to separate consecutive pairs entered on the same card.

Figure 3.5 provides a graphical interpretation for the Function defined in Figure 3.3 (or in Figure 3.4). The Function consists of a series of horizontal "steps".

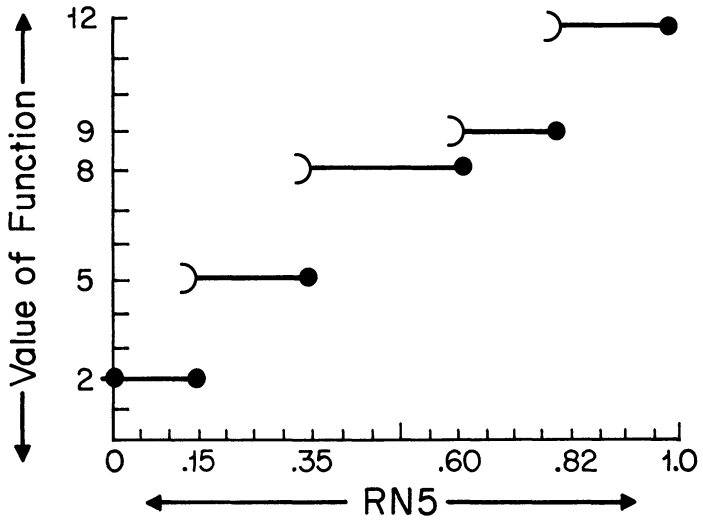


Figure 3.5 Graphical Interpretation for the Figure 3.3 Function

The closed circle at the right of each step indicates that the corresponding argument value is included in the step. For example, the second step covers values up to and including 0.35. The open semi-circle at the left of each step indicates that the corresponding argument value is excluded from the step. For example, the second step starts not with 0.15, but with values greater than 0.15. The 0.15 is included as part of the first step.

3.6. Using Discrete Functions at GENERATE and ADVANCE Blocks

Suppose that the distribution of inter-arrival times at a particular GENERATE Block in a model is non-uniform. The analyst arranges for Transaction arrivals at that GENERATE Block by taking these two steps.

(1) He defines a Function describing the applicable inter-arrival time distribution.

(2) He uses the Function as the A Operand at the GENERATE Block and uses zero, either explicitly or by default, as the B Operand.

Then, when model conditions call for computing the next inter-arrival time at that GENERATE Block, the Processor determines the value of the A Operand by evaluating the indicated Function. That value is then used directly as the next inter-arrival time. This is just as though the analyst were specifying a uniform distribution at the GENERATE Block, with a mean equal to the value returned by the Function, and a spread of zero around that mean. With the spread of zero, the Function's value is used "deterministically" as the inter-arrival time. But because the Function's value is

itself determined on a random basis, the effective inter-arrival times are also randomly distributed. The distribution they follow is, of course, the one defined by the Function.

The way a Function is referenced via a Block Operand depends on whether the Function's name is numeric or symbolic. When the name is numeric, the Function is referenced as FNj, where j is the number of the Function. When the name is symbolic, the Function is referenced as FN$sn, where sn is the Function's symbolic name. For example, Function 16 would be referenced as FN16. On the other hand, the Function symbolically named KATHY would be referenced as FN$KATHY. Note that a $(dollar sign) is interposed between FN and the symbolic name itself.

An example of Function use at a GENERATE Block will now be given. Suppose that Table 3.5 shows the time between consecutive arrivals of orders at a distribution center. A Transaction is to be used to represent an order, and a GENERATE Block is to

Time Between Orders, Hours	Relative Frequency
2	0.10
3	0.30
4	0.40
5	0.20

Table 3.5 An Example of a Discretely-Distributed Inter-Arrival Time Random Variable

be used to arrange for arrival of these orders. First, the information in Table 3.5 is used to construct a corresponding Function. Table 3.5 is repeated and extended in Table 3.6, where cumulative frequency values are shown in column 3. Figure 3.6 shows

Time Between Orders, Hours	Relative Frequency	Cumulative Frequency
2	0.10	0.10
3	0.30	0.40
4	0.40	0.80
5	0.20	1.0

Table 3.6 The Cumulative Distribution for the Table 3.5 Random Variable

a Function defined from the Table 3.6 information. The Function has been named TBO, for

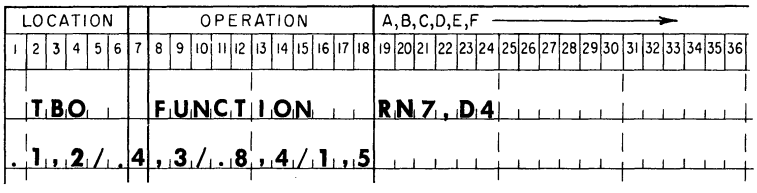


Figure 3.6 A Function for Sampling from the Table 3.6 Distribution

"time between orders". The seventh random number generator has been chosen, arbitrarily,

as the Function's argument. Figure 3.7 shows the GENERATE Block through which Transactions representing orders come into the model. The Function TBO has been used as the

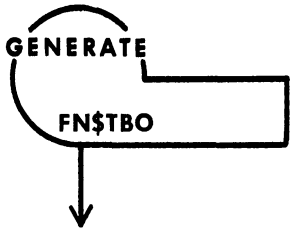


Figure 3.7 Use of the Figure 3.6 Function at a GENERATE Block

A Operand. The default has been taken on the B Operand. The inter-arrival time distribution followed at the Block, then, is the one in Table 3.5.

The procedure just described for Function use at GENERATE Blocks is equally valid for Function use at ADVANCE Blocks. For example, assume the service time at a work station in a production system follows the distribution shown in columns 1 and 2 in Table 3.7. The corresponding cumulative distribution appears in column 3 of the table. Figure

Service Time, Minutes	Relative Frequency	Cumulative Frequency
5	.05	.05
6	.12	.17
7	.28	.45
8	.30	.75
9	.18	.93
10	.07	1.0

Table 3.7 An Example of a Discretely-Distributed Service Time Random Variable

3.8 shows a Function defined from the Table 3.7 information. The Function has been named

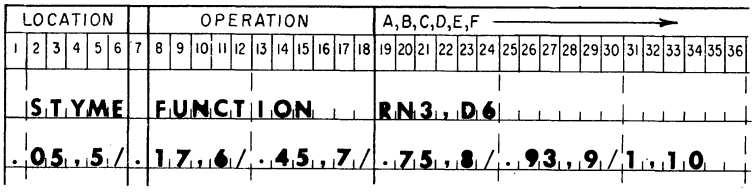


Figure 3.8 A Function for Sampling from the Table 3.7 Distribution

STYME, for "service time". Figure 3.9 shows the ADVANCE Block at which Transactions representing demands for service at the work station are delayed in the model, simulating the time that elapses while service is performed. Whenever a Transaction moves into the ADVANCE Block, the Function STYME is evaluated. A value is drawn from RN3, a table-lookup is performed with information given in the Function definition, and a particular value for the Function is determined. This value is then used, "plus or minus zero", as the holding time at the ADVANCE Block.

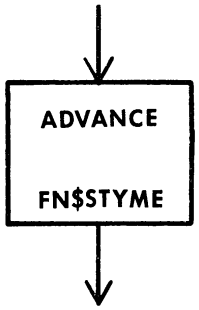


Figure 3.9 Use of the Figure 3.8 Function at an ADVANCE Block

Whenever user-defined Functions are used at GENERATE Blocks, the Function definitions must be placed ahead of the GENERATE Block images in the card deck. The reason for this is straightforward. During the input phase, when the Processor encounters the image of a GENERATE Block, it immediately samples from the indicated inter-arrival time distribution while performing the pre-scheduling task. If the cards defining a referenced Function have not yet appeared in the deck, an error condition is assumed to be in effect. This is the case even though the definition appears later in the deck. As a consequence, the practice in this book is to place all Function definitions ahead of the first Block image. Note that this is not necessary, strictly speaking, for Functions used at ADVANCE Blocks. Those Functions will not be needed by the Processor until sometime after the input phase has been completed. By that time, the Function definitions will have been encountered, as long as they appear somewhere before the START Card.

3.7. Case Study 3A: A Second Tour Through Case Study 2D

(1) Statement of the Problem

Case Study 2D involves a system to manufacture widgets. Assemblers working in the system move repeatedly through this cycle.

(1) Assemble next widget.
(2) Wait, first-come, first-served, to use the oven.
(3) Use the oven.
(4) Return to (1).

In Case Study 2D, the times required to assemble a widget, and use the oven, are uniformly distributed over the interval 30±5 and 8±2 minutes, respectively. Suppose now that, instead of being uniformly distributed, the random variables "assembly time" and "oven-use time" follow the distributions shown in Tables 3A.1 and 3A.2, respectively.

Assembly Time, Minutes:	25	26	27	28	29	30	31	32	33	34	35
Relative Frequency:	.01	.03	.05	.10	.18	.26	.18	.10	.05	.03	.01

Table 3A.1 Distribution of Assembly Times

Oven Use Time, Minutes:	6	7	8	9	10
Relative Frequency:	.05	.25	.40	.25	.05

Table 3A.2 Distribution of Oven-Use Times

Note that these distributions have (arbitrarily) been chosen to be symmetric, and are centered about 30 and 8, respectively. As a result, the average assembly and oven-use times are identical to those in Case Study 2D. It is evident upon inspection, however, that the standard deviation of the distributions in Tables 3A.1 and 3A.2 is less than for the previously-used uniform distributions.

Modify the model in Case Study 2D to take the changes in the time-distributions into account. Then use the model to simulate for a 40-hour work week, assuming there are no discontinuities within a working day, or in moving between consecutive 8-hour work days. Do this for the cases of 4, 5, and 6 assemblers. Compare and contrast the corresponding system behavior, and the number of assemblers for whom the simulated week's profit is a maximum, with the results found in Case Study 2D.

(2) Approach Taken in Building the Model

The inherent logic of the model does not differ from that in Case Study 2D. The required modification of the 2D model only involves defining Functions to represent the distributions in Tables 3A.1 and 3A.2, and replacing the "ADVANCE 30,5" and "ADVANCE 8,2" Block Operands with A Operands referencing the corresponding Functions, and B Operands of zero (either explicitly, or by default). Table 3A.3 repeats the information appearing in Tables 3A.1 and 3A.2 and shows the cumulative probabilities for the distributions.

Assembly Time, Minutes	Relative Frequency	Cumulative Frequency
25	.01	.01
26	.03	.04
27	.05	.09
28	.10	.19
29	.18	.37
30	.26	.63
31	.18	.81
32	.10	.91
33	.05	.96
34	.03	.99
35	.01	1.0

(a) Assembly Time

Oven-Use Time, Minutes	Relative Frequency	Cumulative Frequency
6	.05	.05
7	.25	.30
8	.40	.70
9	.25	.95
10	.05	1.0

(b) Oven-Use Time

Table 3A.3 Cumulative Probabilities for Assembly Time and Oven-Use Time

Figure 3A.1 shows the images of the punchcards with which these distributions are defined through the Functions symbolically names ASSEM and FIRE, respectively. When the Figure 3A.1 punchcards have been placed in the Case Study 2D deck, and "ADVANCE 30,5" and "ADVANCE 8,2" have been replaced with "ADVANCE FN$ASSEM" and "ADVANCE FN$FIRE", respectively, the modifications are complete.

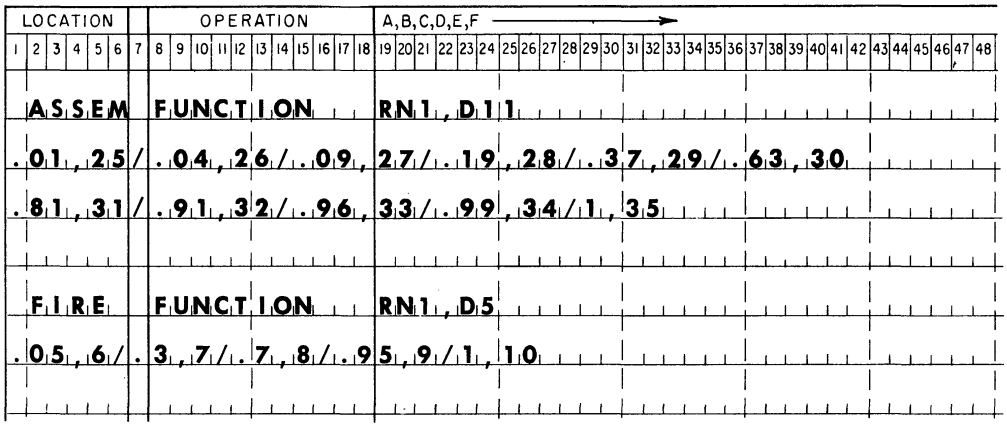


Figure 3A.1 Function Definitions for the Table 3A.1 and 3A.2 Distributions

(3) Table of Definitions

Time Unit: 1 Minute

GPSS Entity	Interpretation
Transaction	
Model Segment 1	Assemblers
Model Segment 2	The timer
Facilities	
OVEN	The oven
Functions	
ASSEM	Assembly time distribution
FIRE	Oven-use time distribution

Table 3A.4 Table of Definitions for Case Study 3A

(4) Block Diagram

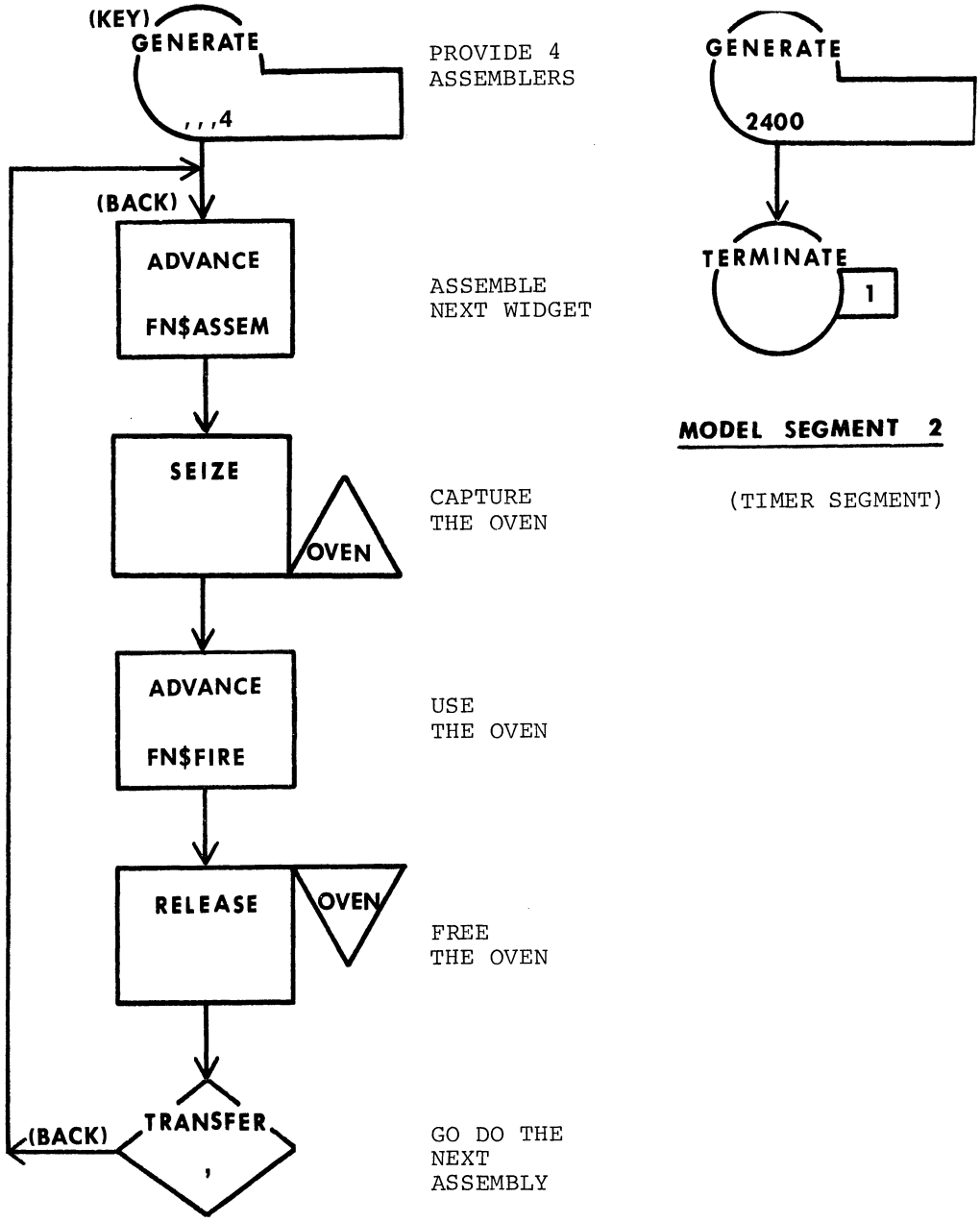


Figure 3A.2 Block Diagram for Case Study 3A

(5) Extended Program Listing

```
BLOCK                                                                CARD
NUMBER  *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS            NUMBER
               SIMULATE                                                  1
        *                                                                2
        *      DEFINE FUNCTIONS                                          3
        *                                                                4
         ASSEM FUNCTION   RN1,D11      ASSEMBLY-TIME DISTRIBUTION       5
        .01,25/.04,26/.09,27/.19,28/.37,29/.63,30                        6
        .81,31/.91,32/.96,33/.99,34/1,35                                 7
         FIRE  FUNCTION   RN1,D5       FIRING-TIME DISTRIBUTION         8
        .05,6/.3,7/.7,8/.95,9/1,10                                       9
        *                                                               10
        *      MODEL SEGMENT 1                                          11
        *                                                               12
1        KEY   GENERATE   ,,,4         PROVIDE 4 ASSEMBLERS             13
2        BACK  ADVANCE    FN$ASSEM     ASSEMBLE NEXT WIDGET             14
3              SEIZE      OVEN         CAPTURE THE OVEN                 15
4              ADVANCE    FN$FIRE      USE THE OVEN                     16
5              RELEASE    OVEN         FREE THE OVEN                    17
6              TRANSFER   ,BACK        GO DO THE NEXT ASSEMBLY          18
        *                                                               19
        *      MODEL SEGMENT 2                                          20
        *                                                               21
7              GENERATE   2400         TIMER ARRIVES AFTERS 5 DAYS      22
8              TERMINATE  1            SHUT OFF THE RUN                 23
        *                                                               24
        *      CONTROL CARDS                                            25
        *                                                               26
               START      1            START THE 1ST RUN                27
1        KEY   GENERATE   ,,,5         RE-CONFIGURE FOR 2ND RUN         28
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
               CLEAR                    CLEAR FOR 2ND RUN               29
               START      1            START THE 2ND RUN                30
1        KEY   GENERATE   ,,,6         RE-CONFIGURE FOR 3RD RUN         31
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
               CLEAR                    CLEAR FOR 3RD RUN               32
               START      1            START THE 3RD RUN                33
               END        RETURN CONTROL TO OPERATING SYSTEM             34
```

Figure 3A.2 Extended Program Listing for Case Study 3A

(6) Program Output

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
OVEN	.812	243	8.028	4	

(a) Oven Statistics When 4 Assemblers Are Used

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
OVEN	.959	291	7.914	6	

(b) Oven Statistics When 5 Assemblers Are Used

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
OVEN	.988	297	7.989	6	

(c) Oven Statistics When 6 Assemblers Are Used

Figure 3A.3 Selected Program Output for Case Study 3A

(7) Discussion

Model Implementation

As shown in Figure 3A.2, the three alternative assemblers-hired configurations were studied in a single computer run.

Program Output

The oven statistics in Figure 3A.3 show that 242, 290, and 296 widgets were finished during the 40-hour week for the cases of 4, 5, and 6 assemblers, respectively. (This "finished widget" count excludes the widget still in the oven at the end of the simulation.) The 242, 290, and 296 figures can be compared to corresponding values of 235, 279, and 294, respectively, in Case Study 2D (Figure 2D.3). The somewhat higher rate of production resulting from the changed distributions is to be expected. The smaller standard deviations associated with assembly and oven-use times lead to relatively better synchronization between the assembly and firing steps. If there were no variation in these times, the system could be made to perform in perfectly synchronized fashion. For example, assume for the moment that assembly and oven-use time are exactly 30 and 8 minutes, respectively. With 4 assemblers, it is then possible to synchronize oven use so that all waiting is entirely eliminated. After an assembler finishes with the oven, it will be another 30 minutes before he wants it again. Meantime, the other three assemblers can each use the oven for 8 minutes, meaning it will be available when the first assembler again needs it. Under such ideal circumstances, an assembler can produce about 12.6 widgets per day [480/(30+8) = 12.6+]. Four assemblers can produce about 252 widgets in a 40-hour work week. Because of randomness in the system, then, about 10 to 20 fewer widgets are produced with 4 assemblers each week than would be the case in a system in which all randomness had been eliminated.

Despite the changes in the time distributions, the optimal number of assemblers is still 5. Return-after-costs for 4, 5, and 6 assemblers is $210, $300, and $180, respectively. The return-after-costs for 5 assemblers in Case Study 2D was $245. Hence, the change in time distributions has resulted in more than a 20% increase in profit.

3.8. Exercises

3.8.1 (a) What are the names of the eight uniform random number generators in GPSS?

(b) When RN7 is called for the fifth time in a simulation, what value does it return?

(c) In which three different contexts may the GPSS Processor require a source of uniform random numbers for its own purposes? Which of the eight random number generators does the Processor use on such occasions?

(d) Describe how the GPSS Processor samples from uniform distributions specified by using A and B Operands at GENERATE and ADVANCE Blocks.

(e) Assume that the Processor is sampling from the 200±100 distribution at a GENERATE Block. Four calls have already been made on RN1. What will the integer portion of the sample value be?

(f) When the Processor samples from the uniform distribution A±B, the sample value is taken to be the integer portion of A-B+RN1(2B+1). Why is only the integer portion used, rather than the entire value?

(g) What is a discrete random variable? Give several examples.

(h) Critically discuss the statement, "A discrete random variable can only assume integer values."

(i) Describe how the GPSS Processor samples from discrete, non-uniform distributions.

(j) What range of values can the RNj take on when used as Function arguments? What is the smallest value possible? The largest value?

(k) What ordering property is imposed on the first members of pairs of points used in defining Functions to sample from probability distributions?

(el) Construct a graphical interpretation of the Function defined in Figure 3.6.

(m) Why must the definition of a Function appear in the card deck before the card image of a GENERATE Block which references that Function?

(n) Show a GENERATE Block at which Function 3 is used as the A Operand. Default on the B Operand.

(oh) Show a GENERATE Block at which the Function SYMBL is used as the A Operand. Default on the B Operand.

(p) Show an ADVANCE Block at which the Function JOE23 is used as the A Operand. Default on the B Operand.

(q) For the ADVANCE Block "ADVANCE FN$CPU", why is the value returned by the Function CPU used without modification as the holding time?

(r) When the Function TBO in Figure 3.7 is called the first time, what value will it return? Assume that the random number sequence shown in Table 3.1 is in effect. What other assumption must be made before an unconditional answer can be given to the question?

(s) When the Function STYME in Figure 3.8 is called the sixth time, what value will it return? Assume that the random number sequence shown in Table 3.1 is in effect. Is your answer conditioned on another assumption as well?

3.8.2 Define a discrete GPSS Function which can be used to sample from the population whose values are -3, 0, 5, 9, and 14, with probabilities 0.1, 0.05, 0.15, 0.30, and 0.40, respectively. The Function should be given the symbolic name DRAW, and should use RN7 as its argument. Could this Function meaningfully describe an inter-arrival time distribution, or a service time distribution? Why or why not?

3.8.3 The time between consecutive arrivals of ships at a harbor follows the distribution shown in Table T3.8.3.

Inter-Arrival Time, Hours	Relative Frequency
2	.06
3	.09
4	.17
5	.28
6	.24
7	.16

Table T3.8.3

(a) Assuming that the implicit time unit in a GPSS model is one hour, define a Function named SHIPS for sampling from this distribution. Then show how the Function can be used at a GENERATE Block through which Transactions representing these ships enter a model. What source of random numbers did you specify as the Function's argument? Assuming that the sequence of random numbers in Table 3.1 is in effect, what value will your Function return the first time it is called? What value would it return on the first call if you had used one of the other sources of random numbers?

(b) Show how to re-define the Function in (a) if the implicit time unit is changed from one hour to one minute.

3.8.4 The time required to repair a failed machine is distributed as shown in Table T3.8.4.

Repair Time, Minutes	Relative Frequency
10	.15
20	.20
30	.30
40	.25
50	.10

Table T3.8.4

(a) Assuming that the implicit time unit in a model is one minute, define a Function, numbered 6, for sampling from this distribution. Then show how the Function can be used at an ADVANCE Block where Transactions representing machines are held to simulate repair time. What source of random numbers did you specify as the Function's argument?

(b) Can your Function in (a) ever taken on values such as 12, 15, 23, 37, and so on? If not, this means that repair times of 12, 15, 23, and 37 minutes, etc., can never occur in a model using the Function. Discuss the implications of this in terms of the extent to which such a model simulates reality.

3.8.5 Exercise 2.34.2 describes a machine shop where a single machine is used to polish castings. For convenience, the steps required to polish a casting are repeated below, where the times required are indicated, in minutes, after each step.

(1) Fetch raw casting from a storage area (12±3).
(2) Load raw casting on the polishing machine (10±4).
(3) Carry out polishing phase 1 (80±20).
(4) Re-position the casting on the machine for additional polishing (15±7).
(5) Carry out polishing phase 2 (110±30).
(6) Unload the finished casting from the machine (10±4).
(7) Store the finished casting (12±3), then return to step (1).

(a) Suppose that instead of being uniformly distributed across the closed interval of integers from 9 to 15, the time required to fetch a raw casting from the storage area follows the distribution shown in Table T3.8.5.

Time, Minutes	Relative Frequency
9	.06
10	.14
11	.27
12	.25
13	.16
14	.09
15	.03

Table T3.8.5

(i) Compute the average fetch time for the distribution in Table T3.8.5, and compare it with the average value of 12 previously in effect.

(ii) Compute the standard deviation for the distribution in Table T3.8.5 and compare it with the standard deviation for the population uniformly distributed over the interval of integers 12±3.

(iii) Define a discrete GPSS Function named FETCH which can be used to sample from the Table T3.8.5 distribution.

(iv) Show a graphical interpretation for the Function FETCH.

(b) Referring to step (2), note that the time required to load a raw casting on the polishing machine is 10±4 minutes. Replace this uniform distribution with any non-uniform distribution of your own choosing, retaining only the restriction that loading time ranges over the integers from 6 to 14. After you have done this, carry out the assignments in a(i) through a(iv) for your distribution.

(c) Now prepare a model for Exercise 2.34.2 in which fetch time and load time follow the distributions in (a) and (b), respectively, whereas the distributions for steps (3) through (7) are still uniform as previously indicated. Run the model under the conditions described in Exercise 2.34.2. Has the change in the step (1) and (2) distributions produced a noticeable change in the time the machine operator spends waiting for the overhead crane? If so, can you explain the difference in terms of the changed step (1) and (2) distributions?

3.9. Random Variable Values: Integer vs. Non-Integer

The two random variables of primary interest so far have been inter-arrival time, and service time. Only integer time values are used in GPSS models, because the GPSS clock is of integer mode. When a value is drawn from a distribution of time values, then, it must eventually be treated as an integer.

Recall, for example, how the Processor samples from uniform distributions described via A and B Operands at GENERATE and ADVANCE Blocks. It uses RN1 to compute the value of the sample, then it discards the fractional part. Only the integer portion is retained as the result. No rounding occurs. The Processor follows this same procedure when using user-defined Functions to determine inter-arrival times and service times.

In all the Function examples used so far, the random variables have been described as assuming only integer values. This need not be the case. Consider Figure 3.10, which shows a Function that can take on non-integer values. This Function could be the

```
LOCATION  |  OPERATION  |  A,B,C,D,E,F ------>
1 2 3 4 5 6 | 7 8 9 10 11 12 13 14 15 16 17 18 | 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
MIXED       FUNCTION        RN3,D6
.07,1/.22,1.9/.46,2.6/.67,4.3/.86,5/1,5.6
```

Figure 3.10 A GPSS Discrete Function Which Can Return Non-Integer Values

A Operand at one or more GENERATE and/or ADVANCE Blocks. It is important, though, to realize what range of values will be used at such Blocks if this is done. In general, the values used will differ from the values taken on by the Function because only the integer portion is meaningful in GPSS as a time value.

Suppose the Figure 3.10 Function is used at the GENERATE Block shown in Figure 3.11.

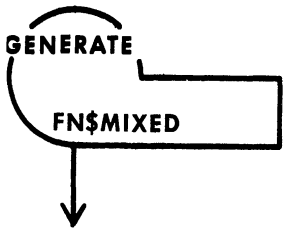


Figure 3.11 Use of the Figure 3.8 Function at a GENERATE Block

Table 3.8 shows the values the Function can assume, and their corresponding relative

Function Value	Relative Frequency	Corresponding Value Used	Relative Frequency of Used Values
1	.07	1	.22
1.9	.15	1	
2.6	.24	2	.24
4.3	.21	4	.21
5	.19	5	.33
5.6	.14	5	

Table 3.8 Inter-Arrival Time Values Used at the Figure 3.9 GENERATE Block

frequencies. For each assumed value, the value which will be used at the Figure 3.11 GENERATE Block is also entered in the table, in column 3. Examination of columns 2 and 3 in Table 3.8 shows that an inter-arrival time of 1 results 22% of the time, 2 occurs 24% of the time, 4 occurs 21% of the time, and 5 occurs 33% of the time.

As shown in Table 3.8, the fact that only the integer portion of Function values is used in a time context means that the effective inter-arrival and service time distributions can vary widely from those which the Functions describe. This deviation may be unacceptable. For example, if the Figure 3.10 Function describes an actual inter-arrival time distribution, measured in hours, then the effective inter-arrival time distribution is a poor approximation to it. This problem can be remedied, of course, by changing the implicit time unit in the model, and re-defining the Function accordingly. In the case of Figure 3.10, the implicit time unit could be changed from 1 hour to 0.1 hours. Then the Function MIXED could be re-defined, with the second member of each ordered pair made larger by a factor of 10. The re-defined function would then take on values which could be used directly as inter-arrival times. There would be no need for the Processor to first discard a fractional portion.

3.10. Continuous Random Variables Viewed Discretely

In Section 3.4, it is pointed out that discrete random variables, by definition, take on a finite number of different values. In contrast, continuous random variables can take on an infinite number of different values. For example, the time between consecutive arrivals of customers at a ticket window can take on an infinity of values, if one is willing to measure time with arbitrarily fine precision. Suppose arrivals at a ticket window always occur within 15 minutes of each other. Table 3.9 then shows how

Time Unit Used, Minutes	Number of Different Values Between 0 (Inclusive) and 15 (Exclusive)
1.0	15
.1	150
.01	1500
.001	15000
.0001	150000
.	.
.	.
.	.

Table 3.9 The Number of Different Values a Random Variable Can Assume, Depending on the Precision of Measurement Being Used

many different values the inter-arrival time random variable can assume, depending on the precision with which time is measured. As the time unit used becomes arbitrarily small, the number of different values becomes arbitrarily large, tending toward infinity.

Strictly speaking, then, inter-arrival time is a continuous random variable. So is service time. So far, however, these two random variables have been treated in this book as though they are discrete, taking on a finite number of values. In fact, for our purposes, all random variables only take on a finite number of values of interest

Beyond some degree of precision, it is no longer meaningful to measure the values these random variables assume. The greatest precision required is that which makes it possible to capture realism in a model. Nothing is gained by requiring precision beyond that point.

It is possible, then, to sample from continuous distributions by discretizing them. After they have been discretized, they can be represented in GPSS by defining discrete Functions for them.

As an example, suppose that a particular inter-arrival time is uniformly and continuously distributed over the interval from 2 minutes, inclusive, to 6 minutes, exclusive. Measured to six digits, this means inter-arrival times such as 4.12274, 2.00783, and 3.57392 can be realized in practice. Because the interval is assumed to include 2 but exclude 6, the smallest and largest possible values are 2.00000 and 5.99999, respectively. If the implicit time unit in a model is 1 minute, and if only the integer portion of the actual six-digit inter-arrival time is used, then values of 2, 3, 4, and 5 will occur with equal likelihood. Table 3.10 shows the relative and

Inter-Arrival Time, Minutes	Relative Frequency	Cumulative Frequency
2	0.25	0.25
3	0.25	0.50
4	0.25	0.75
5	0.25	1.00

Table 3.10 Discretized Version of a Continuous Random Variable

cumulative frequencies for these possible values. Figure 3.10 shows a corresponding GPSS Function for the discretized random variable. The Function is named IAT, for "inter-

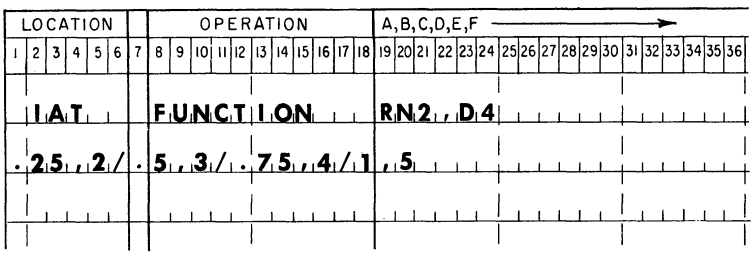


Figure 3.12 A Discrete Function Corresponding to the Table 3.10 Information

arrival time". It might be noted, incidentally, that the uniform distribution of integers between 2 and 5, inclusive, cannot be represented by direct use of the A and B Operands at a GENERATE Block. The list "2, 3, 4, 5" contains an even number of entries, whereas there is always an odd number of entries in the list of integers described by "A±B", where A and B are themselves integers.

3.11. Defining Continuous GPSS Functions

It is suggested in Section 3.10 that the values of continuous random variables can be discretized and used in that form in GPSS. The steps involved were easy to

perform for the Table 3.10 example, but this is not always the case. In fact, it would be extremely impractical to discretize certain important random variables. For this reason, the capability of defining continuous Functions is available in GPSS.

The computational difference between discrete and continuous GPSS Functions will now be described. Then the definitional difference will be indicated. Finally, several examples of elementary GPSS continuous Functions will be given.

A continuous GPSS Function is initially evaluated in the same way as a discrete Function. When the Function is called, a number is first drawn from the random number stream used as the Function's argument. Then, a table-lookup is performed to determine the cumulative probability interval in which the random number falls. At this point, if the Function were defined to be discrete, the second member of the corresponding pair of points would then be returned as the Function's value. In contrast, when the Function is defined as continuous, a linear interpolation is performed next between the pairs of points at the ends of the cumulative probability interval. The number resulting from the interpolation is then returned as the Function's value.

From the graphical point of view, then, a discrete GPSS Function consists of a series of horizontal steps, as has been shown in Figure 3.5; a continuous GPSS Function, on the other hand, consists of a series of straight line segments connecting the pairs of points defining the Function.

A continuous GPSS Function is defined by using C (for "continuous"), rather than D (for "discrete"), as the first character in the B Operand on the Function definition card. With this one exception, discrete and continuous GPSS Functions are defined exactly the same way.

Due to the linear interpolation feature of continuous GPSS Functions, all values in a given cumulative frequency interval have an equally likely chance of occurring. This makes it extremely easy to sample from a continuous, uniform distribution in GPSS. As an example, suppose that a particular random variable is uniformly and continuously distributed over the interval from 2 to 6-. Values of 2 are included in the interval, but values of 6 are excluded. The probability that the random variable's value is less than 2 is zero. The probability it is less than 6 is 1.0. These two cumulative probability values are used to define the continuous GPSS Function shown in Figure 3.13. A graphical interpretation of the Figure 3.13 Function is given in Figure 3.14. Note how

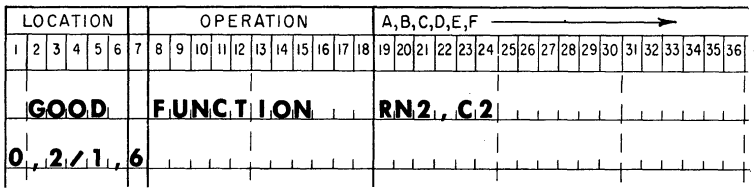


Figure 3.13 A Continuous, Two-Point Function to Sample from A Continuous, Uniformly-Distributed Population

the graph consists of a straight line connecting the two pairs of points, "0,2" and "1,6". When the Function is called, a value of RN2 is fetched, and the value is mapped into a corresponding Function value via linear interpolation. For example, if the value

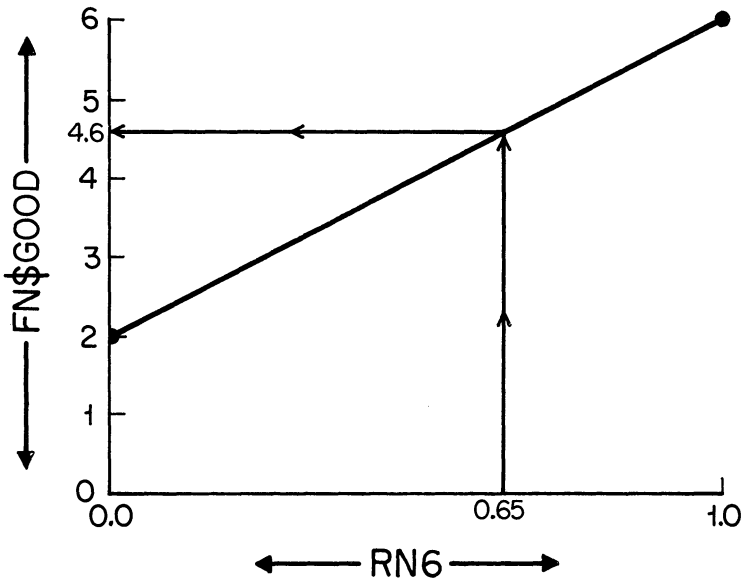


Figure 3.14 A Graphical Interpretation of the Function in Figure 3.13

of RN2 is 0.650000, the value of the Function is 4.60000. This mapping of 0.65 into 4.60 is suggested graphically in Figure 3.14. When the value of RN2 is as small as possible (0.000000), the Function's value is 2. When the value of RN2 is as large as possible (0.999999), the Function's value is 5.99999, measured to six digits.

Now assume that the Figure 3.13 Function is used as the A Operand at a GENERATE Block, as suggested in Figure 3.15. Although the Function takes on values uniformly

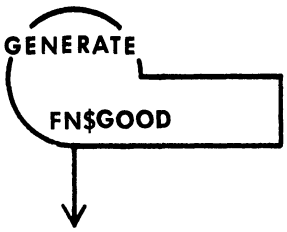


Figure 3.15 Use of the Figure 3.13 Function at a GENERATE Block

distributed between 2 and 6⁻, only the integer portion is used as an inter-arrival time. The effective inter-arrival times will consequently be uniformly distributed over the integers between 2 and 5, inclusive. The reason for this is illustrated in Figure 3.16, where the graph in Figure 3.14 has been repeated. In Figure 3.16, the various cumulative probability intervals leading to effective inter-arrival times of 2, 3, 4, and 5, have been marked off. The corresponding ranges of RN2 values which result in Function values with 2, 3, 4, and 5 as their integer portions are listed in

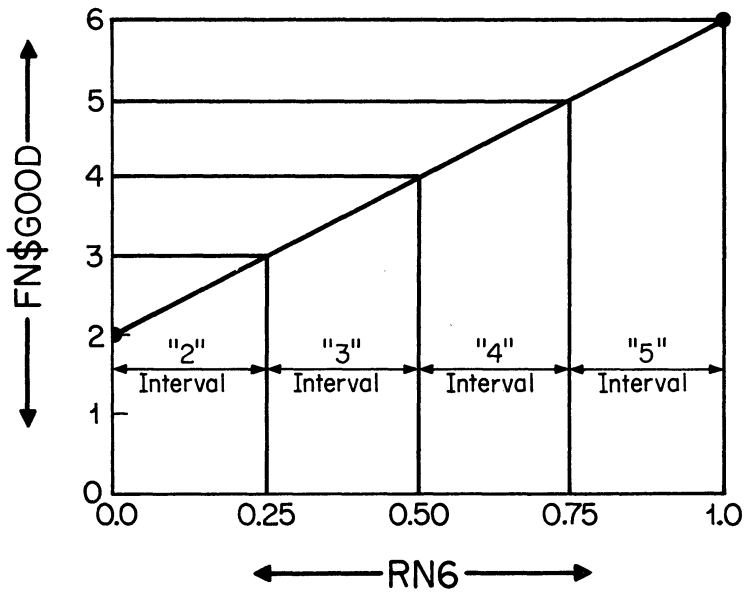


Figure 3.16 Cumulative Probability Intervals Corresponding to Function Values with Integer Portions of 2, 3, 4, and 5

Table 3.11. For example, RN2 values from 0.0 up to, but not including, 0.25 (i.e.,

Integer Portion of Figure 3.14 Function Value	Corresponding Range of RN2 Values
2	0.0 to 0.25⁻
3	0.25 to 0.50⁻
4	0.50 to 0.75⁻
5	0.75 to 1.0⁻

Table 3.11 Correspondence Between Cumulative Probability Intervals and Integer Portion of the Function's Value for the Figure 3.13 Function

0.25⁻), lead to Function values from 2 up to, but not including, 3. The integer portion is consequently 2. This occurs 25% of the time, in the long run. Similarly, function values with integer portions of 3, 4, and 5 occur about 25% of the time, in the long run.

It bears repeating that, although the second pair of points used in the Function definition is "1,6", the Function can never return 6 as its value. RN2 is never larger than 0.999999. With this RN2 value, the returned Function value is 5.99999, measured to six digits. The largest possible integer portion of the returned value is then 5.

The example above demonstrates how a continuous, two-point GPSS Function can be used at GENERATE and/or ADVANCE Blocks to sample from what is effectively a uniform population of integers. The utility of using this approach is not evident in the

preceding example. For a better example, suppose that inter-arrival times are uniformly distributed between 200 and 375 seconds, inclusive. Figure 3.17 shows a continuous, two-point GPSS Function to sample from this population. (Note that the second member

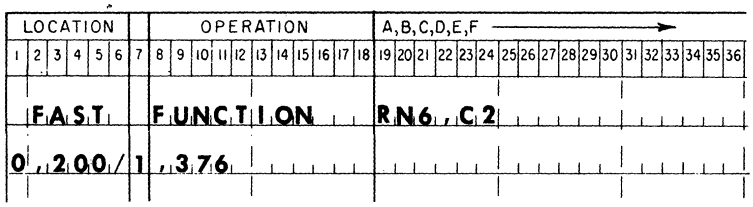


Figure 3.17 A Continuous Function for Sampling from a Uniformly-Distributed Population of Integers

of the second pair of points is 376, not 375.) Imagine defining a discrete GPSS Function to accomplish this same task. A total of 176 cumulative frequencies would have to be computed, and 176 pairs of points would have to be supplied to define the Function. Furthermore, the execution time required to do a table-lookup with the 176-point discrete Function would be much greater than that to perform a simple linear interpolation between two pairs of points. The technique demonstrated in Figure 3.17 will be appealed to later, when it will be of interest to force the Processor to use a random number stream other than RN1 to sample from a uniform population of integers.

3.12. Interpreting Observed Data with Continuous GPSS Functions

When a system is to be modeled, the distribution of various pertinent inter-arrival times and service times is usually not known initially. If the system already exists, it is possible to set up experiments to collect data with which these distributions can be estimated. These data must then be incorporated into the corresponding model of the system. This can be done by using the data to define Functions which can then be referenced from various Blocks in the model.

For example, assume that as part of a bank study, the time required for a teller to serve a customer is measured and classified within 15-second intervals.[3g] The results are shown in Table 3.12. The entries in the table indicate that no customer is

Service Time Interval	Relative Frequency with which Service Time Falls in the Interval	Cumulative Frequency
Less than 15 Seconds	0.0	0.0
From 15 to 30 Seconds	0.07	0.07
From 30 to 45 Seconds	0.25	0.32
From 45 to 60 Seconds	0.41	0.73
From 60 to 75 Seconds	0.19	0.92
From 75 to 90 Seconds	0.08	1.0

Table 3.12 Observed Service Time Data for a Bank Teller

[3g] It is beyond the scope of this book to discuss the design of experiments to gather data, and the evaluation of the resulting measurements. This example is offered, then, without attempting to indicate why the data might have been measured in the manner described.

served in fewer than 15 seconds, and no service completion requires more than 90 seconds. Now suppose that these data are to be used in a model in which the implicit time unit is to be 1 second, and service times are to be represented to the nearest second. In the absence of other information, it is convenient to assume that within any of the indicated 15-second intervals, the various possible service times occur with equal likelihood. This means that the Table 3.12 data can be used to define a continuous GPSS Functi representing the service time distribution.

The definition of a continuous GPSS Function corresponding to the Table 3.12 data is shown in Figure 3.18. The first pair of points in the Function is "0.0,15", indicating the probability is zero that the service time will be less than 15 seconds. The

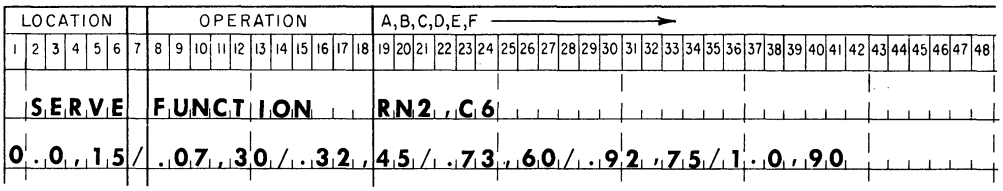


Figure 3.18 A Continuous GPSS Function Constructed from the Table 3.12 Data

second pair of points indicates the probability is 0.07 that the service time is less than 30 seconds. When the value of the Function's argument, RN2, falls in the interval bounded by 0.0 (inclusive) and 0.07 (exclusive), the Processor interpolates linearly between service times of 15 and 30 to determine the Function's value. Because all RN2 values between 0.0 and 0.07 have equally likely chances of occurring, this means all service times between 15 and 30 seconds have equal chances of being the Function's value, given that RN2 falls in the 0.0-0.07 interval. These same remarks are true for service times from 30 to 45 seconds; 45 to 60 seconds; 60 to 75 seconds; and 75 to 90 seconds. Hence, for each particular interval, the Function values within that interval are uniformly distributed. This situation is summarized in Figure 3.19, which shows a graphical interpretation of the Figure 3.18 Function.

The Figure 3.19 graph can be thought of as a series of straight-line segments which are being used to approximate the true, but unknown, function describing the bank teller's service time distribution. Figure 3.20 repeats Figure 3.19, but includes a continuous curve intended to suggest what the true distribution function might be. The fact that the consecutive straight line segments only approximate the true underlying distribution is evident in the Figure.

In this book, inter-arrival time and service time distributions will frequently be described by supplying information intended to summarize observed data, in the spirit of the preceding example. Whenever this is done, only the random variable values and their corresponding cumulative frequencies will be stated. For example, the information in Table 3.12 would be provided as shown in Table 3.13. Whenever such information is given, the following is to be understood.

(1) The random variable is continuously distributed.

(2) The distribution is to be described with a continuous GPSS Function.

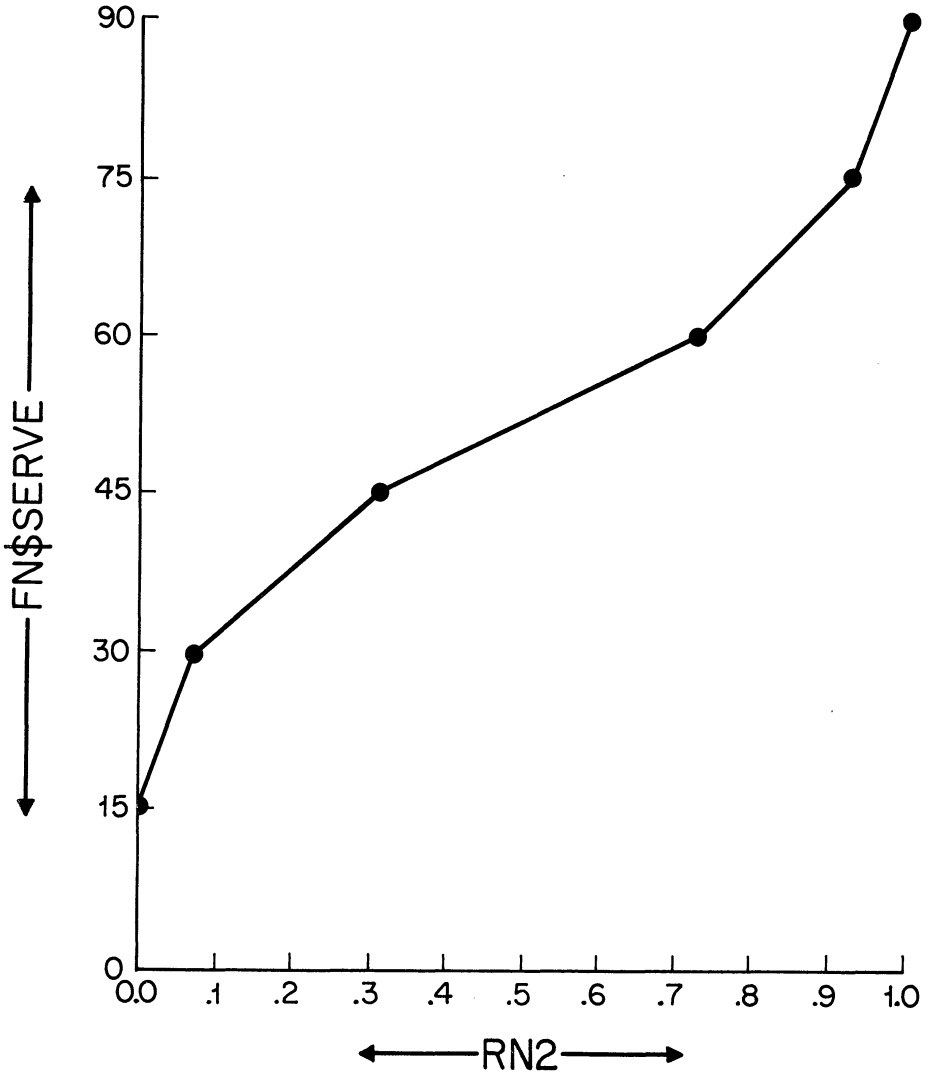


Figure 3.19 Graphical Interpretation of the Figure 3.18 Function

(3) The resulting GPSS Function is an approximation to the true underlying cumulative distribution, whatever that might be. Because of the GPSS linear interpolation feature, the true distribution is being represented with a sequence of straight line segments.

Value of Random Variable	Cumulative Frequency
Less than 15	0.0
30	0.07
45	0.26
60	0.51
75	0.82
90	1.0

Table 3.13 Condensed Version of the Information in Table 3.12

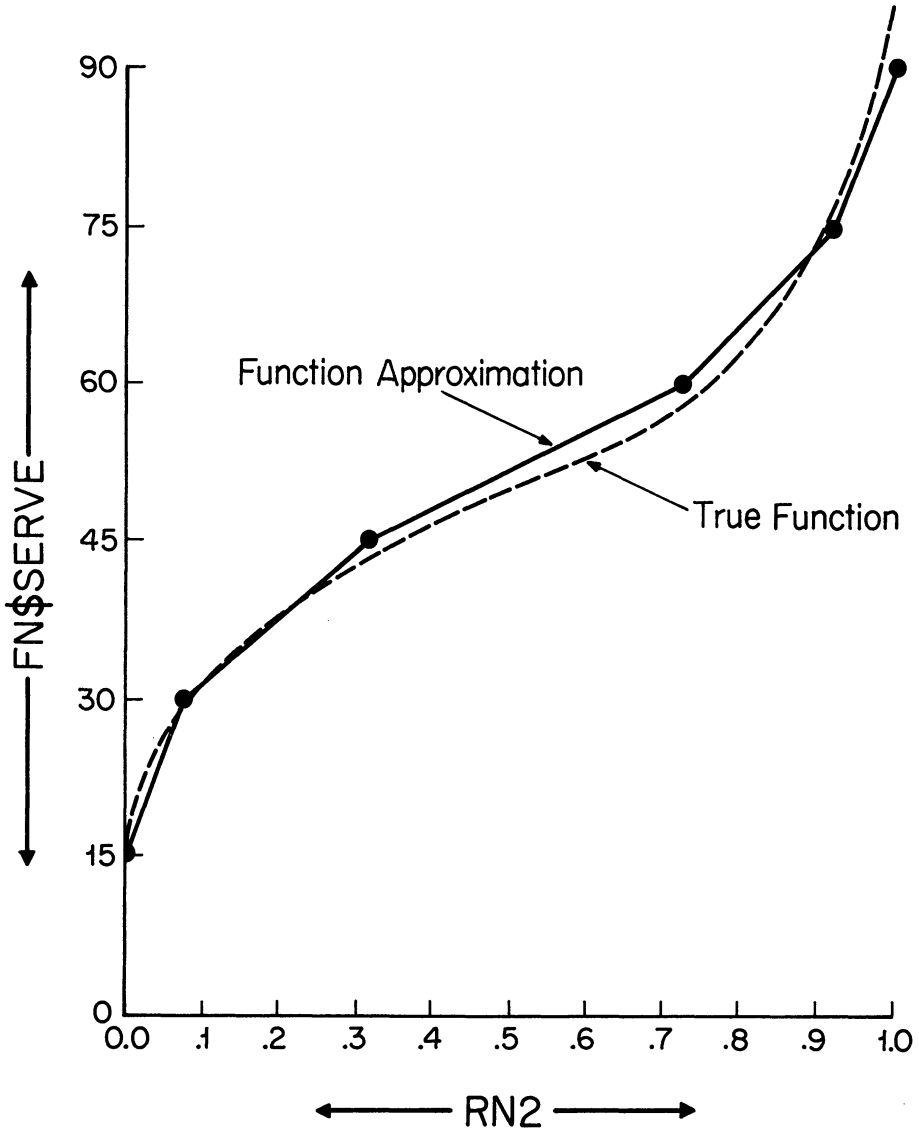


Figure 3.20 Graphical Indication of the Manner in Which the Figure 3.19 Function Approximates the True Service Time Distribution

3.13. Simulation of Poisson Arrivals

When inter-arrival times are experimentally observed, it is sometimes found that the following conditions are satisfied.

(1) The probability that an arrival occurs during a small time interval is proportional to the size of the interval.

(2) The probability of two or more arrivals during a small time interval is negligibly small.

(3) The inter-arrival times are independent of each other.

Under these conditions, analytic results can be developed which describe the distribution of the arrival rate. The distribution is shown in Equation 3.1, where

$$P_k(T) = \frac{e^{-\lambda T}(\lambda T)^k}{k!} \quad , k = 0, 1, 2, 3, \cdots$$

Equation 3.1 Relative Frequency Function for the Poisson Distribution

$P_k(T)$ is the probability that exactly k arrivals will occur during a time interval of duration T, λ is the mean arrival rate relative to that time interval, and e is the base of the natural logarithms. The distribution described by Equation 3.1 is named after the French mathematician credited with its development, Poisson. When a "Poisson arrival process" is in effect, it is understood that the arrival rates are Poisson-distributed, i.e., the rate distribution is described by Equation 3.1. Similarly, the three conditions listed above and leading to the Poisson distribution are termed the "Poisson assumptions".

When a Poisson arrival process is to be simulated, it is not arrival rates which are of direct interest; instead, it is the corresponding inter-arrival times which must be known. This is consistent with the approach of computing the time of the next Transaction's arrival at a GENERATE Block by adding to the clock's current reading a value drawn from an inter-arrival time distribution. Equation 3.1 can be manipulated to produce the associated distribution of inter-arrival times.[3h] The result is called the exponential distribution. When arrival rates are Poisson-distributed, then, the corresponding inter-arrival times are exponentially distributed.

The exponential distribution can be put into such a form that, given a value from the 0-1 uniform distribution, the corresponding inter-arrival time can be directly computed. The pertinent equation is shown as Equation 3.2. In that equation, IAT_{sample} stands

$$IAT_{sample} = (IAT_{avg}) \ (-\log_e[1-RNj])$$

Equation 3.2 Function for Sampling From the Exponential Distribution

for "inter-arrival time". IAT_{avg} is the average inter-arrival time in effect. RNj is the name of one of the GPSS uniform random number generators, where the choice of j, as usual, is up to the analyst. $\log_e$ represents the natural logarithm operation. To draw a sample from the exponential distribution whose average value is IAT_{avg}, then, Equation 3.2 indicates that this sequence be followed: draw a value from a 0-1 uniform distribution; compute the natural logarithm of 1 minus this random number; finally, multiply the negative of this natural logarithm by IAT_{avg}. Note that, because the value of RNj is less than 1, 1-RNj is also less than 1. The natural logarithm of 1-RNj is consequently negative. The negative of this logarithm, in turn, is positive; and, because IAT_{avg} must be positive, IAT_{sample} is itself a positive number.

Making use of Equation 3.2 to simulate Poisson arrivals at a GENERATE Block involves two steps. First, given a value of RNj, $-\log_e[1-RNj]$ must be computed. Then, this result must be multiplied by IAT_{avg}. The first step is accomplished by defining a GPSS Function having RNj as its argument, and returning $-\log_e[1-RNj]$ as its value.

[3h] Appendix E shows how the three Poisson assumptions are used to develop Equation 3.1, the exponential distribution, and Equation 3.2.

In algebraic programming languages such as FORTRAN, BASIC, and PL/I, for example, a logarithm function is provided as part of the language. Given a value of RNj, the value of $-\log_e[1-RNj]$ could be directly computed in such a language. There is no logarithm function in GPSS. As a result, the distribution represented by $-\log_e[1-RNj]$ must be approximated with a continuous GPSS Function constructed by fitting a series of straight-line segments to the true function. An approximation suitable for this purpose has been developed by IBM. The approximation consists of a series of 23 line segments which span the range of exponential random variable values from 0 to 8.[3i] Figure 3.21 shows the

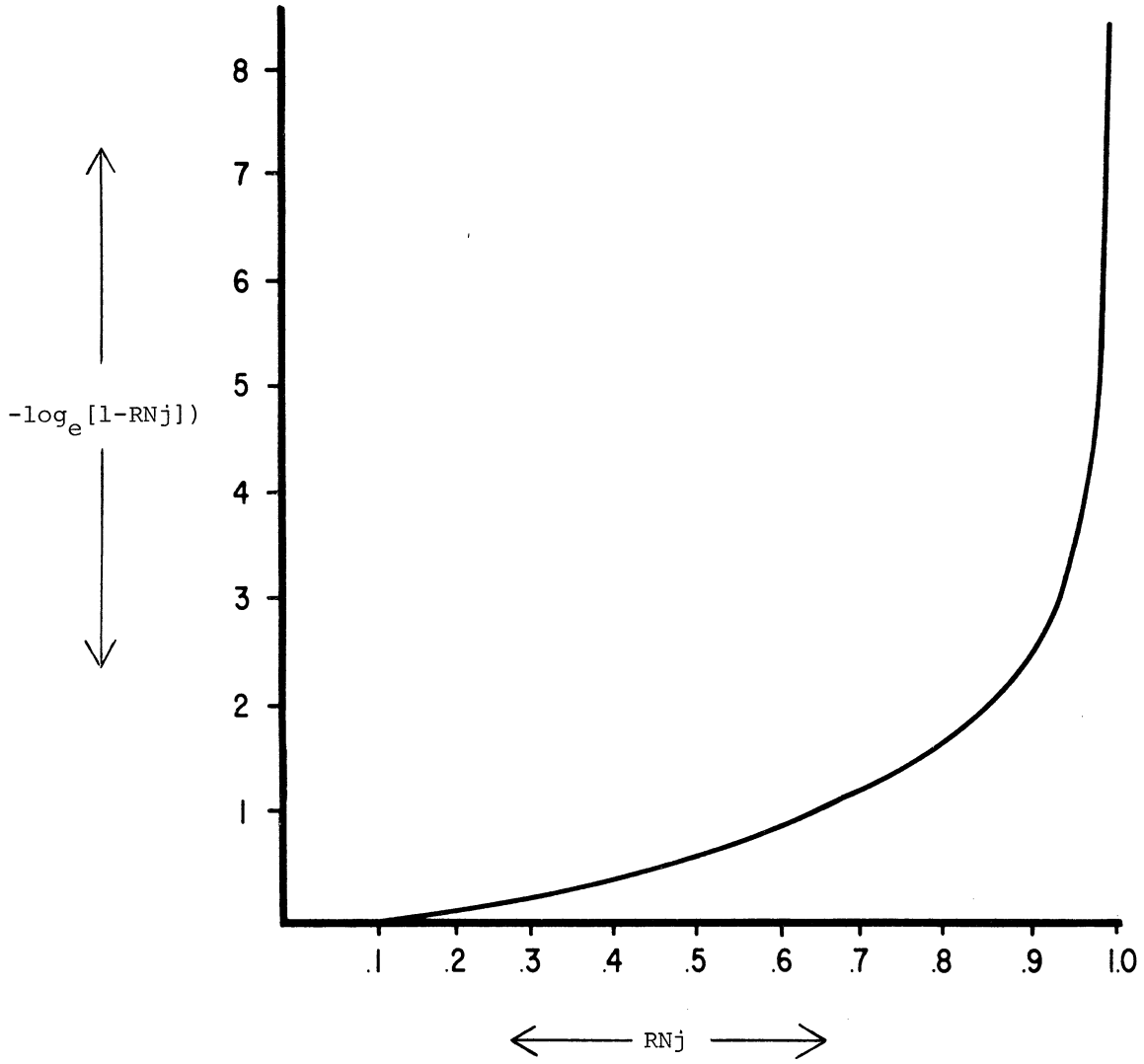


Figure 3.21 Graph of the Function Used to Sample from the Exponential Distribution

true behavior of $-\log_e[1-RNj]$, as RNj varies from 0 to 1. The 23 line segments used to approximate this true behavior are not shown in the figure, because for all practical purposes they fall directly on the true curve. Figure 3.22 shows the definition of the

[3i] The approximation can be found in any of IBM's GPSS user manuals or Introductory Users Manuals, as well as in Figure 3.22.

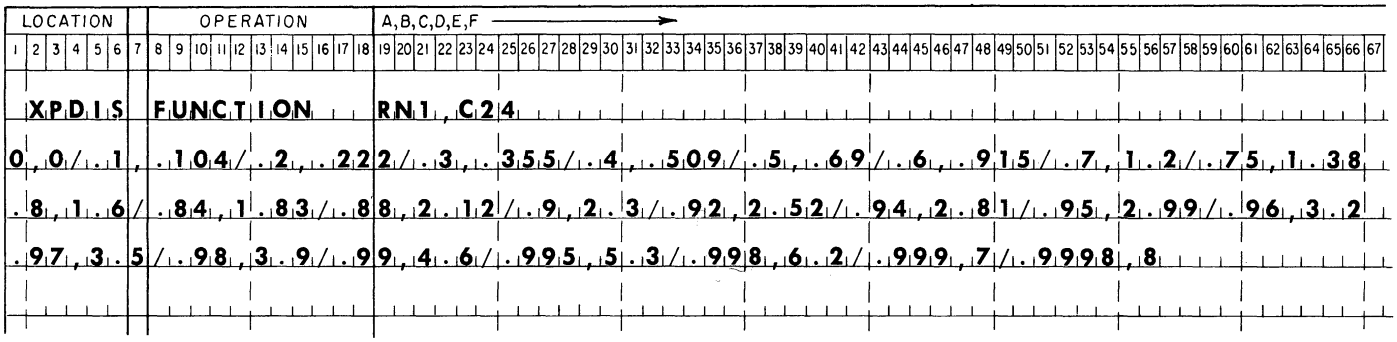


Figure 3.22 Continuous 24-Point GPSS Function Used to Sample from the Exponential Distribution

continuous GPSS Function composed of the 24 pairs of points used to convert values of RNj into values of $-\log_e[1-RNj]$. The Function in Figure 3.22 has been symbolically named XPDIS, for "exponential distribution". RN1 has been arbitrarily chosen as the Function's argument. Note that, whereas the values of $-\log_e[1-RN1]$ can range from 0 (for RN1 values of 0) to infinity (for RN1 values arbitrarily close to 1), the values taken on by FN$XPDIS can only vary from 0 to 8 (the last ordered pair in Figure 3.22 is ".9998,8"). Recalling that the RNj values can be as large as .999999, this means that the argument for the Function XPDIS will be out of range with probability 0.000199 (i.e., .999999 minus .9998), or about .0199% of the time. Whenever this happens, the value assumed by FN$XPDIS will be 8, whereas it should be something larger than that. In view of the other inaccuracies in simulation modeling (for example, use of approximate data, and making simulation runs of finite duration), the small discrepancies which arise in using the Figure 3.22 function to approximate $-\log_e[1-RN1]$ are acceptable in most cases.

Given that the values of FN$XPDIS are a satisfactory approximation of $-\log_e[1-RN1]$, these values must now somehow be multiplied by IAT_{avg} so that, per Equation 3.2, the inter-arrival times corresponding to a Poisson arrival process can be computed. A special provision is made in GPSS at the GENERATE Block to perform this multiplication automatically. When the GENERATE Block B Operand is FNj (or FN$sn), inter-arrival time is computed at the Block by multiplying the value of the Function numbered j (or symbolically named sn) by the A Operand. The integer portion of the product is then used as an inter-arrival time. This means that a Poisson arrival process can be simulated at a GENERATE Block by doing these two things.

(1) For the A Operand, use the average inter-arrival time.

(2) For the B Operand, use FN$XPDIS, where the Function XPDIS is defined in Figure 3.22.

Suppose, for example, that at a certain point in a system arrivals occur in a Poisson stream (i.e., according to a Poisson arrival process) at an average rate of 4 arrivals every 24 hours. These arrivals are to be simulated in a model, using 1 minute as the implicit time unit. A GENERATE Block accomplishg this effect is shown in

Figure 3.23. The A Operand, 360, has been determined by (a) converting the given arrival

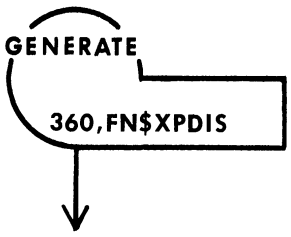


Figure 3.23 Example of a GENERATE Block Used to Simulate a Poisson Arrival Process

rate to the corresponding average inter-arrival time, and (b) expressing the average inter-arrival time in terms of the indicated implicit time unit. Hence, when 4 arrivals occur every 24 hours, the average time between consecutive arrivals is 6 hours, or 360 minutes. When a Transaction arrives at the Figure 3.23 GENERATE Block and moves to the next Block, the Processor will compute its successor's time of arrival by evaluating FN$XPDIS, multiplying the result by 360, and then adding the integer portion of the product to the current clock value. To evaluate the Function XPDIS, the value of RN1 must first be determined. If RN1 is, say, 0.35, then FN$XPDIS is 0.432 (halfway between .355 and .509; see the fourth and fifth pairs of points in Figure 3.22). The product of 360 and 0.432 is 155.52. The corresponding inter-arrival time, then, is 155.

The preceding example shows that, when a Function is referenced as a GENERATE Block's B Operand, the Function's value is used as a multiplier of the A Operand. This is the first context encountered so far in which a Function's true value, rather than just the integer portion, is used in a GPSS computation.

When a Function is used as a GENERATE Block's B Operand, the average inter-arrival time (A Operand) is said to be Function-modified. In contrast, when the B Operand is anything other than a Function reference, the average inter-arrival time is said to be spread-modified. Spread-modified inter-arrival times are always understood to be uniformly distributed over the interval of integers A±B (i.e., the A Operand plus or minus the B Operand). Function-modified inter-arrival times are always understood to be the integer portion of the product of A and B (i.e., A Operand times the value of the Function referenced with the B Operand). Strictly speaking, any Function can be used as a GENERATE Block B Operand. In this book, however, only the Figure 3.22 Function will be used in this context. In fact, except to simulate Poisson arrivals, the author knows of no other compelling reason to use the Function-modifier option at GENERATE Blocks.

It should be noted that the Function XPDIS can be referenced from any number of different points in a model. At one point, for example, Poisson arrivals may occur at an average rate of 4 every 24 hours. At another point, arrivals may occur in a Poisson stream at an average rate of 3 every 24 hours. The two Blocks "GENERATE 360,FN$XPDIS" and "GENERATE 480,FN$XPDIS" can be used at the corresponding points in the model to simulate the arrivals. Both GENERATE Blocks make use of the same Function, XPDIS.

One of the implications of the second Poisson assumption ("the probability of two or more arrivals during a small interval of time is negligibly small") is that two

or more arrivals do not occur simultaneously. Put differently, this means that when arrivals occur in a Poisson pattern, inter-arrival time is never zero. Now, suppose Poisson arrivals are being simulated in a model at the Block "GENERATE 5,FN$XPDIS". Whenever the value of the Function XPDIS is less than 1/5, the product of 5 and FN$XPDIS will be less than 1. The integer portion of the product will then be zero, thereby violating the second Poisson assumption. Examination of Figure 3.22 shows that FN$XPDIS will have a value less than 1/5 about 18.5% of the time (whenever RN1 is less than about .185, the Function's returned value is less than 0.2; the value .185 results from inverse interpolation between the points ".1,.104" and ".2,.222"). The second Poisson assumption will then frequently be violated in the model. On the other hand, suppose that in the same model, the implicit time unit is made smaller by a factor of 10. The GENERATE Block must then be changed, becoming "GENERATE 50,FN$XPDIS". For the product of 50 and FN$XPDIS to be less than 1, the value of FN$XPDIS must be less than 1/50. Examination of Figure 3.22 shows that this will only occur with a probability of about 0.02, or 2% of the time. Violation of the second Poisson assumption to this relatively small extent may be acceptable in the model. These two examples suggest, then, that if the second Poisson assumption is not to be grossly violated, the A Operand at pertinent GENERATE Blocks must be kept large by choosing a correspondingly small implicit time unit. In this context, it is recommended that GENERATE Block A Operands not be smaller than 50.

It is of interest to note the variation possible in inter-arrival times when a Poisson arrival process is in effect. Each particular inter-arrival time is computed by multiplying the average by a factor that can range in value from 0 to 8. If the average is 50, for example, then realized inter-arrival times can vary from 0 to 400. Statistically, the variance of exponentially-distributed inter-arrival times is the square of the average value. Because of the wide variation possible, large sample sizes (i.e., long simulation runs) are often required to obtain "representative" long-run results when exponential sources of randomness are present in a model.

3.14. Exercises

3.14.1 (a) What are the three Poisson assumptions?

(b) In words, how are the Poisson and exponential distributions related?

(c) What is the relationship between an average arrival rate and the corresponding average inter-arrival time?

(d) Explain the difference between using a spread-modifier and a Function-modifier as the B Operand at a GENERATE Block.

(e) What must the first two characters be in a GENERATE Block's B Operand if a Function modifier is being used?

(f) Why is it important to have a relatively large A Operand at a GENERATE Block be used to simulate a Poisson arrival process?

3.14.2 (a) In the long run, what percentage of the time will inter-arrival times of zero be used at the Block "GENERATE FN$XPDIS"? (The Function XPDIS is defined in Figure 3.22.)

(b) What is the smallest value that can be used as a GENERATE Block's A Operand if the Block is being used to simulate Poisson arrivals, and inter-arrival times of zero are to occur not more than 0.5% of the time in the long run?

3.14.3 Ships arrive at a harbor in a Poisson pattern at an average rate of four ships per week. Assuming that the implicit time unit in a GPSS model is one hour, show a GENERATE Block which can be used to simulate the arrival of these ships. What percentage of the time will the computed inter-arrival time turn out to be zero?

3.14.4 What inter-arrival times can be realized at these four GENERATE Blocks.

(a) GENERATE FN$IAT

(b) GENERATE 1,FN$IAT

(c) GENERATE 2, FN$IAT

(d) GENERATE FN$IAT,2

The Function IAT is defined in Figure P3.14.4.

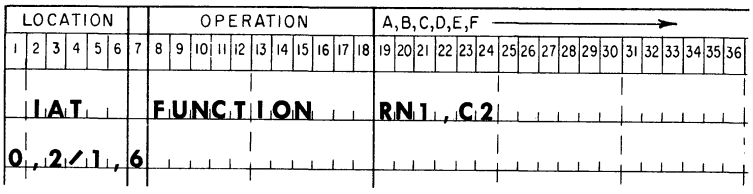


Figure P3.14.4

3.15. Simulation of Exponentially-Distributed Holding Times

Important random variables other than inter-arrival times are sometimes found to be exponentially distributed. For example, the duration of telephone conversations follows the exponential distribution. So does the life of electronic equipment components, such as vacuum tubes. The time required by a server to perform a service is also sometimes exponentially distributed. The practical consequence of this is that it can be of interest to have the holding time at one or more ADVANCE Blocks be determined by drawing a sample from the exponential distribution.

Analogous to computation of exponentially-distributed inter-arrival times, a particular exponentially-distributed holding time, or "service time", can be computed by multiplying the average holding time by the value of $-\log_e[1-RNj]$. This is suggested in Equation 3.3, where HT_{sample} is the holding-time sample, and HT_{avg} is the average holding time

$$HT_{sample} = (HT_{avg})\ (-\log_e[1-RNj])$$

Equation 3.3 Sampling from the Exponential Distribution in a Service Time Context

in effect. As before, values of $-\log_e[1-RNj]$ can be determined by use of the 24-point Continuous GPSS Function defined in Figure 3.22. The Function-modifier option available via the GENERATE Block's B Operand is also available with the ADVANCE Block. That is, when the ADVANCE Block's B Operand is FNj (or FN$sn), holding time at the Block is computed by multiplying the value of the referenced Function by the Block's A Operand. The integer portion of the product is then used as the holding time. This means that exponential holding times can be simulated at an ADVANCE Block by supplying the average holding time as its A Operand, and using FN$XPDIS as its B Operand. For example, if the average service time required by a particular "exponential server" is 75 time units, the Block in Figure 3.24 can be used to simulate the server in a model.

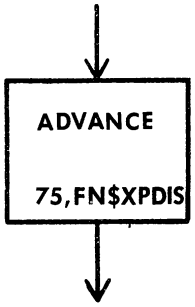


Figure 3.24 Example of an ADVANCE Block Used to Simulate an Exponential Holding Time

The same remarks made concerning simulation of Poisson arrival patterns at GENERATE Blocks are in effect for simulating exponential holding times at ADVANCE Blocks. Care should be taken to distinguish between Function and spread modifiers at ADVANCE Blocks. Furthermore, the implicit time unit in a model should be sufficiently small so that when exponential holding times are to be simulated, the corresponding average holding times are no smaller than 50.

3.16. Case Study 3B: Poisson Arrivals to an Exponential Server

(1) Statement of the Problem

At a one-car wash facility, it must be decided how many spaces to provide for cars waiting to use the facility. Cars arrive in a Poisson stream with an average interarrival time of 5 minutes. Car washing time is exponentially distributed with a mean of 4 minutes. Potential customers who find no waiting space available go elsewhere to have their car washed.

Build a GPSS model for this simple system, then use the model to observe system behavior for the alternatives of 1, 2, and 3 waiting spaces. For each configuration, simulate for one 8-hour day of operation. Based on model output, estimate the fraction of potential customers who actually remain at the car wash to be served. Compare the estimate for each configuration with the theoretically-computed fraction of customers who will be served in the long run.

(2) Approach Taken in Building the Model

The model is straightforward. A Storage is used to simulate the number of waiting spaces at the car wash. When a customer-Transaction arrives at the system, it enters a TRANSFER Block operating in BOTH mode. From this Block, the customer-Transaction attempts to enter the Storage used to simulate waiting space. If entry is denied, the Transaction moves immediately to a TERMINATE Block. Based on Block Counts available in the output, it is a simple matter to compute the fraction of customers who arrived at the car wash and then remained for service, rather than immediately leaving the system.

(3) Table of Definitions

Time Unit: 0.01 Minutes

GPSS Entity	Interpretation
Transactions	
Model Segment 1	Customers
Model Segment 2	A timer
Facilities	
WASHR	Car-wash capability
Functions	
XPDIS	Function for sampling from the exponential distribution with a mean value of 1
Storages	
SPACE	Storage simulating the number of waiting spaces available

Table 3B.1 Table of Definitions for Case Study 3B

(4) Block Diagram

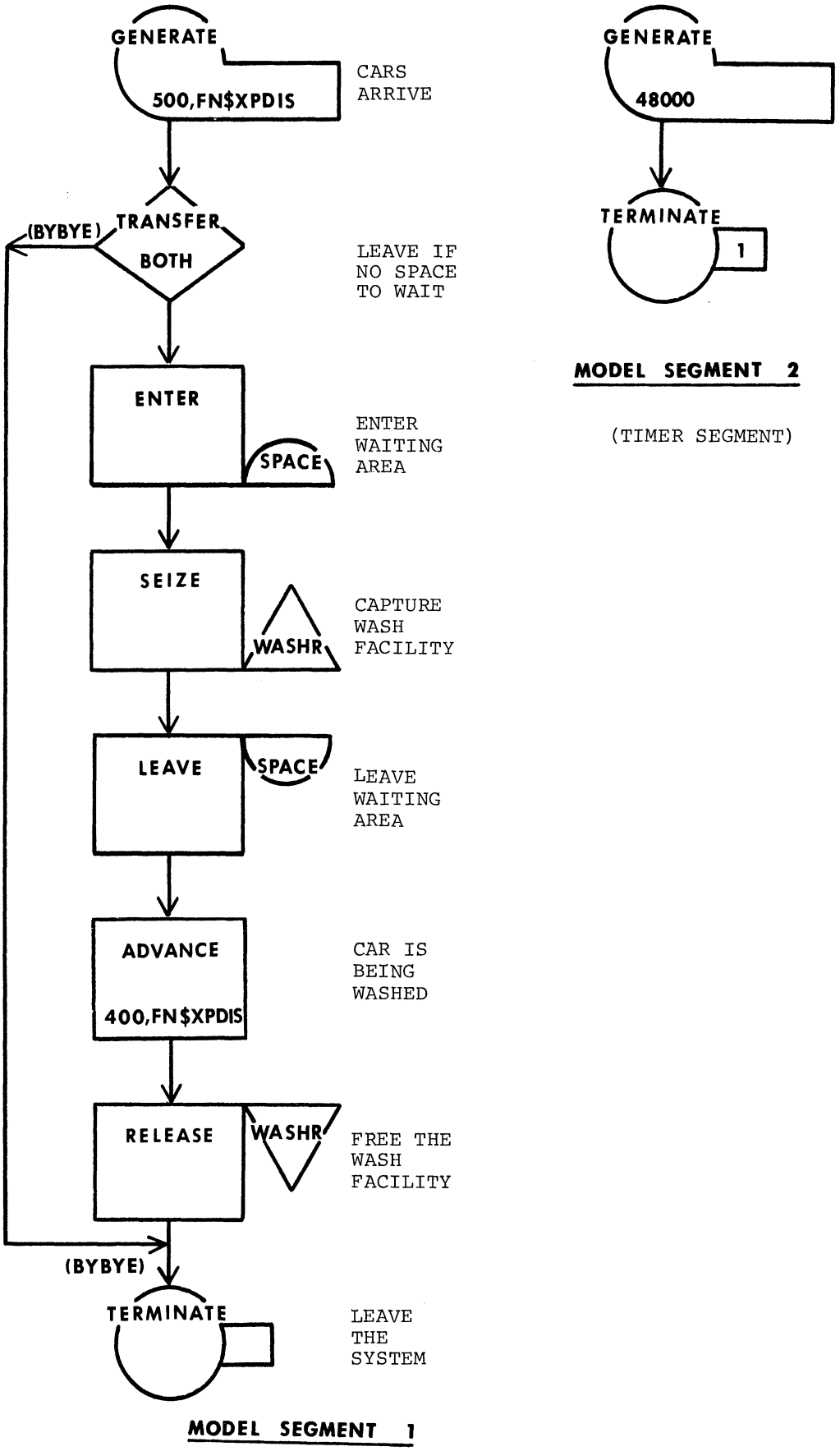


Figure 3B.1 Block Diagram for Case Study 3B

(5) Extended Program Listing

```
BLOCK                                                                    CARD
NUMBER   *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS               NUMBER
                SIMULATE                                                    1
         *                                                                  2
         *      FUNCTION DEFINITIONS                                        3
         *                                                                  4
          XPDIS FUNCTION   RN1,C24          EXPONENTIAL DISTRIBUTION FUNCTION 5
         0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38 6
         .8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2 7
         .97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8           8
         *                                                                  9
         *      STORAGE CAPACITY DEFINITIONS                                10
         *                                                                  11
                STORAGE    S$SPACE,1      ONE WAITING SPACE INITIALLY       12
         *                                                                  13
         *      MODEL SEGMENT 1                                             14
         *                                                                  15
1                GENERATE   500,FN$XPDIS   CARS ARRIVE                      16
2                TRANSFER   BOTH,,BYBYE    LEAVE IF NO SPACE TO WAIT        17
3                ENTER      SPACE          ENTER WAITING AREA               18
4                SEIZE      WASHR          CAPTURE WASH FACILITY            19
5                LEAVE      SPACE          LEAVE WAITING AREA               20
6                ADVANCE    400,FN$XPDIS   CAR IS BEING WASHED              21
7                RELEASE    WASHR          FREE THE WASH FACILITY           22
8         BYBYE  TERMINATE                 LEAVE THE SYSTEM                 23
         *                                                                  24
         *      MODEL SEGMENT 2                                             25
         *                                                                  26
9                GENERATE   48000          TIMER ARRIVES AT END OF THE DAY  27
10               TERMINATE  1              SHUT OFF THE RUN                 28
         *                                                                  29
         *      CONTROL CARDS AND STORAGE CAPACITY CHANGES                  30
         *                                                                  31
                START      1              START THE 1ST RUN                32
                STORAGE    S$SPACE,2      SET WAITING SPACES = 2 FOR 2ND RUN 33
                CLEAR                       CLEAR FOR 2ND RUN               34
                START      1              START THE 2ND RUN                35
                STORAGE    S$SPACE,3      SET WAITING SPACES = 3 FOR 3RD RUN 36
                CLEAR                       CLEAR FOR 3RD RUN               37
                START      1              START THE 3RD RUN                38
                END        RETURN CONTROL TO OPERATING SYSTEM              39
```

Figure 3B.2 Extended Program Listing for Case Study 3B

RELATIVE CLOCK 48000 ABSOLUTE CLOCK 48000

BLOCK COUNTS

BLOCK	CURRENT	TOTAL	BLOCK	CURRENT	TOTAL	BLOCK	CURRENT
1	0	94					
2	0	94					
3	0	62					
4	0	62					
5	0	62					
6	0	62					
7	0	62					
8	0	94					
9	0	1					
10	0	1					

(a) Clock Values and Block Counts

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
WASHR	.674	62	522.145		

(b) Facility Statistics

STORAGE	CAPACITY	AVERAGE CONTENTS	AVERAGE UTILIZATION	ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENTS	MAXIMUM CONTENTS
SPACE	1	.294	.294	62	227.661		1

(c) Storage Statistics

Figure 3B.3 Selected Program Output for Case Study 3B (1 Waiting Space Provided)

(7) Discussion

Model Implementation

Note in the Table of Definitions that a time unit of 0.01 minutes is in effect. With this small implicit time unit, the average interarrival time and service time for the Poisson arrivals and the exponential server are 500 and 400, respectively. These large values insure that the method followed in sampling from the exponential distribution produces results faithful to the Poisson assumptions to a satisfactory degree.

Program Output

Typical program output is shown in Figure 3B.4, for the case of 1 waiting space. During the simulation, 94 customers arrived at the system [Total Count, Block 1, Figure 3B.4(a)]. Of these 94, 62 remained to be served [Total Count, Block 3 (the ENTER Block)]. Hence, 32 customers were lost. Some 66% of the customers were served. At the time of model shutdown, no customers were in the system (all Current Counts are zero)

The Facility WASHR has an average utilization of 0.674 [Figure 3B.4(b)]. This figure can be compared with the long-run Facility utilization of 0.80 which would result in a perfectly synchronized system.

The average car-wash time for the sample of size 62 is 522 time units, or 5.22 minutes. There is considerable variation, then, between this observed value and the long-run average value of 4 minutes. On the other hand, the 94 customers who arrived at the system closely approximated the expected number of arrivals, 96.

Figure 3B.4(c) displays statistics for the Storage SPACE. Even though 34% of the customers found no waiting space available, utilization of the Storage is only 0.294. This fact, coupled with the large discrepancy between observed and expected Facility holding time, suggests that Poisson arrivals and/or exponential servers are characterized by a high degree of variability, producing results which can differ markedly from those which could be realized if perfect synchronization were achieved.

Table 3B.2 shows a summary of the model output for the cases of 1, 2, and 3

Number of Waiting Spaces	Waiting Space Utilization	Number of Customers Served	Number of Customers Lost	Server Utilization	Average Se Time
1	0.294	62	32	0.674	522
2	0.233	83	15	0.670	388
3	0.272	82	9	0.660	387

Table 3B.2 Summary of Program Output

waiting spaces. The observed number of lost customers is cut from 32 to 15 when 2 waiting spaces are available rather than just 1. With three waiting spaces, only 9 customers were lost.

Average Facility holding times are entered in the rightmost column in Table 3B.2. For the cases of 2 and 3 waiting spaces, these averages happen to be close to the long-run value of 400. Note that the corresponding sample sizes (number of customers served) range from 62 to 83.

Table 3B.3 compares the observed fraction of customers served with the

Number of Waiting Spaces	Fraction of Customers Served	
	Observed	Theoretical Value
1	0.660	0.738
2	0.846	0.826
3	0.902	0.861

Table 3B.3 Comparison of Observed Fraction of Customers Served to the Corresponding Known Long-Run Values

corresponding long-run values known to be in effect. The long-run values have been computed on theoretical grounds from equation 3B.1.[3j] With the exception of the configuration providing 1 waiting space, the observed results compare reasonably well with

$$\text{Fraction Served} = 1 - \left[\frac{1-\rho}{1-\rho^{M+1}}\right]\rho$$

where M is one larger than the number of waiting spaces, and

ρ is the ratio of mean service time to mean interarrival time.

Equation 3B.1 Result for Computing the Expected Fraction of Poisson Arrivals Serviced by an Exponential Server, When Waiting Space is Finite

their expected values. If the average service time in the first case had been closer to its expected value, this would have tended to increase the fraction of customers served, bringing the estimate into closer agreement with the expected values in this case.

[3j] For the development of this equation, see Chapter 10 in Frederick S. Hiller and Gerald J. Lieberman, Introduction to Operations Research (Holden-Day, 1967).

3.17. Exercises

Problems 3.17.1 through 3.17.4 are taken from Chapter 10 in Introduction to Operations Research, by Frederick S. Hillier and Gerald J. Lieberman [Copyright (c) 1967 by Holden-Day, Inc.; by permission of Holden-Day, Inc.; no further reproduction or distribution is authorized without permission of Holden-Day, Inc.]. Like Case Study 3B, they are typical of queuing systems problems for which analytic results are available. For each problem, build a GPSS model which simulates the behavior of the corresponding system, then run the model and interpret the output to provide answers (estimates) for the questions asked. Compare these estimates with the theoretical results, which are given after each problem statement.

Note that, in practice, the arrival processes in effect in these problem contexts may not be Poisson, and the service time distributions may not be exponential. In such cases, analytic results very likely would not be available. Computer models for the systems would consequently provide useful tools with which to investigate system behavior.

It should also be noted that each analytic result given below is the average value of a random variable, i.e., describes average system behavior in the long run. The analytic results provide no indication of the extent to which short-run experience can differ from experiences averaged over a long period of time. Simulation models have the advantage of generating results which could actually take place in the short run. If enough such short-run observations are made, the distributions of the random variables of interest can be estimated. The method of estimating such probability distributions with GPSS models is discussed in Chapter 4.

3.17.1 It is necessary to determine how much in-process storage space to allocate to a particular work center in a new factory. Jobs would arrive at this work center according to a Poisson process with a mean rate of two per hour, and the time required to perform the necessary work has an exponential distribution with a mean of 0.4 hours. If each job would require one square foot of floor space while in in-process storage at the work center, how much space must be provided in order to accommodate all waiting jobs (a) 50%, (b) 90%, (c) 99% of the time?

Answers: 2, 9, and 19 square feet of floor space, respectively.

3.17.2 Suppose that one repairman has been assigned the responsibility of maintaining three machines. For each machine, the probability distribution of the running time before a breakdown is exponential with a mean of eight hours. The repair time also has an exponential distribution with a mean of two hours. Determine the expected number of machines that are not running.

Answer: 0.80

3.17.3 A company currently has two tool cribs, each with a single clerk, in its manufacturing area. One tool crib handles only the tools for the heavy machinery, while the second one handles all other tools. However, for each crib, the arrival process is Poisson with a mean rate of 20 per hour,

and the service-time distribution is exponential with a mean of two minutes.

Because of complaints that the mechanics coming to the tool cribs have to wait too long, the proposal has been made to combine the two tool cribs so that either clerk can handle either kind of tool as the demand arises. It is believed that the mean arrival rate to the combined two-clerk tool crib would double to forty per hour, while the expected service time would continue to be two minutes.

Compare the current approach and the proposed new approach with respect to the total expected number of mechanics at the tool crib(s), and the expected waiting time (including service) for each mechanic.

Answers: For the current approach, there are 2 mechanics on average at each tool crib, and expected waiting time is 6 minutes. When the two tool cribs are combined, the total number of waiting mechanics averages 2.4, and the expected waiting time is 3.6 minutes.

3.17.4 A particular work center in a job shop can be represented as a single-server queuing system where jobs arrive according to a Poisson process with a mean rate of 4.5 per day. Although the arriving jobs are of three distinct types, the time required to perform any of these jobs has the same exponential distribution with a mean of 0.2 working days. The practice has been to work on arriving jobs on a first-come, first-served basis. However, it is important that jobs of "type 1" do not have to wait very long, whereas this is only moderately important for jobs of "type 2", and relatively unimportant for jobs of "type 3". These three types arrive with a mean rate of 1.5, 2.0, and 1.0 per day, respectively. Since all three types have been experiencing rather long delays on the average, it has been proposed that the jobs be selected according to an appropriate priority discipline instead.

Compare the expected waiting time (including service) for each of the three types of jobs when the queue discipline is (a) first-come, first-served, and (b) first-come, first-served within priority class.

Answers: For the first-come, first-served queue discipline, expected waiting time is 2 days, independent of job type. When priority distinctions are made, the expected waiting times are 0.457, 1.057, and 6.2 days for jobs of type 1, 2, and 3, respectively.

3.17.5 In Problem 3.17.3, assume that mechanic interarrival times at the two tool cribs follow the distribution shown in Table T3.17.5(a), and that service times at the cribs vary as shown in Table T3.17.5(b). Use a GPSS model to compare and contrast the "separate cribs" approach to the proposed combined operation of the cribs. Assume that the difference in service time distributions at the two cribs is attributable to the nature of the

Interarrival Time, Minutes	Relative Frequency	
	Tool Crib 1	Tool Crib 2
50	.03	
100	.04	
125	.05	
150	.07	.01
160	.09	.02
170	.12	.05
175	.16	.07
180	.13	.11
185	.10	.13
190	.09	.16
200	.07	.18
210	.04	.14
235	.01	.08
260		.03
310		.02

Table T3.17.5(a)

Service Time, Minutes	Relative Frequency	
	Tool Crib 1	Tool Crib 2
80		.03
90		.10
100	.01	.14
110	.05	.16
115	.10	.18
120	.15	.23
125	.19	.12
130	.17	.04
140	.14	
150	.12	
160	.07	

Table T3.17.5(b)

service demands, and not to differences between the two servers. Is the improvement resulting from the combined operation as great as the improvement found in Problem 3.17.3? Would you have expected lesser or greater improvement as a result of the changed distributions?

3.17.6 Modify the model in Case Study 2F under the assumption that television sets arrive at the last inspection station from the preceding station in a Poisson stream. Also assume that the inspection station workers, and the adjustment station worker, are characterized by exponential service times. Assume that the mean inter-arrival time and service times from Case Study 2F are still in effect. Run the modified model, and compare the experienced staging space requirements with those observed in the case study. Does the assumption of Poisson arrivals to exponential servers increase, or decrease, the amount of staging space required?

3.18. GPSS Control Cards: The RMULT Card

Table 3.1 shows that all eight GPSS random number generators start at the same point and produce identical sequences of random numbers. This synchronization among the generators is sometimes undesirable. As a result, GPSS provides the analyst with a control card which makes it possible to change this "default starting point" for any one or more of the eight generators. This card can be used in several different ways.

(1) The starting point of selected generators can be changed from the default value prior to beginning the first simulation.

(2) Between START Cards, the starting point of selected generators can be restored, either to values originally prescribed by the analyst, or to the default values originally assumed by the Processor.

(3) Between START Cards, instead of restoring the previously effective starting points, entirely new starting points can be prescribed.

The Control Card used for these purposes is the RMULT Card. The Location Field is not used on the card. The word RMULT is entered in the Operation Field. There are 8 Operands, A through H, corresponding to random number generators 1 through 8, respectively. To alter the starting point for a given generator, it is only necessary for the analyst to supply a positive, odd integer in the corresponding Operand position on the RMULT Card. When 1 is supplied as the integer, the starting point for the corresponding generator is re-set to its default value. When any odd integer other than 1 is supplied, this value determines some non-standard starting point at which the corresponding generator is set.

Figure 3.25 provides several examples for the RMULT Card. The first example

```
LOCATION | OPERATION | A,B,C,D,E,F ------>
1 2 3 4 5 6 | 7 | 8 9 10 11 12 13 14 15 16 17 18 | 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
                RMULT          1,,1,,1,,1,,1,,1,,1,,1                FIRST EXAMPLE
                RMULT          ,,1,,1,,,,,,1                         SECOND EXAMPLE
                RMULT          453,,721,,,555                        THIRD EXAMPLE
```

Figure 3.25 Examples of RMULT Card Use

shows an RMULT Card which will cause all 8 generators to be restored to their default starting point. In the second example, only generators 2, 3, and 6 are restored to their default points; the other generators are left undisturbed. The third example shows how generators 1, 2, and 4 can have their starting points set to values differing from the default cases.

The term "MULT" arises in connection with the RMULT Card because the Control Card Operands are literally taken to be the values to which multipliers used in the random number generation scheme are to be set. When a call is made on a generator, the value of the corresponding multiplier is changed as part of the algorithm followed by the Processor to form the next random number.[3h] Re-setting the multiplier to a previously effective value guarantees that the sequence of previously-produced random numbers will be duplicated in subsequent calls on the generator. The default multiplier values are

---

[3h] The algorithm used by the Processor to generate 0-1 uniform random numbers will not be explained here. The interested reader can find a detailed discussion of the algorithm in any IBM Users Manual for the language.

initially 1; this explains why restoring a given multiplier to a value of 1 with the RMULT Card causes the "default starting point" to come into effect again. It also explains why a generator's starting point can be controlled by the analyst through his choice of the initial multiplier value, as supplied via the RMULT Card.

As a last point, consider where the RMULT Card should be placed in a job deck. If the analyst wants to choose a non-standard sequence of random numbers at the outset of a simulation, the RMULT Card should appear ahead of the first GENERATE Block in the model. This assures that the generator multipliers are set before the corresponding generators are referenced by the Processor during the input phase.

Similarly, if certain random number sequences are to be modified with the RMULT Card between simulations, the RMULT Card should be used ahead of the CLEAR Card. The reason for this is simple. It was explained in Chapter 2 that, as part of the clearing operation, the Processor arranges for the first Transaction arrivals at GENERATE Blocks in the model. If the corresponding inter-arrival times are to depend on one or more newly-modified random number sequences, then the modifications must precede the clearing.

3.19. Replication of Experimental Conditions in Simulation Modeling

Experimental work in any field involves controlling as closely as possible the conditions under which experiments are performed. By deliberately changing the value of only one variable (the "independent variable"), observed changes in another variable (the "dependent variable") can be attributed exclusively to a single cause. For example, assume that variables named A, B, C, D, and E are subject to experimental control, and that A depends only on B, C, D, and E. To measure the dependence of A on B alone, the values of C, D, and E are held constant, then B is deliberately varied and the resulting changes in A are observed. Under the conditions described, the changes in A are due to the changes in B. In contrast, if the values of B, C, D, and E were varied simultaneousl it would be difficult and perhaps impossible to isolate B's influence on A.

These simple remarks also apply to experiments conducted with simulation models. Simulation is frequently used to compare alternative system configurations to find which one best satisfies a given objective. In this sense, the way a system is configured can be thought of as an "independent variable". By changing system configurations, the analyst is essentially changing the value of an independent variable. His goal is to observe the corresponding changes in some measure of system effectiveness. If changes in effectiveness are to be attributed exclusively to configuration changes, all other conditions under which the alternatives are investigated should be identical from experiment to experiment.

Now, each source of randomness in a system has a potential influence on the measure of system behavior. Important sources of randomness in GPSS models are inter-arrival times and holding times. The analyst might think that if the A and B Operands at a given GENERATE or ADVANCE Block are the same when alternative configurations are studied, this "experimental condition" is being "controlled". This is not true. Not only is it important for pertinent inter-arrival and service time distributions to be the same for each configuration studied, but the individual values and the sequence in

which they are drawn should ideally be duplicated for the alternative configurations tested. Otherwise, short-run patterns might influence the results in such a way that misleading conclusions are drawn.

As a case in point, consider the tool crib system modeled in Case Study 2C. The tool crib was manned by a single clerk. Mechanics making tool requests in Category 1 arrived every 420±360 seconds, each requiring 300±90 seconds of the clerk's time. Category 2 requests for tools, each requiring 100±30 seconds of the clerk's time, were made by mechanics arriving every 360±240 seconds. The measure of system effectiveness was "average time mechanics spend waiting for the clerk". The "independent variable" was the queue discipline used by the clerk. It was of interest to determine whether giving Category 2 mechanics a higher priority resulted in lower overall average mechanic waiting times, thereby reducing costs. In Case Study 2C, operation of the tool crib was simulated for an 8-hour day under both alternatives, and it was tentatively concluded that costs were significantly reduced by using a priority distinction.

For the tool crib problem, average mechanic waiting time depends on these five factors.

(1) Queue discipline in effect.

(2) The particular inter-arrival time sequence of Category 1 mechanics.

(3) The particular sequence of service times required by Category 1 mechanics.

(4) The particular inter-arrival time sequence of Category 2 mechanics.

(5) The particular sequence of service times required by Category 2 mechanics.

The dependence of waiting time on five factors makes it difficult to interpret the results if the intention is to isolate the influence of queue discipline. If waiting time differences are to be attributed solely to differences in queue discipline, then the other conditions under which the alternatives are compared should be identical.

At least three ways suggest themselves for comparing alternatives under otherwise identical conditions.

(1) Devise some means for analytically modifying the simulation output to take the masking effect of "non-identical randomness" into account.

(2) Make simulation runs of such extended duration that short-run patterns cancel each other out. In Case Study 2C, for example, the random variable "average waiting time" has an expected value which depends only on the queue discipline in effect. As the duration of a simulation increases, the model's outputted estimate of this expected value tends toward the theoretical expectation.

(3) Design the simulation model so that each alternative is investigated under identical sets of conditions. In Case Study 2C, this means arranging for items (2) through (5) above to follow identical random patterns when the queue discipline is changed. This approach, then, involves "duplicating randomness" in the model by controlling the experimental conditions under which the simulation is run.

Method (3) admittedly does not eliminate the possibility that short-run patterns will produce misleading results for small sample sizes. It does sharpen the contrast among alternatives, however, and makes possible relatively strong statements of the "other things being equal" type. It is superior to method (1), because it is less demanding to duplicate randomness in a model than to take masking effects into account analytically

on a post-simulation basis. Its advantage over method (2) is that the latter may involve inordinate amounts of computer time, especially when models are large and there are many alternatives to investigate. Of course, methods (2) and (3) are not of the "either-or" type. Even when long simulation runs can be justified, it is advantageous to duplicate sources of randomness among the various alternatives.

Consider, then, how randomness can be duplicated for the two configurations studied in Case Study 2C. For convenience, the extended program listing for the model with priority distinctions, as shown in Figure 2C.4, is repeated in Figure 3.26. For the model as shown, the GENERATE Blocks at Locations 1 and 8 and the ADVANCE Blocks at Locations 5 and 12 all use RN1 as their source of uniform 0-1 random numbers. (No random number source is used at the GENERATE Block in Location 15, because a deterministic time of arrival is specified there.) In other words, the four sources of randomness "share" a common stream of random numbers. The manner in which this sharing takes place is suggested in Table 3.14, where the first 25 calls on RN1 are listed in chronological order, and the source of the call (i.e., Location from which the call is made) is shown both for the Figure 3.26 model ("Model 1"), and for the equivalent model with priority distinctions eliminated ("Model 2"). For the first 16 calls on RN1, the Block Locations from which the corresponding calls are made match, model-by-model. Then, as indicated in Table 3.14, the 17th call comes from the ADVANCE Block in Location 12 in Model 1, and from the ADVANCE Block in Location 5 in Model 2. The 17th call is made at simulated time 1457 (as determined from information not shown in Table 3.14). In terms of the system itself, this means that at time 1457, when the tool crib clerk completed a service, he found at least one Category 1 and one Category 2 mechanic waiting. Furthermore, the Category 1 mechanic had been waiting the longer of the two. In Model 2, the Category 1 mechanic consequently was served next. In Model 1, however, the Category 2 mechanic was given preference.

From the 17th call forward, the particular RN1 values used at Blocks 1, 5, 8, and 12 in Model 1 differ considerably from those used in Model 2. As pointed out in Table 3.14, for example, it is Block 5 in Model 1 which uses the 20th value supplied by RN1, whereas Block 8 in Model 2 uses the 20th RN1 value. And so on. In fact, it is clear that when the call synchronization is broken on the 17th call, the role played by any particular RN1 value will often differ between Models 1 and 2. With respect to calls 18 through 25 on RN1, Table 3.14 shows that the context in which the RN1 value is used differs in 50% of the cases. Information not shown in Table 3.14 indicates that, throughout the course of an 8-hour day, 282 and 289 calls on RN1 are made in Models 1 and 2, respectively. From call 1 through 282, the RN1 values are used in different contexts 55% of the time. It should be clear, then, that among the five factors on which average waiting time depends, no two factors are the same in both Model 1 and Model 2. The experimental investigation has not been performed under controlled conditions.

It is relatively easy to introduce controlled conditions in Case Study 2C. This simply involves taking steps to insure that each of the four sources of randomness uses its own "private" 0-1 random number generator. Suppose, then, that Functions are defined to sample from the uniform distributions at Blocks 1, 5, 8, and 12, and that

```
BLOCK                                                                  CARD
NUMBER   *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS             NUMBER
                SIMULATE                                                 1
         *                                                               2
         *      MODEL SEGMENT 1                                          3
         *                                                               4
1                GENERATE   420,360,,,1       CATEGORY 1 MECHANICS ARRIVE  5
2                QUEUE      LINE              ENTER "CATEGORY 1 SEGMENT" OF LINE  6
3                SEIZE      CLERK             CAPTURE THE CLERK            7
4                DEPART     LINE              LEAVE THE LINE               8
5                ADVANCE    300,90            USE THE CLERK                9
6                RELEASE    CLERK             FREE THE CLERK              10
7                TERMINATE                     LEAVE THE TOOL CRIB AREA    11
         *                                                              12
         *      MODEL SEGMENT 2                                         13
         *                                                              14
8                GENERATE   360,240,,,2       CATEGORY 2 MECHANICS ARRIVE 15
9                QUEUE      LINE              ENTER "CATEGORY 2 SEGMENT" OF LINE  16
10               SEIZE      CLERK             CAPTURE THE CLERK           17
11               DEPART     LINE              LEAVE THE LINE              18
12               ADVANCE    100,30            USE THE CLERK               19
13               RELEASE    CLERK             FREE THE CLERK              20
14               TERMINATE                     LEAVE THE TOOL CRIB AREA    21
         *                                                              22
         *      MODEL SEGMENT 3                                         23
         *                                                              24
15               GENERATE   28800             TIMER ARRIVES AFTER 8 HOURS 25
16               TERMINATE  1                 SHUT OFF THE RUN            26
         *                                                              27
         *      CONTROL CARDS                                           28
         *                                                              29
                START      1                 START THE RUN               30
                END        RETURN CONTROL TO OPERATING SYSTEM            31
```

Figure 3.26 A Repetition of Figure 2C.4

Number of the Call	Block Location from Which Call is Made: Model 1: with PR Distinctions	Model 2: without PR Distinctions
1	1	1
2	8	8
3	1	1
4	5	5
5	8	8
6	12	12
7	1	1
8	5	5
9	8	8
10	8	8
11	12	12
12	12	12
13	1	1
14	5	5
15	1	1
16	8	8
17	12	5
18	8	8
19	12	12
20	5	8
21	8	1
22	1	12
23	12	12
24	5	5
25	1	8

Calls With Differing Sources

Table 3.14 Source of First 25 Calls on RN1 in the Two Alternative Models for Case Study 2C

random number generators 1, 2, 3, and 4, respectively, are specified as the Function arguments. Figure 3.27 shows the extended source listing for the priority-distinction model in which this has been done. The Functions IAT1, STYM1, IAT2, and STYM2 are used at Blocks 1, 5, 8, and 12, in that order.

```
BLOCK                                                                    CARD
NUMBER  *LOC   OPERATION  A,B,C,D,E,F,G              COMMENTS           NUMBER
               SIMULATE                                                    1
        *                                                                  2
        *      SET NON-STANDARD RANDOM NUMBER SEQUENCES                    3
        *                                                                  4
               RMULT      511,39,7,663                                     5
        *                                                                  6
        *      FUNCTION DEFINITIONS                                        7
        *                                                                  8
         IAT1  FUNCTION   RN1,C2      CATEGORY 1 MECHANIC INTER-ARRIVAL TIMES  9
        0,60/1,781                                                        10
         IAT2  FUNCTION   RN3,C2      CATEGORY 2 MECHANIC INTER-ARRIVAL TIMES 11
        0,120/1,601                                                       12
         STYM1 FUNCTION   RN2,C2      CATEGORY 1 MECHANIC SERVICE TIMES       13
        0,210/1,391                                                       14
         STYM2 FUNCTION   RN4,C2      CATEGORY 2 MECHANIC SERVICE TIMES       15
        0,70/1,131                                                        16
        *                                                                 17
        *      MODEL SEGMENT 1                                            18
        *                                                                 19
1              GENERATE   FN$IAT1,,,,1    CATEGORY 1 MECHANICS ARRIVE     20
2              QUEUE      LINE            ENTER "CATEGORY 1 SEGMENT" OF LINE 21
3              SEIZE      CLERK           CAPTURE THE CLERK               22
4              DEPART     LINE            LEAVE THE LINE                  23
5              ADVANCE    FN$STYM1        USE THE CLERK                   24
6              RELEASE    CLERK           FREE THE CLERK                  25
7              TERMINATE                   LEAVE THE TOOL CRIB AREA        26
        *                                                                 27
        *      MODEL SEGMENT 2                                            28
        *                                                                 29
8              GENERATE   FN$IAT2,,,,2    CATEGORY 2 MECHANICS ARRIVE     30
9              QUEUE      LINE            ENTER "CATEGORY 2 SEGMENT" OF LINE 31
10             SEIZE      CLERK           CAPTURE THE CLERK               32
11             DEPART     LINE            LEAVE THE LINE                  33
12             ADVANCE    FN$STYM2        USE THE CLERK                   34
13             RELEASE    CLERK           FREE THE CLERK                  35
14             TERMINATE                   LEAVE THE TOOL CRIB AREA        36
        *                                                                 37
        *      MODEL SEGMENT 3                                            38
        *                                                                 39
15             GENERATE   28800           TIMER ARRIVES AFTER 8 HOURS     40
16             TERMINATE  1               SHUT OFF THE RUN                41
        *                                                                 42
        *      CONTROL CARDS                                              43
        *                                                                 44
               START      1               START THE RUN                   45
               END        RETURN CONTROL TO OPERATING SYSTEM              46
```

Figure 3.27 Extended Program Listing for Case Study 2C with Controlled Experimental Conditions and a Priority Distinction

In Figure 3.27, Card 5 is an RMULT Card. The card is used to set generators 1 through 4 at non-standard starting points which differ from each other. If the default starting points were used, or if the non-standard starting points were identical, the model would be invalid. The reason for this is that inter-arrival times and service times would be strongly related to each other; this relationship, in turn, would be unrealistic in terms of the real system.

Consider this matter of a correlation between the inter-arrival and service times more closely. Assume all four generators used by the four Figure 3.27 Functions were set at a common starting point. Then the same random number value is used to determine both a mechanic's inter-arrival time, and his service time. For example, the first call on RN1 is made to determine arrival time of the first low-priority mechanic. Similarly, the first call on RN2 is made to compute service time of the first low-priority mechanic. But under the hypothesis of identical starting points, RN1 and RN2 return identical values on the first call. The result is one of correlation between the inter-arrival time and service time of the first low-priority mechanic. The same remark holds for each low-priority mechanic entering the model, and for each high-priority mechanic as well. Mechanics with small inter-arrival times (because of small RN1 or RN3 values) have small service times (because of the identical RN2 or RN4 values used to compute their service time). Similarly, mechanics with large inter-arrival times have correspondingly large service times. This unacceptable "synchronization" among the four generators is broken simply by setting their starting points at different values.

Queue statistics from the Figure 3.27 model, and its no-priority-distinction equivalent, is shown in Figure 3.28. Average queue contents with and without priority

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS
LINE	6	1.824	148	21	14.1	354.945	413.637

(a) With Priority Distinctions

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS
LINE	8	2.609	148	21	14.1	507.743	591.700

(b) Without Priority Distinctions

Figure 3.28 Selected Output from the Figure 3.27 Model and its No-Priority-Distinction Equivalent

distinctions is 1.824 and 2.609, as compared with 0.770 and 2.731, respectively, in Case Study 2C. The resulting cost savings is $56.52, as compared with $141.19. What is important here is not these particular numeric differences, but the ability to say for the Figure 3.27 model and its no-priority-distinction equivalent that "on a given day, if priority distinctions had been made, a savings of thus-and-such would have been realized as compared with making no priority distinctions on the same day."

Of course, the same number of mechanic arrivals was experienced in each of the two simulations, because the inter-arrival time sequences for Category 1 and Category 2 mechanics are in one-to-one correspondence between the two models. And, in both models, the tool crib clerk left 5 mechanics standing in line at the end of the day. In the real system, the clerk probably would not accept additional arrivals within 5 minutes of the end of the day, and/or would remain on the job beyond quitting time, if necessary, until the last service had been completed. The way such close-up conditions can be modeled in GPSS will be explained later in the book.

There is a large discrepancy between the results produced in Case Study 2C, and those resulting from the Figure 3.27 model. Despite the "controlled conditions" associated with the latter output, then, the question still remains, just how much better is the priority-distinction system likely to be than the alternative in the long run. The case study in the next section undertakes to answer this question.

3.20. Case Study 3C: A Second Tour Through Case Study 2C

(1) Statement of the Problem

Modify the model shown in Figure 3.27 so that, in a single batch run, ten different days of operation will be simulated for the tool crib problem originally described in Case Study 2C. In addition to controlling experimental conditions, arrange the model so that both the "no priority" and "priority" queue disciplines are investigated in the single run. The resulting output will provide two samples, each consisting of ten different "average Queue content" estimates for the corresponding disciplines. Compute the mean and standard deviation for each of these two samples. How much better is the priority-distinction discipline based on these samples of size 10?

(5) [3k] Extended Program Listing

The extended program listing appears in Figure 3C.1 on the next page. For compactness, Cards 52 through 69 and 81 through 101 have been eliminated from the figure.

(7) Discussion

Model Implementation

Inspection of Figure 3C.1 shows how the RMULT-CLEAR-START sequence has been used to arrange for the ten different days of operation requested in the problem statement. The "no priority" discipline is run first; then, through redefinition of the appropriate GENERATE Blocks (Cards 73 and 74), the priority discipline is introduced. A Control Card sequence then follows which is identical to the one used with the no-priority configuration.

It is important that Cards 73, 74, 75, and 76 be in the order shown. In particular, if Card 75 were placed ahead of Cards 73 and 74, the first day's simulation for the priority configuration would not be identical to that for the no-priority case. There is a simple explanation for this. When the Processor encounters a "new" definition of a GENERATE Block, it performs these steps.

(1) The Future Events Chain Transaction scheduled to be the next arrival at the GENERATE Block is removed from that chain and returned to the latent pool.

(2) Based on the "new" GENERATE Block Operands in effect, the Processor then pre-schedules the first arrival at the re-defined GENERATE Block.

Step (2) requires drawing the next value from the corresponding random number generator. If the generator's starting point has been re-set (via earlier appearance of the RMULT Card), the first value in the re-set sequence is called and used. Then, when the clearing operation later takes place (Card Number 76), all Transactions pre-scheduled to move into GENERATE Blocks are removed from the Future Events Chain, returned to the latent pool, and replaced with freshly-determined pre-scheduled arrivals. This, in turn, means that the second draw from the re-set random number sequence would determine the time of the first arrival to the GENERATE Block in question. But in day 1 for the no-priority case, it was the first draw which determined the first arrival to the GENERATE Block. Hence, an inconsistency arises. The problem is avoided simply by re-setting the generators after the GENERATE Blocks have been re-defined, but before the clearing operation occurs. In the re-definitions of the GENERATE Blocks, then, the old random number sequences supply the needed random numbers, instead of the new sequences.

[3k] Sections (2), (3), (4), and (6) are not needed in this case study, because the information they would contain has been presented earlier.

(5) Extended Program Listing

```
BLOCK                                                                    CARD
NUMBER  *LOC   OPERATION  A,B,C,D,E,F,G            COMMENTS              NUMBER
               SIMULATE                                                     1
        *                                                                   2
        *      SET 1ST NON-STANDARD RANDOM NUMBER SEQUENCES                 3
        *                                                                   4
               RMULT      511,39,7,663                                      5
        *                                                                   6
        *      FUNCTION DEFINITIONS                                         7
        *                                                                   8
         IAT1  FUNCTION   RN1,C2      CATEGORY 1 MECHANIC INTER-ARRIVAL TIMES 9
        0,60/1,781                                                          10
         IAT2  FUNCTION   RN3,C2      CATEGORY 2 MECHANIC INTER-ARRIVAL TIMES 11
        0,120/1,601                                                         12
         STYM1 FUNCTION   RN2,C2      CATEGORY 1 MECHANIC SERVICE TIMES     13
        0,210/1,391                                                         14
         STYM2 FUNCTION   RN4,C2      CATEGORY 2 MECHANIC SERVICE TIMES     15
        0,70/1,131                                                          16
        *                                                                   17
        *      MODEL SEGMENT 1                                              18
        *                                                                   19
1        TAG1  GENERATE   FN$IAT1     CATEGORY 1 MECHANICS ARRIVE           20
2              QUEUE      LINE        ENTER BACK OF LINE                    21
3              SEIZE      CLERK       CAPTURE THE CLERK                     22
4              DEPART     LINE        LEAVE THE LINE                        23
5              ADVANCE    FN$STYM1    USE THE CLERK                         24
6              RELEASE    CLERK       FREE THE CLERK                        25
7              TERMINATE               LEAVE THE TOOL CRIB AREA              26
        *                                                                   27
        *      MODEL SEGMENT 2                                              28
        *                                                                   29
8        TAG2  GENERATE   FN$IAT2     CATEGORY 2 MECHANICS ARRIVE           30
9              QUEUE      LINE        ENTER BACK OF LINE                    31
10             SEIZE      CLERK       CAPTURE THE CLERK                     32
11             DEPART     LINE        LEAVE THE LINE                        33
12             ADVANCE    FN$STYM2    USE THE CLERK                         34
13             RELEASE    CLERK       FREE THE CLERK                        35
14             TERMINATE               LEAVE THE TOOL CRIB AREA              36
        *                                                                   37
        *      MODEL SEGMENT 3                                              38
        *                                                                   39
15             GENERATE   28800        TIMER ARRIVES AFTER 8 HOURS           40
16             TERMINATE  1            SHUT OFF THE RUN                      41
        *                                                                   42
        *      CONTROL CARDS AND BLOCK RE-DEFINITIONS                       43
        *                                                                   44
               START      1            START RUN FOR 1ST SETTING OF RANDOM NUMBER 45
               RMULT      741,211,483,659      SET 2ND RANDOM NUMBER SEQUENCES 46
               CLEAR                    CLEAR  FOR 2ND RUN                 47
               START      1            START RUN FOR 2ND SETTING OF RANDOM NUMBER 48
               RMULT      111,157,539,211      SET 3RD RANDOM NUMBER SEQUENCES 49
               CLEAR                    CLEAR FOR 3RD RUN                  50
               START      1            START RUN FOR 3RD SETTING OF RANDOM NUMBER 51
                 .                                                           .
                 .                                                           .
                 .                                                           .
               RMULT      41,527,9,55          SET 10TH RANDOM NUMBER SEQUENCES 70
               CLEAR                    CLEAR FOR 10TH RUN                 71
               START      1            START RUN FOR 10TH SETTING OF RANDOM NUMBE 72
1        TAG1  GENERATE   FN$IAT1,,,,1     RE-CONFIGURE FOR PRIORITY DISTINCTION 73
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
8        TAG2  GENERATE   FN$IAT2,,,,2     RE-CONFIGURE FOR PRIORITY DISTINCTION 74
MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD
               RMULT      511,39,7,663         RESTORE 1ST SETTING OF RANDOM NU 75
               CLEAR                    CLEAR FOR 11TH RUN                 76
               START      1            START RUN FOR 1ST SETTING OF RANDOM NUMBER 77
               RMULT      741,211,483,659      RESTORE 2ND SETTING OF RANDOM NU 78
               CLEAR                    CLEAR FOR 12TH RUN                 79
               START      1            START RUN FOR 2ND SETTING OF RANDOM NUMBER 80
                 .                                                           .
                 .                                                           .
                 .                                                           .
               RMULT      41,527,9,55          RESTORE 10TH SETTING OF RANDOM N 102
               CLEAR                    CLEAR FOR 20TH RUN                 103
               START      1            START RUN FOR 10TH SETTING OF RANDOM NUMBE 104
               END        RETURN CONTROL TO OPERATING SYSTEM                 105
```

Figure 3C.1 Extended Program Listing for Case Study 3C

Program Output

Table 3C.1 summarizes the information produced when the Figure 3C.1 model was run. The statistic "number of Queue entries" shows relatively little variation on a day-by-day basis. The extreme values, 141 and 154, each differ from the average value by about 4.5% The situation is quite different with the statistic, "average queue content". For the "no-priority" case, the minimum and maximum values, 1.129 and 3.215, differ from the mean value by about 53% and 35%, respectively. The sample standard deviation, 0.833, is 35% of the sample mean. The same observations hold for the priority-distinction case. The large day-to-day variation possible in the Queue-contents statistic explains why the results for the one-day simulation in Case Study 2C, and the "controlled conditions" simulation in Section 3.18, can differ from each other to such an extent. Based on the average values in Table 3C.1, waiting costs for the "no-priority" and "priority"

Day	Number of Queue Entries	Average Queue Contents Without Distinctions	With Distinctions
1	148	2.609	1.824
2	150	3.946	2.754
3	146	2.296	1.688
4	145	1.839	1.350
5	152	3.215	2.317
6	142	1.129	0.906
7	147	2.516	1.826
8	141	1.218	0.945
9	148	2.082	1.516
10	154	3.030	2.174
Average:	147.3	2.388	1.723
Standard Deviation:	3.9	0.833	0.556

Table 3C.1 Output Summary for Case Study 3C

disciplines are about $172 and $124 per day, respectively. The daily savings for the priority-distinction case, then, is estimated at $48 from these data.

3.21 Exercises

3.21.1 In Case Study 3A, the Functions ASSEM and FIRE both use RN1 as their argument.

(a) What subset of the calls on RN1 is used to sample from the assembly-time distribution?

(b) What subset of the calls on RN1 is used to sample from the firing-time distribution?

(c) Critically discuss this statement: "If RN1 and RN2 had been used as the arguments for the Functions ASSEM and FIRE, respectively, the Case Study 3A model would be invalid."

3.21.2 Problem 3.17.3 involves a comparison of two alternative system configurations. In the spirit of Case Study 3C, show how to build appropriate GPSS models for Problem 3.17.3 in which the two alternatives are investigated under identical sets of conditions.

3.21.3 In Problem 3.17.4, two alternative queue disciplines are to be investigated. In the spirit of Case Study 3C, show how the investigation can be conducted with GPSS models which guarantee that the alternatives are compared under identical sets of conditions.

3.21.4 The alternatives under investigation in Case Study 3B are "number of waiting spaces to provide at a car wash". Devise a GPSS model to measure behavior of the system under identical circumstances when the alternatives of 1, 2, and 3 waiting spaces are investigated. By "identical circumstances" is meant that the exact time of car arrivals is to be duplicated from configuration to configuration. In addition, whether it remains for service or not, the j-th arrival is to have the same randomly-determined service time requirement from configuration to configuration, for j = 1,2,3, ... For example, suppose that the first 5 arrivals represent the latent service demands shown in column 2 of Table T3.21.4. The last three columns in the table show the hypothesized

Number of Car	Latent Service Demand, Minutes	Does Car Stay for Service? 1-Space	2-Spaces	3-Spaces
1	8	Yes	Yes	Yes
2	5	Yes	Yes	Yes
3	6	No	Yes	Yes
4	3	No	No	Yes
5	4	Yes	Yes	Yes

Table T3.21.4

behavior of these first 5 arrivals, depending on the configuration under investigation. Because only cars 1, 2, and 5 remain for service at the 1-space facility, the first three service times experienced there are 8, 5, and 4 minutes, in that order. Because the first three cars to arrive remain at the 2 and 3-space facilities, however, the first three service times experienced for each of those configurations are 8, 5, and 6, in that order. In short, because the 1-space Facility could not accommodate the third arrival, it lost not only the "business" but also failed to experience the demand-on-resources represented by that "business". If a given arrival did not represent the same latent service demand for each configuration, it is clear that the investigations would not be conducted under "identical circumstances".

Chapter 4

Intermediate GPSS Modeling Concepts, Part I

4.1 Introduction

When constants are used as data values, the data are said to be directly specified. In Chapter 2, all the Block Operands were constants. Probably the simplest (but also the least powerful) way to supply Block Operands, then, is to specify them directly.

It is also possible to specify data indirectly, by use of "variables" whose values are understood to be the data. In Chapter 3, indirect data specification in GPSS models was introduced. Function references were provided as Block Operands in certain cases, with the understanding that the values of those Functions were to be used as Operand values. In particular, Functions were used to indirectly provide A and/or B Operand values at GENERATE and ADVANCE Blocks in Chapter 3.

Further possibilities for indirect specification of Block Operands in GPSS models will be explored in this chapter. Roughly speaking, the sources of Block Operand values to be introduced fall into two categories, system properties and Transaction properties.

System properties are values which describe the state of the system being modeled. Such quantities as "the content of Queue 5", and "the number of times the Facility CPU has been captured", are typical system properties. Entity attributes such as these are automatically maintained by the GPSS Processor. They are known as Standard Numerical Attributes (SNA's). Their values, which usually vary as a simulation proceeds, are available during a simulation. The values are referenced by use of pre-defined names. When these names are used as Block Operands, the corresponding current values are the Operand values.

Transactions also have numerical properties. One of these properties, Priority Level, has already been introduced, although it has not yet been used as a Block Operand. In addition to having a Priority Level, Transactions also possess Parameters. Each Transaction can have from zero to 100 different Parameters. Parameters are integer-valued.[4a] Their values can be assigned and modified in a model according to logic provided by the analyst when the model is built. These values can then be used to advantage in the role of Block Operands. Finally, Transactions are also endowed with a Mark Time. Mark Time can be thought of as a Transaction's "time of birth", or time of entry into a model. In certain contexts, Mark Time provides valuable information in assessing the relative performance of a system being modeled.

As well as discussing the use of system and Transaction properties in GPSS modeling, this chapter will introduce the Table entity. With this entity, sample values can be gathered and tabulated "automatically" during a simulation. The Table entity provides a convenient and useful method for investigating the properties of random variables with GPSS models.

4.2 Standard Numerical Attributes

During a simulation, the GPSS Processor automatically records and updates certain information about the various entities used in a model. Some of this information is then printed out at the end of the simulation run. Typical of such information are the Block

---

[4a] In GPSS V, real-valued Parameters are also available.

Counts, Facility and Storage utilizations, average Queue residence times, etc., that appear as part of the standard model output. Instead of only being available at the end of a run, however, many of these properties are also available during a run. The course of a simulation can consequently be controlled dynamically by having the model make direc-ture of such properties. For example, the rate at which a server works might realistically depend on the number of people waiting for service. When the length of a waiting line grows, a server might tend to work faster than otherwise. To realistically model such a situation, service time should depend on the current Queue content. If this value is available, it can be used in determining service time when each customer goes into service

Tables 4.1, 4.2, and 4.3 show the various numeric properties of Facilities, Storages and Queues, respectively, which can be referenced during a simulation. The properties listed in the tables are know as Standard Numerical Attributes, or SNA's.[4b]

The name of a Standard Numerical Attribute consists of two parts. The first part is a family name. It identifies both the type of entity involved (e.g., Facilities, or Storages, or Queues) and the particular information of interest (e.g., capture count of a Facility, or utilization of a Storage, or current content of a Queue). The second part of a Standard Numerical Attribute identifies the specific family member involved (e.g., which Facility, or which Storage, or which Queue). In Tables 4.1, 4.2, and 4.3, the family names of the various Standard Numerical Attributes appear in capital letters in the "predefined name" column. Appended to the family name is the number or symbolic name of the specific family member involved. The number is represented in general by "j" in the tables; the symbolic name is represented by "sn".

For example, referring to the second entry in Table 4.1, FCj is the number of times the Facility numbered "j" has been captured during a simulation. In practice, "j" must be the number of a Facility in the model. FC12, for instance, is the "capture count" of Facility 12; FC3 is the capture count of Facility 3, and so on.

Alternatively, referring again to the second Table 4.1 entry, FC$sn is the number of times the Facility symbolically named "sn" has been captured during a simulation. FC$JOE, for instance, is the capture count of the Facility JOE; FC$KATHY is the capture count of the Facility KATHY, etc. As these examples show, a dollar sign ($) must be inserted between the Standard Numerical Attribute's family name, and the symbolic name of the specific family member.

All the Standard Numerical Attributes listed in Tables 4.1, 4.2, and 4.3 are integer-valued. Some of the properties are inherently integer-valued, e.g., capture counts, current Queue contents, Storage entry counts, and so on. Other properties normall include a fractional part in their true representation, e.g., average Storage content, average residence time in a Queue. In GPSS, only the integer part of the "true" values is carried by the Standard Numerical Attributes as their value. In the case of Facility and Storage utilizations, the integer portion of the true utilizations would be zero (or at most 1) if utilizations were expressed as a decimal fraction. As a result, the Standard Numerical Attributes for Facility and Storage utilizations have values expressed in parts per thousand. For example, if a Facility has been in a state of capture 58.2% of the time during a simulation, then its utilization is 0.582. Expressed in parts per thousand, the utilization is 582.

(4b) There are other Standard Numerical Attributes in addition to those discussed in this section. They will be introduced in succeeding parts of the book.

Pre-defined Name (4c)	Value
Fj, or F$sn	0 if the Facility is not currently captured; 1 otherwise
FCj, or FC$sn	Number of times the Facility has been captured
FRj, or FR$sn	Facility utilization, expressed in parts per thousand
FTj, or FT$sn	Integer portion of the average Facility holding time

Table 4.1 Facility Standard Numerical Attributes

Pre-defined Name (4c)	Value
Rj, or R$sn	Remaining Storage capacity
Sj, or S$sn	Current Storage content
SAj, or SA$sn	Integer portion of the average Storage content
SCj, or SC$sn	Storage entry count; each time an ENTER Block referencing the Storage is executed, SCj (or SC$sn) is incremented by the ENTER Block's B Operand; the value is never decremented
SMj, or SM$sn	Maximum Storage content; the maximum value Sj (or S$sn) has attained; not to be confused with the Storage capacity
SRj, or SR$sn	Storage utilization, expressed in parts per thousand
STj, or ST$sn	Integer portion of the average Storage holding time; expressed relative to SCj (or SC$sn)

Table 4.2 Storage Standard Numerical Attributes

Pre-defined Name (4c)	Value
Qj, or Q$sn	Current Queue content
QAj, or QA$sn	Integer portion of the average Queue content
QCj, or QC$sn	Queue entry count; each time a QUEUE Block referencing the Queue is executed, QCj (or QC$sn) is incremented by the QUEUE Block's B Operand; the value is never decremented
QMj, or QM$sn	Maximum Queue content; the maximum value Qj (or Q$sn) has attained
QTj, or QT$sn	Integer portion of the average residence time in the Queue, expressed relative to QCj (or QC$sn); note that QCj (or QC$sn) includes zero Queue entries, that is, entries whose residence time in the Queue was zero
QXj, or QX$sn	Integer portion of the average residence time in the Queue; expressed relative to "QTj minus QZj" (or "QT$sn minus QZ$sn"), thus excluding zero Queue entries
QZj, or QZ$sn	Zero Queue entry count; sum of Queue entries for which the residence time in the Queue was zero

Table 4.3 Queue Standard Numerical Attributes

(4c) "j" is the number of the specific family member in question; or, if the name has been chosen symbolically, "sn" is the symbolic name

The numeric form of certain entity properties depends on whether the results appear in model output, or are used as data within the model. For example, the average utilization of Facilities and Storages appears in decimal form as part of standard model output; nevertheless, the Standard Numerical Attributes FRj (or FR$sn) have as their values integer utilizations measured in parts per thousand. Similarly, average Facility holding time is expressed to the third decimal place in model output, whereas the value of FTj (or FT$sn) is only the integer portion of this "true" average holding time.

Each time a Transaction captures a Facility, it puts exactly "one unit of demand" on it, by the definition of Facility. In contrast, when a Transaction engages a Storage, it may put more than one unit of demand on the Storage, i.e., may increase its content by more than one. When a Transaction moves through the "ENTER 5,3" Block, the content of Storage 5 is increased by 3. Only one Transaction "entered" an ENTER Block, but the current Storage content (Sj, or S$sn) and the Storage entry count (SCj, or SC$sn) were each increased by three. For Storages, then, "content" and "units of demand" are the same. All the Storage SNA values are computed with respect to units of demand. In particular, average Storage holding time (STj, or ST$sn) is expressed per unit of demand placed on the Storage.

The term content has the same significance for Queues as for Storages. When a Transaction enters the "QUEUE 3,2" Block, the content of the Queue is increased by 2. If the Transaction then moves through the "DEPART 3,2" Block at the same clock reading, the zero Queue entry count (QZj, or QZ$sn) is increased by two. This is because the Queue residence time was zero for two content-units, even though only one Transaction was involved. As with Storages, all the Queue SNA values are computed with respect to units of content.

Block Counts can also be referenced in a simulation as Standard Numerical Attributes. The family names for Current and Total Block Counts are W and N, respectively. Hence, W$BLOK1 is the number of Transactions currently at the Block in the Location BLOK1. Or, N$PATHC is the total number of Transactions which have entered the Block in the Location PATHC. The position occupied by Blocks can also be referred to numerically. For example, N17 is the current count at the Block in Location 17. As mentioned earlier, Block Locations are usually named symbolically; as a result, the forms W$sn and N$sn are most frequently used when Block Counts are included in the logic of a model.

The Relative Clock can also be used as "data" in a model. The name of the Relative Clock is C1. The value of the Absolute Clock is not available as a Standard Numerical Attribute. There is, however, a method for determining the Absolute Clock's Value. The method is explained later in this chapter.

Note that Block Counts are inherently interger-valued. Also, because of the use of an "integer clock" in GPSS, C1 is "inherently" integer-valued.

The random number generators, RNj, are Standard Numerical Attributes. The values assumed by the RNj's are context-dependent. There are two possibilities.

(1) When used as Function arguments, the value of RNj is a six-digit decimal fraction drawn from the population uniformly distributed on the interval between 0.000000 and 0.999999, inclusive.

(2) When used in any other context, the value of RNj is a three-digit integer drawn from the population uniformly distributed between 000 and 999, inclusive.

Functions are also Standard Numerical Attributes. The discrete and continuous Functions studied in Chapter 3 are numeric-valued. In general, the "true value" of a

Function includes a fractional part. Whether this true value, or only its integer portion, is used is context-dependent.

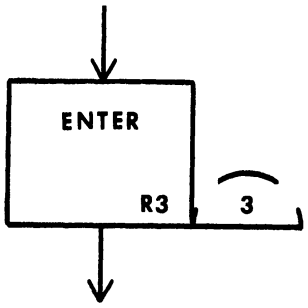
(1) When a Function is employed as the B Operand in a GENERATE or ADVANCE Block, the true value is used. (There are also two other contexts, not yet discussed, in which a Function's true value is used.)

(2) In any other context, only the integer portion of a Function's true value is used.

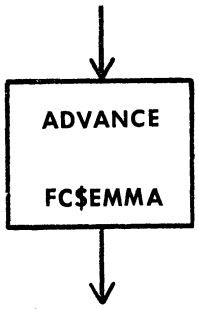
Finally, constants can also be thought of as Standard Numerical Attributes in GPSS. The "family name" of a constant is K. The "specific member" is the constant itself. Hence, K3 is the constant 3; K561 is the constant 561, and so on. When constants are used in GPSS, inclusion of the K is optional. In this book the practice is to always simply write the constant itself, without prefacing it with a K.

4.3 Use of Standard Numerical Attributes

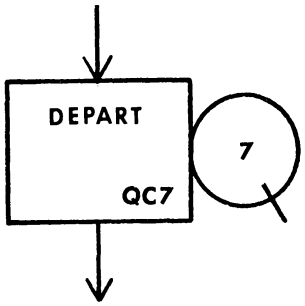
The most obvious application of Standard Numerical Attributes in a model is to use them as Block Operands. Several examples of such use are suggested in Figure 4.1. When a Transaction moves into the ENTER Block in Figure 4.1(a), for instance, it captures "R3"



(a) Use of Rj as a Block Operand



(b) Use of FC$sn as a Block Operand



(c) Use of QCj as a Block Operand

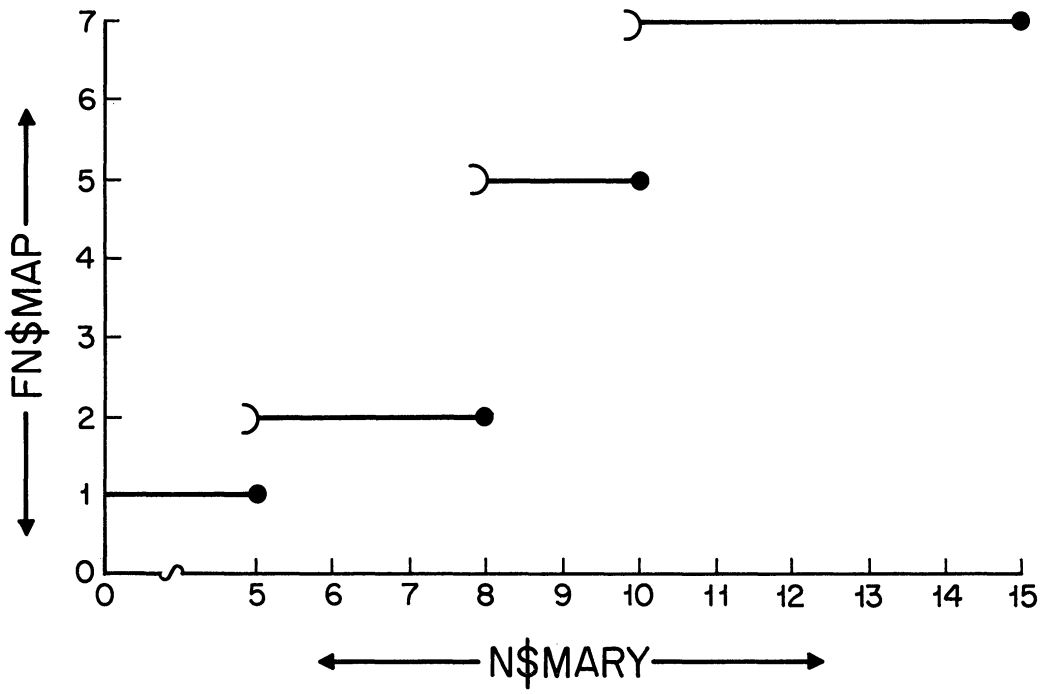
Figure 4.1 Examples of the Use of Standard Numerical Attributes as Block Operands

of the parallel servers simulated with Storage 3. Because "R3" is the number of currently available servers represented by that Storage, the Transaction consequently takes up the remaining capacity of the Storage. Or, when a Transaction moves into the Figure 4.1(b) ADVANCE Block, its holding time there equals the capture count of the Facility EMMA. And so on. These examples are out of context, and consequently do not look "meaningful". They do illustrate, however, how Standard Numerical Attributes can be used as Operands in the various GPSS Blocks.

A less obvious application for Standard Numerical Attributes is to use them as Function arguments. The Functions introduced in Chapter 3 always had an RNj as their argument, because they were to be used in sampling from probability distributions. There is no reason, however, that a Function argument must be an RNj. For example, consider the discrete Function defined in Figure 4.2(a). The Function's argument is the entry count to

```
LOCATION    OPERATION    A,B,C,D,E,F
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
MAP         FUNCTION     N$MARY,D4
5,1/8,2/10,5/15,7
```

(a) Definition of the Function



(b) Graphical Interpretation of the Function

Figure 4.2 Example of a Discrete Function with a Block Count as Its Argument

the Block in Location MARY. When the Function is referenced during a simulation, the current entry count to the Block in that Location will be used as the value of the independent variable. The corresponding Function value will be determined in the usual manner for discrete Functions. The correspondence between argument values and Function values is shown graphically for this example in Figure 4.2(b). Argument values of 5 or less produce a Function value of 1; argument values of 8 or less, but greater than 5, produce a Function value of 2; and so on. Two considerations bear mention here.

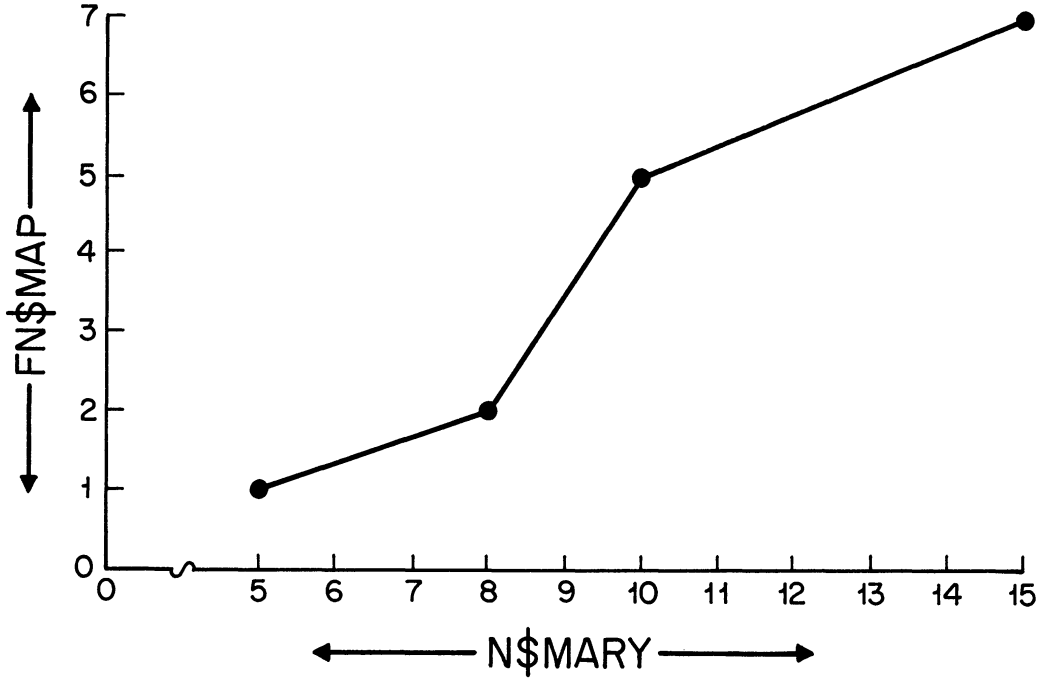
(1) When arguments other than an RNj are used in Function definitions, the argument values that can be realized are restricted to being integers. This follows from the fact that Standard Numerical Attributes are integer-valued in the language.

(2) When Function arguments extend beyond the range of values supplied in the Function definition, the Function value is taken to be that of the nearest point. In the Figure 4.2 example, the range of argument values supplied in the definition is from 5 to 15. As already indicated, for arguments less than 5 (i.e., out of range on the low side), the Function value is 1. For arguments greater than 15 (i.e., out of range on the high side), 7 would be used as the Function's value.

Functions having Standard Numerical Attributes as arguments can be either discrete or continuous. Suppose that the Figure 4.2 Function is to be continuous, rather than discrete. Definitionally, this simply involves changing the "D4" on the FUNCTION Card to "C4", as shown in Figure 4.3(a). Computationally, the effect of having the Function be

```
LOCATION | OPERATION | A,B,C,D,E,F ------>
1 2 3 4 5 6 | 7 | 8 9 10 11 12 13 14 15 16 17 18 | 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
MAP         | FUNCTION | N$MARY,C4
5,1/8,2/10,5/15,7
```

(a) Definition of the Function



(b) Graphical Interpretation of the Function

Figure 4.3 Example of a Continuous Function with a Block Entry Count as Its Argument

continuous was explained in Chapter 3. When the Function is evaluated, the interval in which the argument's value falls is first determined. Then, a linear interpolation is performed across the interval. The Function's "true" value, in general, will then include a fractional part. Whether the true value, or only the integer portion, will subsequently be used depends on the context in which the Function is referenced.

Figure 4.3(b) shows the graphical interpretation of the Figure 4.3(a) Function. Consecutive points have been joined with straight line segments, indicating the linear interpolation in effect. Whenever the Function's argument is out of range, the Function value is taken to be that of the nearest point. This situation is identical to that for discrete Functions. Whether the Figure 4.3(a) Function is defined as discrete or continuous, then, an argument of 15 or greater produces a Function value of 7. Similarly, an argument of 5 or less produces a Function value of 1.

4.4 Case Study 4A: Modeling the Influence of Queue Length on Mean Service Rate

(1) Statement of the Problem

In a one-line, one-server queuing system, arrivals occur in a Poisson pattern with a mean rate of 12 arrivals per hour. Service is performed exponentially, but the mean service time depends on the content of the waiting line ahead of the server. The dependence is shown in Table 4A.1.

Content of the Waiting Line	Mean Service Time, Minutes
0	5.5
1 or 2	5.0
3, 4, or 5	4.5
6 or more	4.0

Table 4A.1 Mean Service Time as a Function of Waiting Line Content

Build a GPSS model of the system, then use the model to estimate the effective mean service time. If the arrival rate increases to the extent of one additional arrival per hour, will the server still be able to handle the traffic, or will the waiting line tend to become longer and longer?

(2) Approach Taken in Building the Model

The Block sequence used to model a one-line, one-server queuing system is already familiar, as is the method for simulating Poisson arrivals and exponential servers. The sole addition to previously-developed ideas in this model is inclusion of a discrete GPSS Function using current Queue content as its argument, thereby making it possible to determine the server's mean service time in terms of waiting line length.

(3) Table of Definitions

Time Unit: 1 Second

GPSS Entity	Interpretation
Transactions	Customers
Facilities	
SURVR	The server
Functions	
MEAN	Function relating mean service time to Queue content as shown in Table 4A.1
XPDIS	Function for sampling from the exponential distribution with a mean value of 1
Queues	
WAIT	The Queue used to maintain statistics on the condition of the waiting line

Table 4A.2 Table of Definitions

(4) Block Diagram

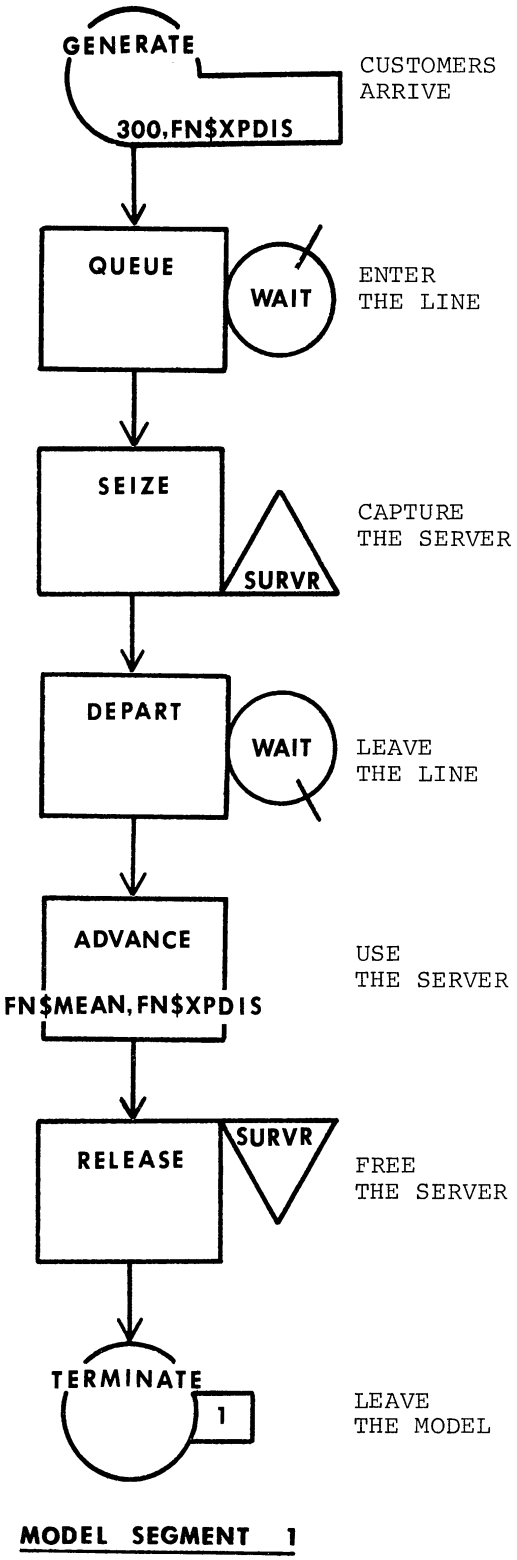


Figure 4A.1 Block Diagram for Case Study 4A

(5) Extended Program Listing

```
BLOCK                                                                    CARD
NUMBER  *LOC    OPERATION  A,B,C,D,E,F,G          COMMENTS               NUMBER
                SIMULATE                                                    1
*                                                                           2
*       DEFINE FUNCTIONS                                                    3
*                                                                           4
 MEAN   FUNCTION   Q$WAIT,D4          MEAN SERVICE TIME DISTRIBUTION        5
0,330/2,300/5,270/6,240                                                     6
 XPDIS  FUNCTION   RN1,C24            EXPONENTIAL DISTRIBUTION FUNCTION     7
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38          8
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2          9
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8                   10
*                                                                          11
*       MODEL SEGMENT 1                                                    12
*                                                                          13
1               GENERATE    300,FN$XPDIS    CUSTOMERS ARRIVE               14
2               QUEUE       WAIT            ENTER THE LINE                 15
3               SEIZE       SURVR           CAPTURE THE SERVER             16
4               DEPART      WAIT            LEAVE THE LINE                 17
5               ADVANCE     FN$MEAN,FN$XPDIS USE THE SERVER                18
6               RELEASE     SURVR           FREE THE SERVER                19
7               TERMINATE   1               LEAVE THE MODEL                20
*                                                                          21
*       CONTROL CARDS                                                      22
*                                                                          23
                START       500,,100        START THE RUN                  24
                END         RETURN CONTROL TO OPERATING SYSTEM             25
```

Figure 4A.2 Extended Program Listing for Case Study 4A

(6) Program Output

No program output is shown directly.

(7) Discussion

Model Implementation

In the extended program listing in Figure 4A.2, note how Q$WAIT has been used as the A Operand in the Function Card used to define the Function MEAN (Card Number 5). Also note that Block 5 is "ADVANCE FN$MEAN,FN$XPDIS". When a Transaction moves into the ADVANCE Block, then, the Processor evaluates the Function MEAN to determine the A Operand value in effect. This value, in turn, depends on the Current Queue Content. Hence, the server is sensitive to the length of the waiting line as he begins his next service.

Inspection of Card 24 shows that the START Card's Snap Interval Option (C Operand) has been used. "START 500,,100" causes statistical printouts to occur just after the 100th, 200th, 300th, 400th, and 500th completion of a service. (The model shuts off with the 500th service completion.) This makes it possible to trace the behavior of statistics such as mean service time as the simulation proceeds.

Program Output

Tables 4A.3 and 4A.4 display selected statistics in effect just after the 100th,

Number of Service Completions	Facility Utilization	Average Holding Time per Capture, Seconds	Average Queue Content	Percent Zero Entries
100	0.996	313	4.6	1.9
200	0.945	296	5.0	6.4
300	0.905	291	5.3	8.7
400	0.899	284	4.5	8.9
500	0.902	292	4.3	9.2

Table 4A.3 Output Summary for the Figure 4A.2 Model

Number of Service Completions	Facility Utilization	Average Holding Time per Capture, Seconds	Average Queue Content	Percent Zero Entries
100	0.999	276	12.4	0.9
200	0.993	277	11.6	0.9
300	0.989	267	10.0	1.3
400	0.955	266	7.8	3.9
500	0.964	270	8.3	3.0

Table 4A.4 Output Summary for the Figure 4A.2 Model with 277 Replacing 300 as the GENERATE Block's A Operand

200th, 300th, 400th, and 500th completion of a service, for arrival rates of 12 and 13 per hour, respectively. The information for 500 service completions in Table 4A.3 indicates that average service time is about 290 seconds, or somewhat under 5 minutes; the server is about 90% utilized; and there are more than 4 customers in the waiting line on average. About 9% of the customers do not have to wait for service.

In Table 4A.4, after 500 service completions the server's utilization is over 96%, and there have been over 8 customers in the waiting line on average. Hence, changing the hourly arrival rate from 12 to 13 (an increase of about 17%) has almost doubled the average queue content. At the same time, the percent zero entries has dropped from 9% to 3%.

Simple inspection of the original data shows that the server can handle the traffic when the arrival rate increases from 12 to 13 per hour. The increased rate corresponds to a mean inter-arrival time of 4.6+ minutes. Table 4A.3 indicates that with 3, 4, or 5 customers in the waiting line, the server's mean service time is 4.5 minutes, which is just insufficient to cope with the traffic. With waiting line content of 6 or more, however, the average service rate drops to 4 minutes, which more than accommodates the traffic. There is no chance, then, that the waiting line will grow without bound as the simulation proceeds.

4.5 Exercises

4.5.1 (a) Of what two parts do the names of Standard Numerical Attributes consist?

(b) What information is conveyed by the first part?

(c) What information is conveyed by the second part?

(d) What character stands between the first and second parts when the second part is symbolic rather than numeric?

(e) In words, what is the value of QZ14? Of SC$BOATS? Of F$ALONE?

(f) What Standard Numerical Attribute has as its value the fractional utilization of the Storage 15? Of the Storage MYRNA? What Standard Numerical Attribute has as its value the current content of Queue 1?

(g) Discuss the difference between the fractional utilization of a Facility as it appears in standard simulation output, and as it is used as the value of a Standard Numerical Attribute.

(h) Explain the difference between QT6 and QX6. Can QX6 ever be greater than QT6? Why or why not?

(i) Explain the differences between Qj, QCj, and QMj.

(j) Explain the difference between W$BYPAS and N$BYPAS. Which one would you always expect to be at least as large as the other?

(k) What is the relationship between RBUS, SBUS, and the user-defined capacity of the Storage named BUS?

(el) What is the value of the B Operand in Figure 4.1(b)?

(m) What will the recorded content of Queue 7 be just after a Transaction enters the DEPART Block in Figure 4.1(c), thereby causing that Block subroutine to be executed?

(n) What Standard Numerical Attribute has the Relative Clock as its value?

(oh) What range of values can RN3 assume when it is used as a Function argument? Do these values include a fractional part?

(p) What range of values can RN3 assume when it is used in any context <u>except</u> as a Function argument? Do these values include a fractional part?

4.5.2 Refer to Figure 2A.5, which shows Facility statistics produced at the end of the Case Study 2A simulation. Using information shown in that figure, indicate the values of the following Standard Numerical Attributes when the simulation shut off.

(a) F$JOE

(b) FC$JOE

(c) FR$JOE

(d) FT$JOE

4.5.3 Refer to Figure 2A.6, which shows Queue statistics produced at the end of the Case Study 2A simulation. Using information shown in that figure, indicate the values of the following Standard Numerical Attributes when the simulation shut off.

(a) Q$JOEQ

(b) QA$JOEQ

(c) QC$JOEQ

(d) QM$JOEQ

(e) QT$JOEQ

(f) QX$JOEQ

(g) QZ$JOEQ

4.5.4 (a) At time 5, Facility 1 is captured. At time 10, it is released. Assume the Facility has not otherwise been captured.

(i) What is F1 at time 4? at time 8? at time 12?

(ii) What is FR1 at time 8? at time 10? at time 12?

(iii) What is FT1 at time 8? at time 10? at time 12?

(b) At times 5 and 10, respectively, the Blocks "ENTER 4" and "LEAVE 4" are executed. Assume Storage 4 has a capacity of 3, and that no other references have been made to it.

(i) What is SC4?

(ii) What is SM4?

(iii) What is SA4 at time 8? at time 12?

(iv) What is SR4 at time 8? at time 12?

(v) What is ST4 at time 8? at time 12?

(c) Repeat (b) when the B Operand on the indicated ENTER and LEAVE Blocks is 2.

(d) The Queue ALPHA had a content of 3 for 5 time units. In addition, it had a content of 2 for 0 time units. Assume no other entries to the Queue have occurred.

(i) What is QC$ALPHA?

(ii) What is QZ$ALPHA?

(iii) What is QT$ALPHA?

(iv) What is QX$ALPHA?

(v) At time 10, what is QA$ALPHA?

(e) The Queue BETA had a content of 3 for 0 time units. Assuming no other entrie to the Queue have occurred, what is the value of QM$BETA?

4.5.5 The Function BOND is defined in Figure P4.5.5

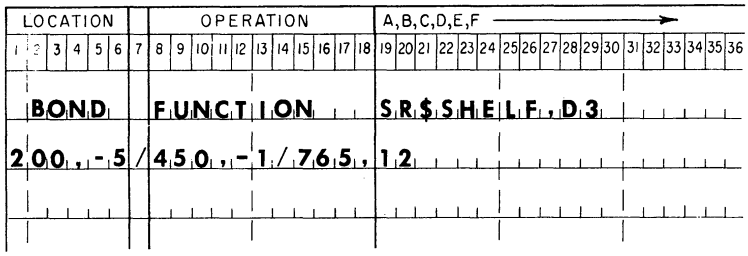


Figure P4.5.5

(a) Show a graphical interpretation of the Function.

(b) What is the Function's value when SR$SHELF is 150? 450? 451? 895?

4.5.6 The Function TOUGH is defined in Figure P4.5.6.

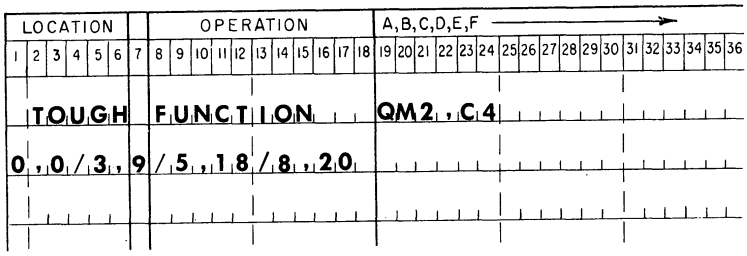


Figure 4.5.6

(a) Show a graphical interpretation of the Function.

(b) What is the Function's true value when QM2 is 0? 1? 3? 4? 6? 10?

4.5.7 Define a Function whose value is to depend on the current count at the Block in Location PATHQ as shown in Table P4.5.7.

Function Value	Current Count
1	0
4	1, 2, or 3
2	4 or 5
4	6
-5	7 or more

Table P4.5.7

Show a graphical interpretation of the Function.

4.5.8 Define a Function whose value is to be twice the current content of the Queue ALPHA, for contents of 0, 1, 2, 3, and 4, and whose value is to be 10 otherwise. Do this two ways: (a) with a discrete, six-point Function, and (b) with a continuous, three-point Function. [In (b), only the integer portion of the Function is to have the values indicated.] Show graphical interpretations for each approach.

4.5.9 Suppose that the Location occupied by the QUEUE Block in Figure 4A.1 is symbolically named HOLD. Discuss how the model in Case Study 4A would be changed, if at all, by using W$HOLD rather than Q$WAIT as the argument of the Function MEAN.

4.5.10 A particular exponential server tends to work at a slower and slower rate as the 8-hour work day goes by. During the first two hours of the day, it takes him 12 minutes on average to perform a service. During the next two hours, his average service time is 15 minutes. During the 5th, 6th, and 7th hours, each service completion requires an average of 17 minutes. Service started during the 8th hour requires an average of 20 minutes for completion. Define a Function whose value is the mean time required by the server to perform a service. Then show how the Function can be used at an ADVANCE Block to simulate service time.

4.6 Transaction Parameters

As indicated in Section 4.1, each Transaction in a GPSS model possesses a set of from zero to 100 Parameters. A Transaction's Parameter set can be conveniently thought of as a collection of Standard Numerical Attributes which the Transaction "owns". As a Transaction moves through a model, its Parameter values can be assigned and modified according to logic provided by the analyst. These values can then be used to advantage as Block Operands, and as Function arguments.

Some of the pertinent features of Transaction Parameters will now be considered.

(1) The number of Parameters a Transaction has is specified via the F Operand at the GENERATE Block through which the Transaction enters a model. The default value of the GENERATE Block F Operand is 12. In all models shown thus far, this default value has been in effect.

(2) As true with most other Standard Numerical Attributes, the name of a Parameter consists of two parts, a family name and indication of a specific family member. The family name is P. Specific family members are designated numerically as 1, 2, 3, 4, ..., 98, 99, 100. Hence, P22 is the name of Parameter 22 of a Transaction, P3 is the name of Parameter 3, and so on. Parameter references are written in general as Pj, where j is some integer between 1 and the number of Parameters a Transaction has. Hence, if a Transaction enters a model through a GENERATE Block with a default value for the F Operand, Pj is meaningful for it when j = 1, 2, 3, ..., 11, or 12, but is meaningless when j = 13, 14, 15, ..., 98, 99, or 100.

Parameters cannot be named symbolically. Hence, references such as P$YEAR, P$COLOR, and so on, are invalid.

(3) The values of Parameters are signed integers. The maximum magnitude of a Parameter depends whether it is of the halfword or fullword type. All of a Transaction's Parameters are either of the halfword or the fullword type. The type is determined by the G Operand of the GENERATE Block. The default value of the G Operand is H (for halfword). If fullword Parameters are required, then F (for fullword) must be used as the G Operand at the corresponding GENERATE Block.

The maximum magnitude of halfword Parameters is 32,767 [i.e., $2^{15}-1$]. For fullword Parameters, the maximum magnitude is 2,147,483,647 [i.e., $2^{31}-1$].

(4) When a Transaction enters a model, the initial value of each of its Parameters is zero.

(5) The meaning of Parameters is determined by the analyst. This is frequently done by using a numeric encoding scheme. At other times, Parameter values have direct significance. Suppose, for example, that a rent-a-car agency is being modeled, and that a Transaction is a car. The characteristics of each car might be represented as Parameter values according to the scheme shown in Table 4.4. Hence, a Transaction having P1, P2, and P3 values of

Value of P1	Significance (color of car)	Value of P2	Significance (make of car)	Value of P3	Significance (model year)
1	Red	1	Volkswagen	1970	1970
2	Blue	2	Pinto	1971	1971
3	Green	3	Vega	1972	1972
4	Grey	4	Gremlin		

Table 4.4 A Possible Interpretation of Parameter Values

3, 1, and 1971, respectively, would represent a green Volkswagen made in 1971. The value of P3 is directly significant. The meaning of the P1 and P2 values is clear only in terms of the given encoding scheme.

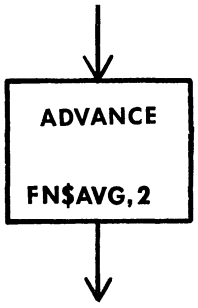
(6) Because Parameters are Standard Numerical Attributes, they can be used as Block Operands and Function arguments.

(7) When a Parameter is used as a Block Operand or Function argument, a question arises as to which Transaction is involved. For example, P6 does not appear to have a unique value. After all, many different Transactions in a model may each have a Parameter numbered 6. Data are only used, though, when Block subroutines are executed. Block subroutines, in turn, are executed only when a Transaction moves through a sequence of Blocks. When Parameter values are required as data, the values are taken from the Parameter set of the "Active Transaction", i.e., the Transaction currently being processed.

For example, suppose that a Function is defined as shown in Figure 4.4(a), and that a Transaction with a Parameter 5 value of 4 enters the ADVANCE Block shown in Figure 4.4(b). Execution of the Block subroutine involves

LOCATION	OPERATION	A,B,C,D,E,F
AVG	FUNCTION	P5,D3
-2,4/3,7/10,8		

(a) Definition of a Function Using P5 as Its Argument



(b) An ADVANCE Block Which Uses the Function as Its A Operand

Figure 4.4 An Example Showing A Parameter Used in Context

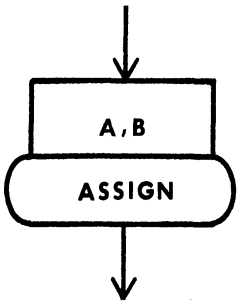
evaluation of the Function AVG. This, in turn, requires finding the value of the Function's argument, P5. The P5 value of the Transaction being processed is 4. The Function's corresponding value is 8. The holding time is therefore uniformly distributed over the interval of integers 8±2. The Processor now samples from this distribution, and the value drawn is used as the holding time.

The example in Figure 4.4 shows how holding time can be made to depend on a Transaction property. Case Study 4A illustrated how a holding time can be made to depend on a property of a Queue. It should be evident that use of system and Transaction properties as typified in these examples provides considerable flexibility in GPSS modeling. After the methods for assigning and modifying Parameter values are introduced in the next section, Case Study 4B is presented to further illustrate this flexibility.

When the Current and/or Future Events Chains are printed out, Parameter values of Transactions on those chains appear as part of the printout. The START Card D Operand will be used in Case Study 4B to force a chain printout at the end of the simulation. The way Parameter values are displayed in the output will then be indicated.

4.7 Modification of Parameter Values: The ASSIGN Block

A Transaction has the value of its Parameters modified when it moves into an ASSIGN Block. This Block, and its A and B Operands, are shown in Figure 4.5.



Operand	Significance	Default Value or Result
A	Number of the Parameter to be modified	Error
B	Data to be used for the modification	Error

Figure 4.5 The ASSIGN Block and Its A and B Operands

When a Transaction enters an ASSIGN Block, the B-Operand data is copied to the Parameter whose number is provided by the A Operand. As a result, the previous value of that Parameter is replaced by a new value. Whenever a Transaction enters the ASSIGN Block shown

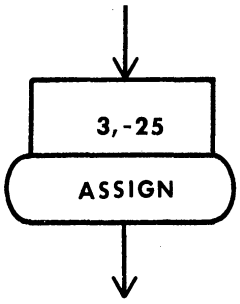


Figure 4.6 A First Example for the ASSIGN Block

in Figure 4.6, for example, the value of its third Parameter becomes -25. The previous value of P3, whatever it might have been, is lost in the process.

In the Figure 4.6 example, the A and B Operands are both supplied directly, as constants. It is also possible to supply one or both of these Operands indirectly, with Standard Numerical Attributes. When a Transaction enters the ASSIGN Block shown in Figure 4.7, for example, the fractional utilization of the Facility BARGE will be copied

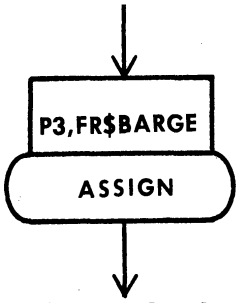


Figure 4.7 A Second Example for the ASSIGN Block

to the Parameter whose number is the value of Parameter 3. If P3 has a value of 5, then the value of FR$BARGE becomes the value of P5; if P3 is 22, the value of FR$BARGE becomes the value of Parameter 22, and so on. Contrast the use of 3 and P3 as the A Operands in Figures 4.6 and 4.7, respectively.

Figures 4.6 and 4.7 illustrate use of the ASSIGN Block in replacement mode. In this mode, the old value of a Parameter is replaced with a new value, without regard to what the old value might have been. It is also possible to use the ASSIGN Block in increment mode, and in decrement mode. In increment mode, the Parameter's new value is computed by adding the B Operand data to the old value. In decrement mode, the Parameter's new value is computed by subtracting the B Operand data from the old value. Increment and decrement mode are specified by placing a plus (+) or minus (-) sign, respectively, ahead of the comma separating the A and B Operands.

Figure 4.8 shows use of the ASSIGN Block in increment mode. When a Transaction

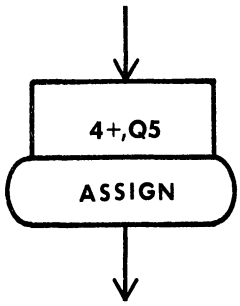


Figure 4.8 A Third Example for the ASSIGN Block

enters the Figure 4.8 ASSIGN Block Parameter 4 will be increased by an amount equal to the current content of Queue 5. In this example, note that the A Operand is directly specified, whereas the B Operand is indirectly specified.

Figure 4.9 shows use of the ASSIGN Block in decrement mode. When a Transaction

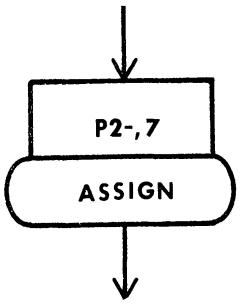


Figure 4.9 A Fourth Example for the ASSIGN Block

enters the Figure 4.9 Block, 7 will be subtracted from the Parameter whose number is in Parameter 2. For example, if P2 is 3, a Transaction entering the Block will have 7 subtracted from Parameter 3. Of, if P2 is 2, 7 will be subtracted from Parameter 2. In Figure 4.9, the A Operand is indirectly specified, and the B Operand is directly specified.

Although not indicated in Figure 4.5, an optional C Operand can be used with the ASSIGN Block. When this is done, the C Operand supplies the number of a Function. The Processor performs these steps in executing the ASSIGN Block subroutine when the C Operand is used.

(1) The C Operand is evaluated.

(2) The Function whose number equals the C-Operand's value is evaluated

(3) The Function's value is combined multiplicatively with the B-Operand data.

(4) The integer portion of the product is used to replace, increment, or decrement the Parameter specified with the A Operand.

For example, assume that a Transaction enters the ASSIGN Block shown in Figure 4.10.

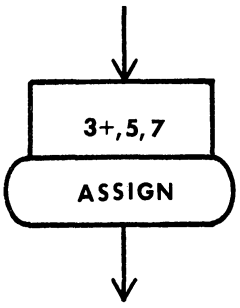


Figure 4.10 An ASSIGN Block Example for the C Operand

The C Operand is directly specified as 7. Function 7 is consequently evaluated. Assume the Function's value is 1.94. This value is then multiplied by the B-Operand data, 5, producing 9.70. The integer portion of this product, 9, is then added to Parameter 3 of the entering Transaction.

The effect in Figure 4.10 would be totally different if FN7, instead of 7, had been used as the C Operand. The Processor would interpret FN7 as indirectly specifying the number of a Function. First, Function 7 would be evaluated. Assume as before the Function 7's value is 1.94. The Processor would take the integer portion, 1, as the number of another Function. It would then evaluate Function 1, multiply the resulting value by 5 (B-Operand data), and add the integer portion of the product to Parameter 3.

It is invalid to supply the symbolic name of a Function as the C Operand at an ASSIGN Block. The analyst is consequently forced to use numeric names in defining Functions which are to be referenced at ASSIGN Blocks via the C-Operand option.(4d)

Observe that the ASSIGN Block B-C Operands stand in the same relationship to each other as the GENERATE and ADVANCE Block A-B Operands. Recall that when FNj or FN$sn is used as the B Operand at a GENERATE or ADVANCE Block, the value of the Function numbered j (or symbolically named sn) is multiplied by the A-Operand data. (The format is admittedly slightly different at the ASSIGN Block, where only j, not FNj, would be used as the C Operand.)

As pointed out in Sections 3.13 and 3.15, the only significant use of Function modification at GENERATE and ADVANCE Blocks is to simulate Poisson arrivals and exponential servers, respectively. Similarly, the only important use of the C Operand at an ASSIGN Block is in sampling from the exponential distribution. For example, assume a particular exponential server requires 50 times units on average to complete a service, and that 5 is the number of the usual 24-point continuous GPSS Function for sampling from the exponential distribution. When a Transaction enters the ASSIGN Block in Figure 4.11, a sample is drawn from the service time distribution and stored in Parameter 9.

(4d) Actually, the analyst also has recourse to two alternative actions. First, he can name the pertinent Function symbolically, then predict the corresponding number the Processor will give it. That number can then be used at the ASSIGN Block. Alternatively, by use of the EQU Control Card, the analyst can force the Processor to make a symbolic name correspond to a given number, with the number then being used at the ASSIGN Block. Further details are given in Section 4.14.

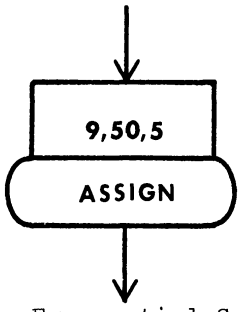


Figure 4.11 Pre-Determining an Exponential Service Time at an ASSIGN Block

Suppose that the Transaction eventually is to capture the exponential server in question. Holding time can then be simulated by moving the Transaction into the Block "ADVANCE P9". Service time required by the Transaction, although a random variable, would have been pre-determined at the ASSIGN Block. This approach is of interest when it is important to control experimental conditions while simulating exponential servers. Case Study 4C in Section 4.12 makes use of such an approach.

4.8 Case Study 4B: A Grocery Store Model

(1) Statement of the Problem

A small grocery store consists of three aisles and a single checkout counter. Shoppers arrive at the store in a Poisson pattern with a mean interarrival time of 75 seconds. After arriving, each customer takes a basket and may then go down one or more of the three aisles, selecting items for purchase as he proceeds. The probability of going down any particular aisle is shown in Table 4B.1. The time required to shop an aisle, and

Aisle	Probability of Going Down the Aisle	Time Required to Travel Down the Aisle, Seconds	Number of Items Selected During Travel
1	0.75	120±60	3±1
2	0.55	150±30	4±1
3	0.82	120±45	5±1

Table 4B.1 Characteristics of Shoppers in the Grocery Store

the number of items selected for purchase in the process, are uniformly distributed random variables. Information describing these variables, which depend on the aisle in question, is also shown in Table 4B.1.

When shopping is complete, the customers queue up first-come, first-served at the checkout counter. In this area, each customer chooses an additional 2±1 "impulse items" for purchase. A customer's checkout time depends on the number of items he is buying. Checkout time is 3 seconds per item. After checking out, a customer leaves his basket at the front of the store and departs.

Build a GPSS model for the grocery store, then use the model to simulate an 8-hour day. Measure the utilization of the checkout girl, and the maximum length of the waiting line at the checkout counter. Assuming there is no limit to the number of shopping baskets, also determine the maximum number of baskets in use at any one time.

(2) Approach Taken in Building the Model

When customer-Transactions are brought into the model, they first enter the Storage CARTS, simulating the process of taking a shopping basket. No capacity definition card is included for this Storage. The Processor consequently assumes a default capacity of 2,147,483,647, as previously explained in Section 2.36. This insures that, consistent with

the problem statement, "there is no limit to the number of shopping baskets". The output can then be examined to determine the basket requirement through the statistic "MAXIMUM CONTENTS" of the Storage.

The beginning of each aisle is modeled with the TRANSFER Block in statistical transfer mode. Customers who decide not to move down a particular aisle are then routed immediately to the TRANSFER Block which simulates the entrance to the next aisle. For each aisle, the distribution of "items selected" is described with a continuous, two-point GPSS Function. Those who shop an aisle sample from the appropriate Function at an increment-mode ASSIGN Block, accumulating the total number of items selected so far in a Parameter. After the ASSIGN Block, time required to shop the aisle is simulated at an ADVANCE Block before the customer-Transaction moves into the TRANSFER Block simulating the beginning of the next aisle.

After finishing the aisle sequence, the customer-Transaction joins the line ahead of the checkout girl. Before capturing her, selection of impulse items is simulated by referencing an appropriate Function at an ASSIGN Block. The number of these additional items is simply added to the items-total accumulated earlier. Checkout time is then determined by sampling from a distribution whose argument is the Parameter containing the items-total. After checking out, the shopping basket is returned by leaving the CARTS Storage, and the customer-Transaction is removed from the model.

In summary, model features of interest are (a) use of a Storage with no capacity definition card, (b) sampling from uniform distributions at ASSIGN Blocks (c) increment-mode use of a Transaction Parameter, and (d) use of a Function which has a Transaction Parameter as its argument

(3) Table of Definitions

Time Unit: 1 Second

GPSS Entity	Interpretation
Transactions	
Model Segment 1	Customers P1: The number of items selected for purchase
Model Segment 2	A timer
Facilities	
GIRL	The checkout girl
Functions	
AYL1, AYL2, and AYL3	Functions describing the "items selected" distributions in aisles 1, 2, and 3, respectively
COTYM	Function describing the dependence of checkout time on the number of items being purchased
IMPUL	Function describing the distribution of impulse items selected at the checkout counter
XPDIS	Function for sampling from the exponential distribution with a mean value of 1
Queues	
GIRL	The Queue used to gather statistics for the line of customers waiting to check out
Storages	
CARTS	Storage used to simulate shopping baskets

Table 4B.2 Table of Definitions

(4) Block Diagram

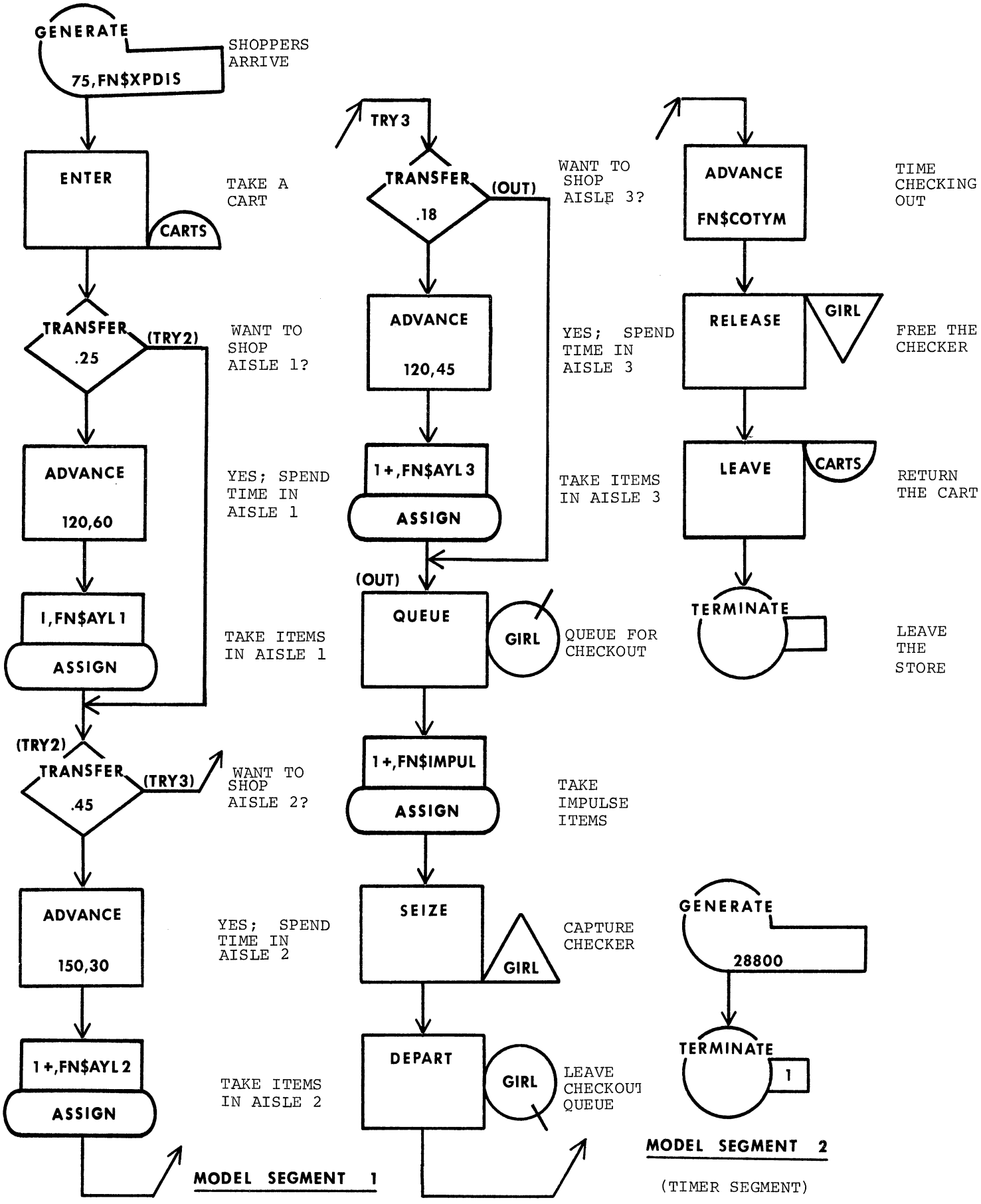


Figure 4B.1 Block Diagram for Case Study 4B

(5) Extended Program Listing

```
BLOCK                                                                    CARD
NUMBER  *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS                NUMBER
               SIMULATE                                                     1
        *                                                                   2
        *      DEFINE FUNCTIONS                                             3
        *                                                                   4
         AYL1  FUNCTION   RN1,C2       ITEMS-SELECTED DISTRIBUTION, AISLE 1  5
        0,2/1,5                                                             6
         AYL2  FUNCTION   RN1,C2       ITEMS-SELECTED DISTRIBUTION, AISLE 2  7
        0,3/1,6                                                             8
         AYL3  FUNCTION   RN1,C2       ITEMS-SELECTED DISTRIBUTION, AISLE 3  9
        0,4/1,7                                                            10
         COTYM FUNCTION   P1,C2        CHECKOUT-TIME DISTRIBUTION           11
        1,3/18,54                                                          12
         IMPUL FUNCTION   RN1,C2       DISTRIBUTION OF ITEMS TAKEN ON IMPULSE 13
        0,1/1,4                                                            14
         XPDIS FUNCTION   RN1,C24      EXPONENTIAL DISTRIBUTION FUNCTION    15
        0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38 16
        .8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2 17
        .97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8           18
        *                                                                  19
        *      MODEL SEGMENT 1                                             20
        *                                                                  21
1               GENERATE   75,FN$XPDIS     SHOPPERS ARRIVE                22
2               ENTER      CARTS           TAKE A CART                    23
3               TRANSFER   .25,,TRY2       WANT TO SHOP AISLE 1?          24
4               ADVANCE    120,60          TIME IN AISLE 1                25
5               ASSIGN     1,FN$AYL1       SET P1 = ITEMS SELECTED IN AISLE 1 26
6        TRY2   TRANSFER   .45,,TRY3       WANT TO SHOP AISLE 2?          27
7               ADVANCE    150,30          TIME IN AISLE 2                28
8               ASSIGN     1+,FN$AYL2      SET P1 = TOTAL ITEMS SELECTED SO FAR 29
9        TRY3   TRANSFER   .18,,OUT        WANT TO SHOP AISLE 3?          30
10              ADVANCE    120,45          TIME IN AISLE 3                31
11              ASSIGN     1+,FN$AYL3      SET P1 = TOTAL ITEMS SELECTED SO FAR 32
12       OUT    QUEUE      GIRL            QUEUE FOR CHECKOUT             33
13              ASSIGN     1+,FN$IMPUL     ADD TO P1 ITEMS TAKEN ON IMPULSE 34
14              SEIZE      GIRL            CAPTURE THE CHECKER            35
15              DEPART     GIRL            LEAVE THE CHECKOUT QUEUE       36
16              ADVANCE    FN$COTYM        CHECK-OUT TIME                 37
17              RELEASE    GIRL            FREE THE CHECKER               38
18              LEAVE      CARTS           RETURN THE CART                39
19              TERMINATE                   LEAVE THE STORE                40
        *                                                                  41
        *      MODEL SEGMENT 2                                             42
        *                                                                  43
20              GENERATE   28800           TIMER ARRIVES AT END OF 8-HOUR DAY 44
21              TERMINATE  1               SHUT OFF THE RUN               45
        *                                                                  46
        *      CONTROL CARDS                                               47
        *                                                                  48
               START      1,,,1           START THE RUN; GET CHAIN PRINTOUT  49
               END        RETURN CONTROL TO OPERATING SYSTEM                50
```

Figure 4B.2 Extended Program Listing for Case Study 4B

FUNCTION SYMBOLS AND CORRESPONDING NUMBERS

1	AYL1
2	AYL2
3	AYL3
4	COTYM
5	IMPUL
6	XPDIS

(a) Function Symbol Table

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
GIRL	.403	363	32.008		

(b) Facility Statistics

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
GIRL	6	.165	363	211	58.1	13.101	31.289		

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(c) Queue Statistics

STORAGE	CAPACITY	AVERAGE CONTENTS	AVERAGE UTILIZATION	ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENTS	MAXIMUM CONTENTS
CARTS	2147483647	4.027	.000	366	316.926	3	14

(d) Storage Statistics

Figure 4B.3 Selected Program Output for Case Study 4B

FUTURE EVENTS CHAIN

TRANS	BDT	BLOCK	PR	SF	NBA	SET	MARK-TIME	P1	P2	P3	P4	SI	TI	DI	CI	MC	PC	PF
16	28806	4			5	16	28632	0	0	0	0				4			
								0	0	0	0							
								0	0	0	0							
12	28830	4			5	12	28766	0	0	0	0				4			
								0	0	0	0							
								0	0	0	0							
7	28836	10			11	7	28380	8	0	0	0				4			
								0	0	0	0							
								0	0	0	0							
15	28840				1	15	-366	0	0	0	0				4			
								0	0	0	0							
								0	0	0	0							
3	57600				20	3	-1	0	0	0	0				4			
								0	0	0	0							
								0	0	0	0							

Row 1 (P1-P4)

Row 2 (P5-P8)

Row 3 (P9-P12)

Figure 4B.4 Future Events Chain at End of Case Study 4B Simulation

(7) Discussion

Program Output

Figure 4B.3(a) shows the Function symbol table. The six Functions, all named symbolically in the model, have been assigned the numbers 1 through 6 by the Processor. The order in which the numbers have been assigned corresponds to the order in which the Function definitions appear in the card deck, as indicated in Figure 4B.2.

Figure 4B.3(b) indicates that utilization of the checkout girl was 0.403. Consistent with this low utilization, Queue statistics in part (c) of that figure show that less than half of the customers had to wait for service at the checkout counter. Those who had to wait were in the line about 31 seconds before being served. As many as 6 people were simultaneously queued up to check out.

The model was not designed to directly measure the average number of items selected for purchase. Nevertheless, average holding time at the checkout counter was about 32 seconds, indicating that about 11 items were checked out on average by each shopper. (The model uses 3 seconds per item as the deterministic checkout time.)

The Storage statistics in part (d) of Figure 4B.3 show that about 4 shopping baskets were in use on average, and that as many as 14 were in use simultaneously. These Storage statistics also reveal that the average customer spent about 314 seconds in the grocery store. Observe that the listed capacity of the Storage is 2,147,483,647. With this large capacity, the AVERAGE UTILIZATION correct to three decimal places is zero, even though the average Storage content was 4.

The D Operand has been used on the START Card (Card 49, Figure 4B.2) to force a printout of the Current and Future Events Chains at the end of the simulation. Because no customers were waiting to check out when the simulation terminated, the Current Events Chain was empty. The Future Events Chain is shown in Figure 4B.4. The figure is primarily of interest for calling attention to the way a Transaction's Parameter values appear in a chain printout. Each Transaction in the model has 12 halfword Parameters, by default. Because only four columns are used in the chain printout to show Parameter values, three rows of numbers are required to display the 12 values involved. The three rows of Parameter values for the last Future Events Chain Transaction have been pointed out explicitly in Figure 4B.4. The first row contains values for P1, P2, P3, and P4; the second row for P5, P6, P7, and P8; and the third row for P9, P10, P11, and P12. Only the labels "P1", "P2", and "P3", and "P4" appear above the four columns of numbers.

4.9 Exercises

4.9.1 (a) When a Transaction enters a model through the Block "GENERATE 52,FN$XPDIS,,10" how many Parameters will it have? Will the Parameters be of the half-word or fullword type?

(b) Transactions entering a model through a particular GENERATE Block are each to have 20 fullword Parameters. The interarrival time of these Transactions is to be 42±8 time units. Show a GENERATE Block which accomplishes the desired effect.

(c) Critically discuss the statement, "-320000 is a valid Parameter value".

(d) Critically discuss the statement, "P15 is a valid Parameter reference".

(e) Critically discuss the statement, "P$SYMBL is a valid Parameter reference".

(d) What is the value of each of a Transaction's Parameters when the Transaction first enters a model?

(e) In Figure 4B.4, give interpretations for each Transaction on the Future Events Chain, and give an explanation for the indicated P1 value of each Transaction.

(f) In Appendix C, read the description of error message 850. If an error of this type were to occur, how would you correct it?

4.9.2 Assume that the Functions ONE and TWO are defined as shown in Figure P4.9.2.

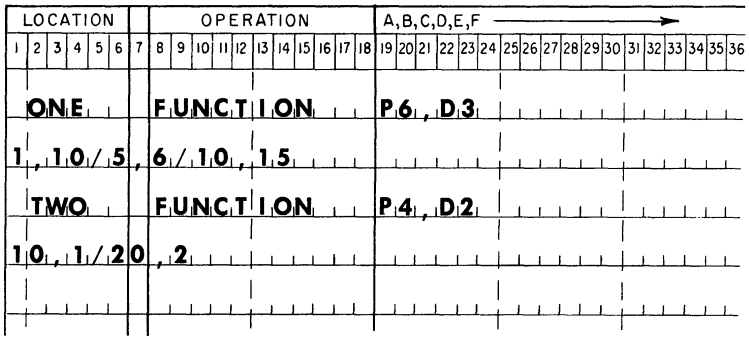


Figure P4.9.2

What will be the holding time when a Transaction with P4 and P6 values of 12 and 14, respectively, moves into each of the ADVANCE Blocks whose punchcard images are shown below?

(a) ADVANCE P4

(b) ADVANCE FN$TWO

(c) ADVANCE P4,FN$ONE

(d) ADVANCE P6,FN$TWO

(e) ADVANCE FN$ONE,FN$TWO

4.9.3 Assume that the Functions ALPHA and BETA are defined as shown in Figure P4.9.3.

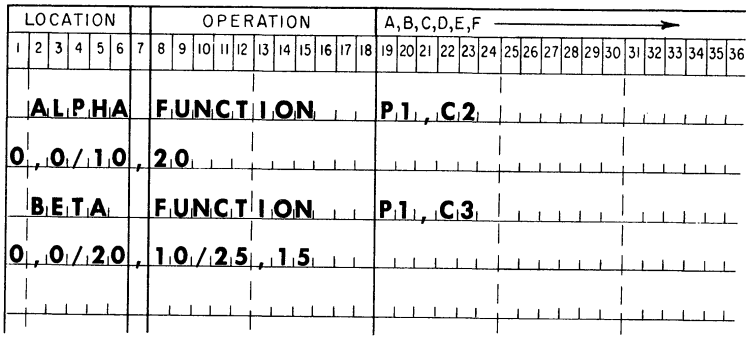


Figure P4.9.3

What will be the holding time when a Transaction with a P1 value of 5 moves into each of the ADVANCE Blocks whose punchcard images are shown below?

(a) ADVANCE FN$BETA

(b) ADVANCE FN$ALPHA,FN$BETA

4.9.4 Assume that, at a given time during the course of a simulation, the entity properties shown in Figure P4.9.4(a) are in effect.

Standard Numerical Attribute	Value
F1	0
FR1	532
R$TUGS	3
Q$LONG	8
QZ$LONG	1

Figure P4.9.4(a)

Also assume that a particular Transaction has the Parameter values shown in Figure P4.9.4(b).

Parameter Number	Value
1	30
2	-10
3	4
4	2
5	3

Figure P4.9.4(b)

State which Parameter will have its value changed, and what the new value will be, when the Transaction moves into each of the ASSIGN Blocks whose punchcard images are shown below. (Note: assume that the ASSIGN Blocks are independent of each other.)

(a) ASSIGN 5,-100

(b) ASSIGN 5-,100

(c) ASSIGN P5,100

(d) ASSIGN P5+,96

(e) ASSIGN P4+,P5

(f) ASSIGN P5+,P4

(g) ASSIGN R$TUGS,P5

(h) ASSIGN QZ$LONG,FR1

(i) ASSIGN P3,F1

4.9.5 (a) Discuss the difference between the two Blocks, "ASSIGN 5-,100,3" and "ASSIGN 5-,100,FN3".

(b) Assume that FN1 has a value of 3.26, and FN3 has a value of 4.1. What value will Parameter 2 have after a Transaction enters the Blocks shown below?

(i) ASSIGN 2,60,1

(ii) ASSIGN 2,60,FN1

(c) Compare and contrast the Block "ASSIGN 7,1,3" with the Block "ASSIGN 7,FN3" with respect to (i) the value assigned to Parameter 7, and (ii) probable relative differences in the time required for Block execution.

4.9.6 Questions for this problem make reference to Figure 4B.2.

(a) What value does the Function COTYM return when 15 is the value of P1?

(b) Explain why the second pair of points used to define the Function COTYM is "18,54".

(c) Show what changes to make in the Function COTYM if it takes 5 seconds on average to check out an item.

(d) What is the least number of items that a shopper in the grocery store will select for purchase?

(e) For the ASSIGN Block in Location 5, discuss the possibility of using 1+, rather than 1, as the A Operand.

4.9.7 The Block Counts in the output produced when the Figure 4B.2 model was run are shown in Figure P4.9.7.

RELATIVE CLOCK 28800 ABSOLUTE CLOCK 28800

BLOCK COUNTS

BLOCK	CURRENT	TOTAL	BLOCK	CURRENT	TOTAL	BLOCK	CURRENT	TOTAL
1	0	366	11	0	306	21	0	1
2	0	366	12	0	363			
3	0	366	13	0	363			
4	2	280	14	0	363			
5	0	278	15	0	363			
6	0	364	16	0	363			
7	0	191	17	0	363			
8	0	191	18	0	363			
9	0	364	19	0	363			
10	1	307	20	0	1			

Figure P4.9.7

(a) Use the Block Counts to compute the fraction of customers who shopped aisle 1. Compare your results with the long-run fraction expected.

(b) Repeat (a) for aisle 2.

(c) How many shoppers were in aisles 1, 2, and 3, respectively, when the simulation shut off?

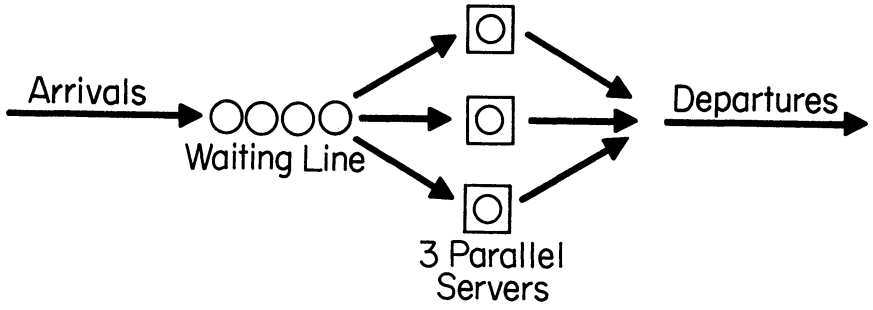
4.9.8 Show how to modify Case Study 4B to incorporate these changes.

(a) There are only 10 baskets available in the grocery store. Customers wait for baskets if none are available when they arrive at the store.

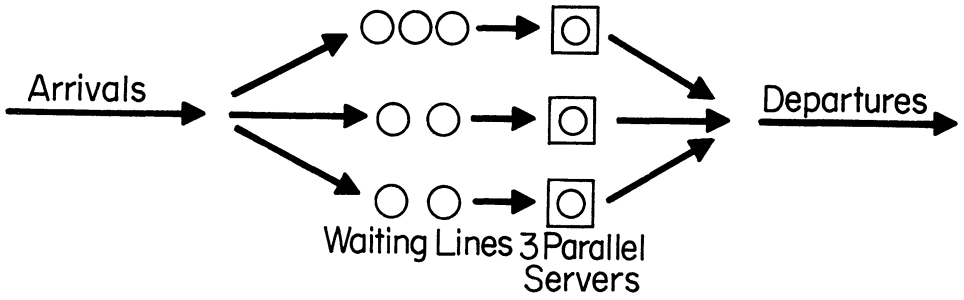
(b) There are only 10 baskets available in the store. Sixty-five percent of the customers wait for baskets if none are available when they arrive. The other 35% leave without shopping.

4.10 Multiple-Line, Multiple-Server Queuing Systems

Figure 4.12(a) illustrates a one-line, multiple-server queuing system for the case of 3 parallel servers. The modeling of such a system in GPSS was discussed in Section 2.37. Recall that this involved simulating the parallel servers with a Storage, and that a single Queue was used to gather waiting line statistics ahead of the Storage (see Figure 2.44).



(a) One-Line, Multiple-Server Queuing System



(b) Multiple-Line, Multiple-Server Queuing System

Figure 4.12 Alternative Queuing Systems with Three Parallel Servers

Part (b) of Figure 4.12 shows a multiple-server queuing system in which a separate waiting line forms ahead of each server. The multiple-line, multiple-server queuing system arises at grocery store checkout counters, expressway toll points, bank teller windows, and so on. We have now reached the point where an elementary approach to modeling the Figure 4.12(b) system in GPSS can be introduced. The approach involves simulating each of the parallel servers with a Facility. Ahead of each Facility is a corresponding Queue. For example, suppose that Facilities 1, 2, and 3 are used to simulate the three parallel servers in Figure 4.12(b). Let the Queues ahead of the three Facilities also be numbered 1, 2, and 3, respectively. Then the multiple servers can be simulated with the Block Diagram segment shown in Figure 4.13. As suggested in the figure, Transactions moving into the Block Diagram segment select one of the three parallel branches to move through. After moving through that branch, they continue on in common in the model.

There is much repetition in each of the three parallel model segments in Figure 4.13. In fact, the Block sequences are identical in each branch. Only the A Operands at the QUEUE, SEIZE, DEPART, and RELEASE Blocks differ. If these Operands were indirectly specified as the value of a Transaction Parameter, a single sequence of Blocks could be used to replace the three parallel branches.

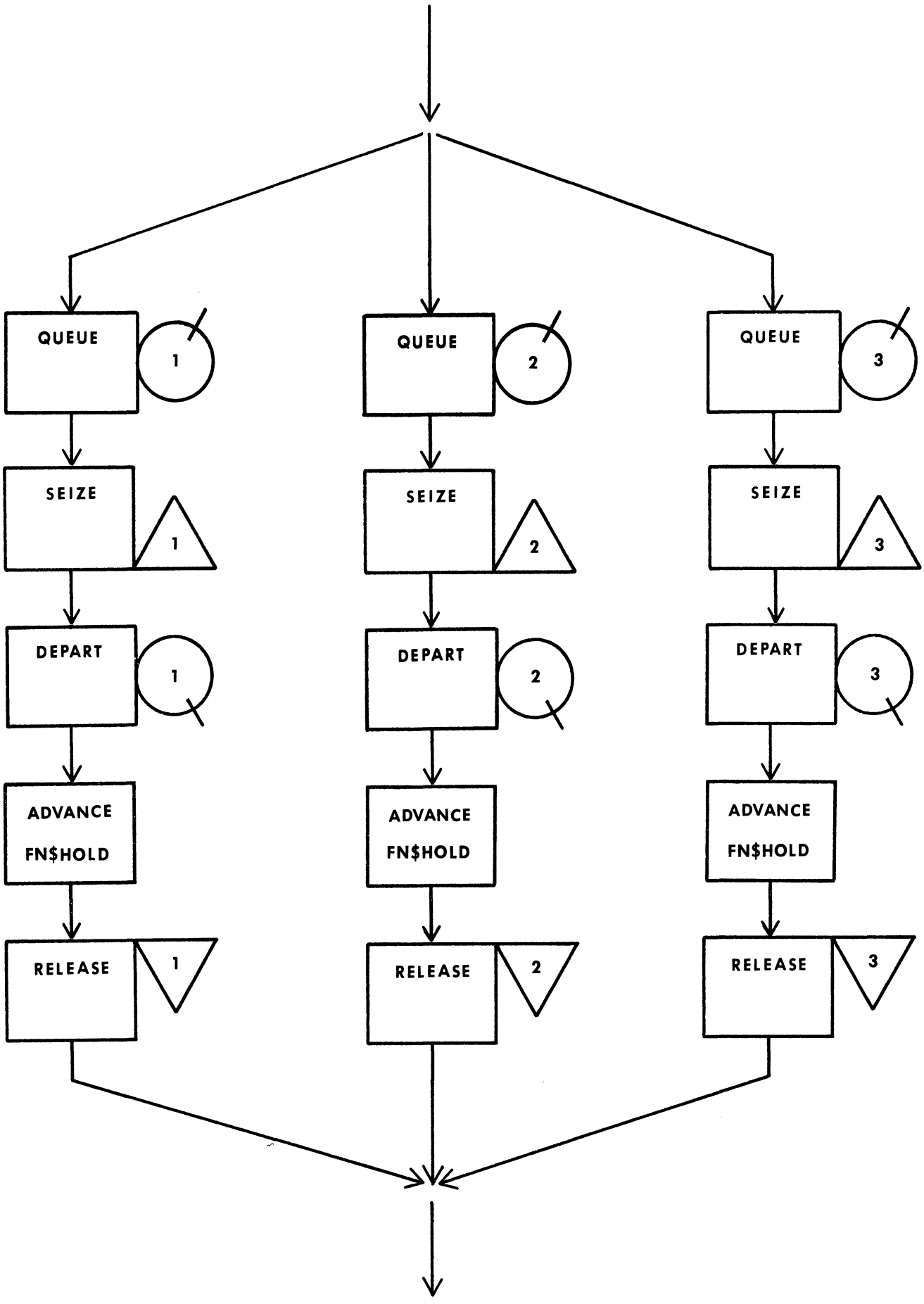


Figure 4.13 A Block Diagram Segment which Simulates Parallel Servers with Parallel Facilities

Suppose, for example, that a Transaction "selects the number of a server" just before it reaches the Figure 4.13 model segment, and that this number is assigned as the value of Transaction Parameter 1. Figure 4.14 then shows a single Block sequence equivalent to the three Figure 4.13 parallel branches. When a Transaction with a P1 value of 1 moves through the Figure 4.14 sequence, it joins Queue 1, waits to capture Facility 1, etc. Similarly, Transactions with P1 values of 2 or 3 join Queue 2 or 3, respectively, wait to capture Facility 2 or 3, and so on. In summary, the Figure 4.14 segment compactly accomplishes the effect of Figure 4.13. Of course, the "economy of expression" achieved by use of indirect specification in Figure 4.14 would be even greater if there were more than 3 servers in parallel.

It was assumed earlier that a Transaction "selects the number of a server" just before it reaches the Figure 4.14 (or 4.13) model segment. A method for doing this in GPSS will be discussed next.

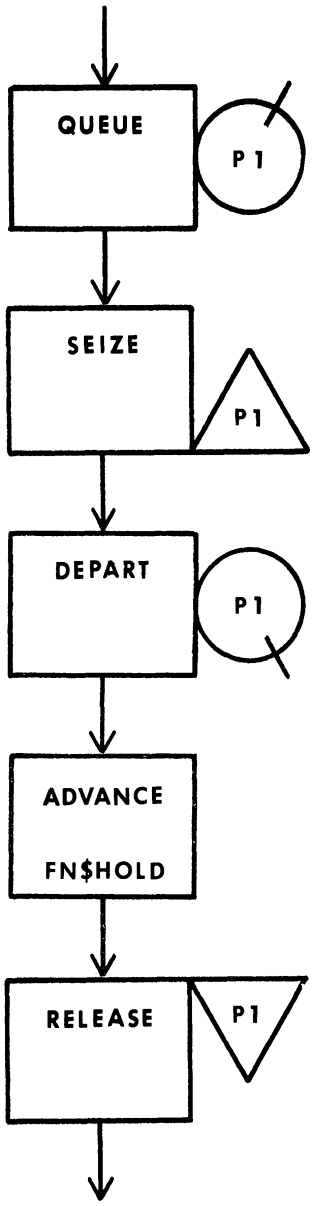


Figure 4.14 A Single Block Sequence Equivalent to the Three Parallel Branches in Figure 4.11

4.11 Finding Entities Satisfying Stated Conditions: The SELECT Block

The customer arriving at a multiple-line, multiple-server queuing system usually does one of two things.

(1) If one of the servers is currently available, he captures that server.

(2) If no servers are available, he joins the shortest waiting line and remains there until his turn for service comes up.

This represents the simplest possible view of the situation. Complications can easily arise. For example, even though one or more servers are free, the customer may elect to wait for a "favorite" server who is busy at the moment. Similarly, instead of simply joining the shortest line and waiting there, customers may "jump lines" when they sense there is an exceptionally long delay in their line, or that another line is moving unusua fast.

In this section, only the simplest customer behavior itemized above will be considered. First, corresponding to (1), a method for finding a currently available server (if any) will be introduced. Then, corresponding to (2), a means of identifying the Queu with the smallest current content will be described. Finally, use of the methods to simulate the server-selection process will be described.

4.11.1 Finding the First Member in a Set

GPSS makes available a Block which can be used to scan a specified set of entity members in search for the first one which satisfies a stated numeric condition. When the first such member (if any) has been found, the search terminates and the search-initiatir Transaction moves on in the model. Searches such as these can be conducted.

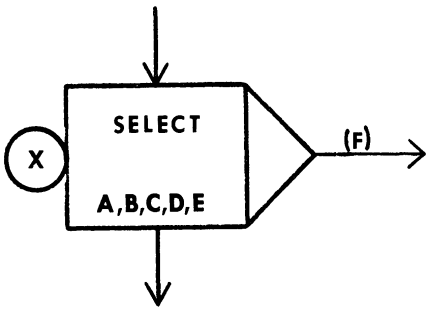
(1) Find the first Facility among those numbered 1, 2, and 3 which is not currently in use (i.e., with an Fj value equal to 0).

(2) Find the first Storage among Storages 5, 6, 7, and 8 with a fractional utilizatic less than 25% (i.e., with an SRj value less than 250).

(3) Find the first Queue among those numbered 13, 14, 15, 16, and 17 with an average residence time, excluding zero entries, greater than or equal to 3 time units (i.e., with QZj greater than 3).

The SELECT Block is used to conduct such searches. The Block is shown with its Auxiliary Operator and Operands in Figure 4.15. The E Operand supplies the family name of the Standard Numerical Attribute being investigated (such as F, SR, or QZ in the above examples). The B and C Operands indicate the smallest and largest numbers, respectively, in the range of entity members to be examined (such as 1 and 3; 5 and 8; or 13 and 17 in the above examples). The D Operand supplies the data against which the E-Operand SNA is to be compared (such as 0, 250, and 3 in the examples). The pertinent relation of interest between the Standard Numerical Attribute and the D-Operand data is spelled out throug an Auxiliary Operator, represented with an X in Figure 4.15. In use, the Auxiliary Operator will be G, GE, E, NE, LE, or L to signify "greater than", "greater than or equal to", and so on.

The A Operand indicates the Parameter into which is placed the number of the first entity member satisfying the stated condition. The Parameter belongs to the selecting Transaction, i.e., the Transaction which moved into the SELECT Block, causing the search to be conducted. Note the possibility that no entity member satisfies the indicated condition. At the user's option, the selecting Transaction can be diverted to a non-sequential Block when this is the case. The optional F Operand indicates the Location of this non-sequential Block. If the F Operand is not used, the selecting Transaction moves unconditionally from the SELECT Block to the sequential Block.



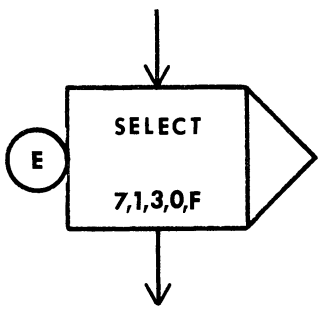
Operand	Significance	Default Value or Result
E	The family name of the Standard Numerical Attribute being investigated.	Error
B and C	The smallest and largest numbers, respectively, in the range of entities to be examined	Error
D	The data against which the E-Operand SNA is to be compared	Error
X	X is an Auxiliary Operator. It represents the relational operator which specifies the way in which the E-Operand SNA is to be related to the D-Operand data. In practice, X takes one of the forms shown below.	Error

Relational Operator	Interpretation in SELECT Block Context
G	Is E greater than D?
GE	Is E greater than or equal to D?
L	Does E equal D?
LE	Is E less than or equal to D?
L	Is E less than D?

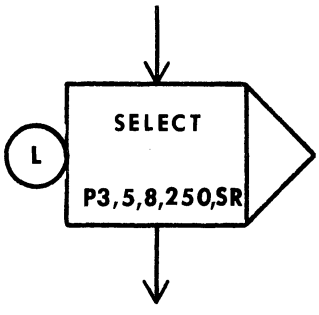
A	Number of the Parameter into which the number of the first-such entity is to be put	Error
F	Optional Operand; Block Location to which the selecting Transaction moves if no entity satisfies the indicated condition	Selecting Transaction moves to sequential Block unconditionally

Figure 4.15 The SELECT Block with Auxiliary Operator and Operands

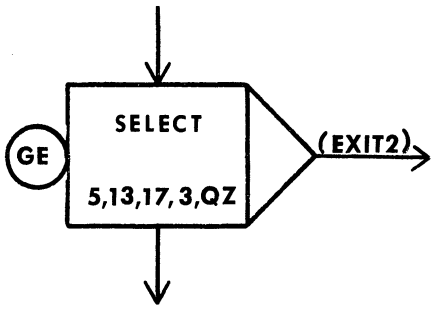
Figure 4.16 shows SELECT Blocks with which the three searches described earlier can be performed. In Figure 4.16(a), a search is conducted to find the first



(a) First Example



(b) Second Example



(c) Third Example

Figure 4.16 Examples of SELECT Block Use

Facility among Facilities 1, 2, and 3, with an Fj value equal to 0. The number of the first such Facility found is placed in Parameter 7 of the selecting Transaction. If none of these Facilities has an Fj value of 0, the selecting Transaction's P7 value is left unchanged. In either case, the selecting Transaction moves to the sequential Block next.

In Figure 4.16(b), Storages 5 through 8 are searched to find the first one with a fractional utilization less than 250 parts per thousand. The number of the first-such Storage found, if any, is placed in the Parameter whose number is the value of P3. The selecting Transaction continues to the sequential Block.

Figure 4.16(c) shows use of the optional exit from the SELECT Block. Queues 13 through 17 are searched to find one with a QZ value greater than or equal to 3. If such a Queue is found, its number is placed in Parameter 5, and the selecting Transaction moves to the sequential Block. If no Queue satisfies the condition, P5 is unchanged, and the selecting Transaction moves to the Block in the Location EXIT2.

A series of additional SELECT Block features should be noted. First, the SELECT Block's A, B, C, and D Operands can all be specified indirectly. In the Figure 4.16 examples, indirect specification was only used for the A Operand in (b). Second, the imposed condition takes the form "SNA/relational operator/D-Operand data". Hence, in Figure 4.16(b), the condition is "SRj less than 250", not "250 less than SRj". Third, the entity members are scanned in order of increasing number. Hence, the smallest-numbered member which satisfies the indicated condition brings the search to an end. Fourth, all entity members in the range specified by the B and C Operands are included in the "search". With direct use of the SELECT Block it is not possible, then, to find the first Facility among those numbered 3, 4, 7, 10, and 12 with an Fj value equal to 0.

SELECT is the first instance of a Block which uses an Auxiliary Operator. There are several other Blocks in the language which also make use of Auxiliary Operators. Wnen preparing the punchcard corresponding to any one of these Blocks, the Auxiliary Operator is punched in the Operation Field. It is entered after the corresponding Block Operation, with a single blank column separating it from the Operation.

4.11.2 Finding the Set Member with a Minimum or Maximum Attribute

The SELECT Block as just described is used in "first-such" mode to find the first member of a set, if any, which satisfies a stated numeric condition. In an alternate mod the same Block can be used to find the set member having a minimum or maximum attribute value. This makes it possible to answer questions such as these.

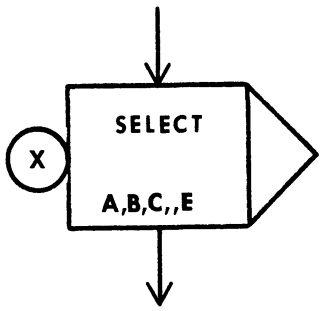
(1) In a set of Queues, which has the minimum current content?

(2) In a set of Facilities, which has the minimum utilization?

(3) In a set of Storages, which has the maximum remaining capacity?

The general form of these questions is "which entity in a given range has the minimum or maximum value of a specified attribute?" The SELECT Block can be used in MIN or MAX mode to provide the answer to such questions.

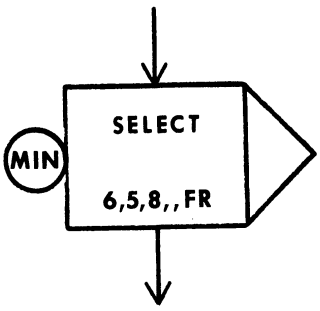
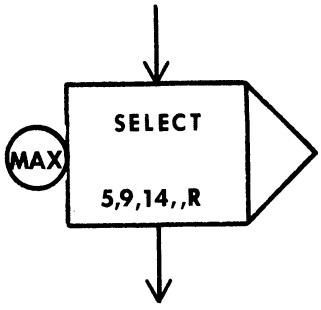
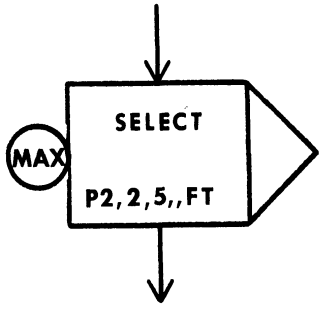
The role of the Auxiliary Operator and Operands for the SELECT Block in MIN or MAX mode is shown in Figure 4.17. The A, B, C, and E Operands have the same significance as when the SELECT Block is used in first-such mode. Taken together, they tell which entity attribute is of interest (E Operand), which entity members are involved (B and C Operands), and where to put the result of the search (A Operand). Because the attribute must satisfy an absolute condition, no comparative D-Operand data is needed, and the D Operand is not used. The Auxiliary Operator, instead of describing a relation, takes the form of MIN or MAX, depending on whether the entity with the minimum or maximum attribute value is sought. Finally, because some entity among those considered has the minimum or maximum attribute value, no "if any" condition is associated with MIN or MAX mode SELECT Block usage. No "none such found" optional exit is available, then, meaning that no F Operand is used with the Block.

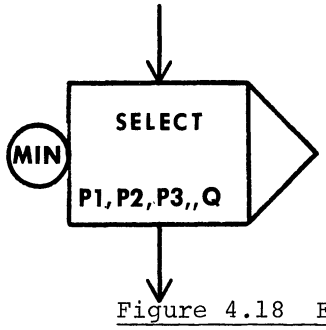


Operand	Significance	Default Value or Result
E	The family name of the Standard Numerical Attribute whose value is being investigated	Error
B and C	The inclusive lower and upper limits, respectively, of the range of numbers of the specific family members involved	Error
A	The number of a Parameter into which is to be put the number of the family member whose attribute is the minimum or maximum	Error
X	The Auxiliary Operator. X is to be MIN or MAX, depending on whether the entity with the minimum or maximum attribute value is sought, respectively	Error

Figure 4.17 The SELECT Block as Used in MIN or MAX Mode

Figure 4.18 shows four examples of SELECT Block use in MIN and MAX mode, and

Example	Explanation
(a) 	Which Facility among those numbered from 5 to 8 has the smallest fractional utilization? Put the number of the Facility in Parameter 6.
(b) 	Which Storage among those numbered from 9 to 14 has the largest remaining capacity? Put the number of the Storage in Parameter 5.
(c) 	Which Facility among those numbered from 2 to 5 has the largest holding time per capture? Put the number of the Facility in the Parameter whose number is the value of P2.



(d) Which Queue among those numbered from the value of P2 to the value of P5 has the smallest content? Put the number of the Queue in the Parameter whose number is the value of P1.

Figure 4.18 Examples of SELECT Block Use in MIN and MAX Mode

provides a corresponding explanation for each Block. These examples should be studied and the role of the Block Operands should be checked. Note, in particular, that examples (c) and (d) make use of indirect specification in supplying the A Operand. Example (d) also makes use of indirect specification in supplying the B and C Operands.

4.11.3 SELECT Block Use for the Multiple-Line Queuing System

Figure 4.19 shows how the SELECT Block can be used to support simulation of the

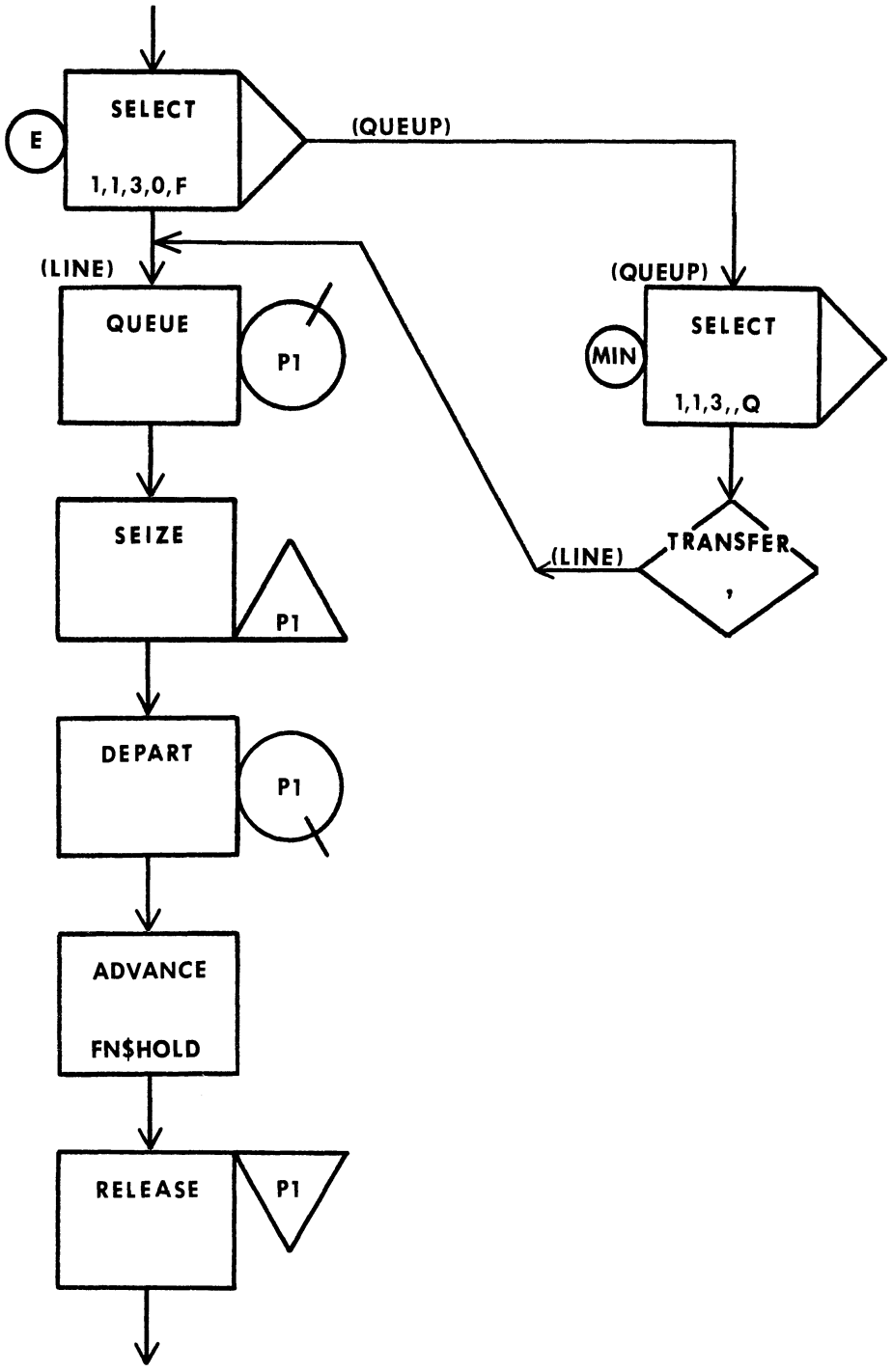


Figure 4.19 SELECT Block Use for the Multiple-Line Queuing System

parallel servers shown earlier in Figure 4.14. The customer-Transaction first enters a SELECT Block used in first-such mode, where a search is conducted to find a Facility which is currently available. If such a Facility is found, the Facility number is placed in Parameter 1. The Transaction then exits sequentially to the QUEUE-SEIZE-DEPART-ADVANCE-RELEASE sequence, where use of the server is simulated. If no Facility is concurrently available, the Transaction exits non-sequentially from the SELECT Block, moving into the MIN-mode SELECT Block at Location QUEUP. There, the number of the shortest waiting line is placed in Parameter 1. The Transaction then transfers back to the QUEUE Block, where it becomes a member of the shortest waiting line. Hence, by use of the SELECT Block both in first-such and MIN mode, the simple server-selection process described earlier is easily modeled in GPSS.

4.12 Case Study 4C: Comparison of Multiple-Server Queuing Systems (One-Line vs. Multiple Line)

(1) Statement of the Problem

Customers arrive at a bank in a Poisson pattern at a mean rate of 200 per hour. Eight teller windows are open at the bank at all times. A separate waiting line forms ahead of each teller. If a teller is free when a customer enters the bank, the customer goes immediately to that teller. Otherwise, he joins whichever waiting line is the shortest. He then remains in that line on a first-come, first-served basis until he has transacted his business, after which he leaves the bank.

The various types of business which customers transact fall into five categories. The relative frequencies of these categories, and the corresponding mean service time requirements, are shown in Table 4C.1. In each category, service time is exponentially distributed. No customer transacts business in two or more categories in a single visit to the bank.

Category of Business	Relative Frequency	Mean Service Time, Seconds
1	.10	45
2	.19	75
3	.32	100
4	.24	150
5	.15	300

Table 4C.1 Data for Business Transacted at the Bank in Case Study 4C

The bank's manager has noticed that the waiting lines are rather long on average. He would like to reduce the customer's typical waiting experience, but prefers not to hire one or two additional tellers to bring this about. He knows that another bank in the vicinity has introduced a "Quickline" queuing system. In such a system, customers entering the bank form a single line. Whenever a teller becomes available, the customer at the head of the line goes to that teller. The Quickline system presumably reduces waiting time by eliminating situations in which slow-moving lines form ahead of tellers who have been captured by a customer requiring a large amount of time. Because this conjecture seems reasonable the manager decides to experiment with the Quickline system.

Build a GPSS model which gathers waiting line information for the bank's operation, both as it currently exists, and for the proposed change to Quickline. Run the model for five different 5-hour days in each case, then compare and contrast the results. Control experimental conditions so that the alternative systems are compared in as fair a manner as possible. This means that the customer inter-arrival time sequence should

be the same in both experiments. So should the customer service time sequence. That is, if the 21st customer to enter the multiple-line system on the third day requires, say, 165 seconds for service, the same should be true for the 21st customer entering the Quickline system on the third day.

(2) Approach Taken in Building the Model

One-Line System

The basic Block sequence for the one-line system is a familiar one. The major question that arises, then, involves arranging for control of experimental conditions. Reproducibility of the inter-arrival time pattern can be guaranteed by dedicating RN1 exclusively to sampling from the inter-arrival time distribution. Control of the service time pattern requires somewhat more thought. The relative order in which customers go into service will, in general, vary between the one-line and multiple-line systems. Hence, it is not possible to control the service times simply by sampling from the service time distribution when service begins. If this were done, for example, the 21st entry to the one-line system would draw the 21st service time sample; in contrast, the 21st entry to the multiple-line system might draw, say, the 18th service time sample in that model. The corresponding values would most likely differ from one another.

A simple remedy for this situation is to have each customer determine his service time immediately upon entering the system. Then the j-th entering customer will draw the j-th sample from the service time distribution in both models, for j = 1, 2, 3, and so on. This pre-determined service time can be stored in a Parameter of the corresponding customer-Transaction. Later, when the Transaction enters the ADVANCE Block, that Parameter can serve as the Block's A Operand. As explained in Section 4.7, the optional C Operand on the ASSIGN Block is of use in sampling from an exponential distribution. Such use of the ASSIGN Block is very convenient for this model.

There is another point to consider regarding control of experimental conditions. The question is whether conflicts will arise if one and the same continuous 24-point Function is used to simulate both the Poisson arrivals and the exponential servers. If there were any doubts about this, two distinct 24-point Functions could be used in the model, each with its own "private" random number source as argument (but see Exercise 4.16.11). But such "redundant" Function definition is not necessary in this problem. The GENERATE Block simulating the arrivals, and the ASSIGN Block pre-determining service time, can reference the same Function, XPDIS. These will be the first and second Blocks, respectively, in each of the two models. Then, in either case, the various calls on the Function will be made from the Blocks as shown in Table 4C.2. The initial two calls come

Number of Call on Function XPDIS	Source of Call
1	GENERATE
2	GENERATE
3	ASSIGN
4	GENERATE
5	ASSIGN
6	GENERATE
7	ASSIGN
and so on	

Table 4C.2 Usage Pattern of the Single Function for Sampling from the Exponential Distribution

from the GENERATE Block, the first to pre-schedule the first day's customer, the second to pre-schedule his successor. The third call pre-determines the first customer's service time. After that, a strict GENERATE-ASSIGN calling sequence is followed throughout the simulation. Hence, although each calling source uses only a subset of the values returned by the Function XPDIS, the subset will be the same in both models.

Multiple-Line System

Forcing the inter-arrival and service time patterns in this model to correspond to the one-line model is straightforward. As just explained, by using GENERATE-ASSIGN as the first two Blocks, and referencing the same Function XPDIS from each Block, reproduction of the one-line-system sequences is guaranteed.

Regarding choice of a teller, the customer-Transaction can move from the ASSIGN into a Block Diagram segment similar to that shown in Figure 4.19. After storing the number of an available server or of the shortest waiting line in a Parameter, the Transaction then moves through a QUEUE-SEIZE-DEPART-ADVANCE-RELEASE sequence to simulate use of its chosen teller. The result is to produce segregated Queue statistics for each of the 8 tellers. In addition, waiting experience aggregated over all the customers should also be maintained, for direct comparison with the output from the one-line model. This can be done by having all customers move through a QUEUE-DEPART pair referencing any Queue whose number is greater than 8 (numbers 1 through 8 will be used as the names of the 8 individual Queues ahead of the 8 tellers).

(3) Table of Definitions

Time Unit: 0.1 Seconds

GPSS Entity	Interpretation
Transactions	
Model Segment 1	Customers P1: Service time required by the customer
Model Segment 2	A timer
Functions	
5	Function for sampling from the exponential distribution with a mean value of 1
MEAN	Function describing the mean service time required for various categories of business
Queues	
ONE	The Queue used to gather statistics for the one waiting line in the bank
Storages	Storage simulating the 8 bank tellers

(a) One-Line Model

GPSS Entity	Interpretation
Transactions	
Model Segment 1	Customers P1: Service time required by the customer P2: The number of the Queue through which the customer passes, and of the teller the customer uses
Facilities	
1,2,3,4,5,6, 7, and 8	Facilities used to simulate the 8 bank tellers
Queues	
1,2,3,4,5,6, 7, and 8	The Queues used to gather statistics for the waiting lines ahead of tellers 1 through 8, respectively
10	The Queue used to gather aggregate waiting statistics for all customers moving through the bank

(b) Multiple-Line Model

Table 4C.3 Table of Definitions for Case Study 4C

(4) Block Diagrams

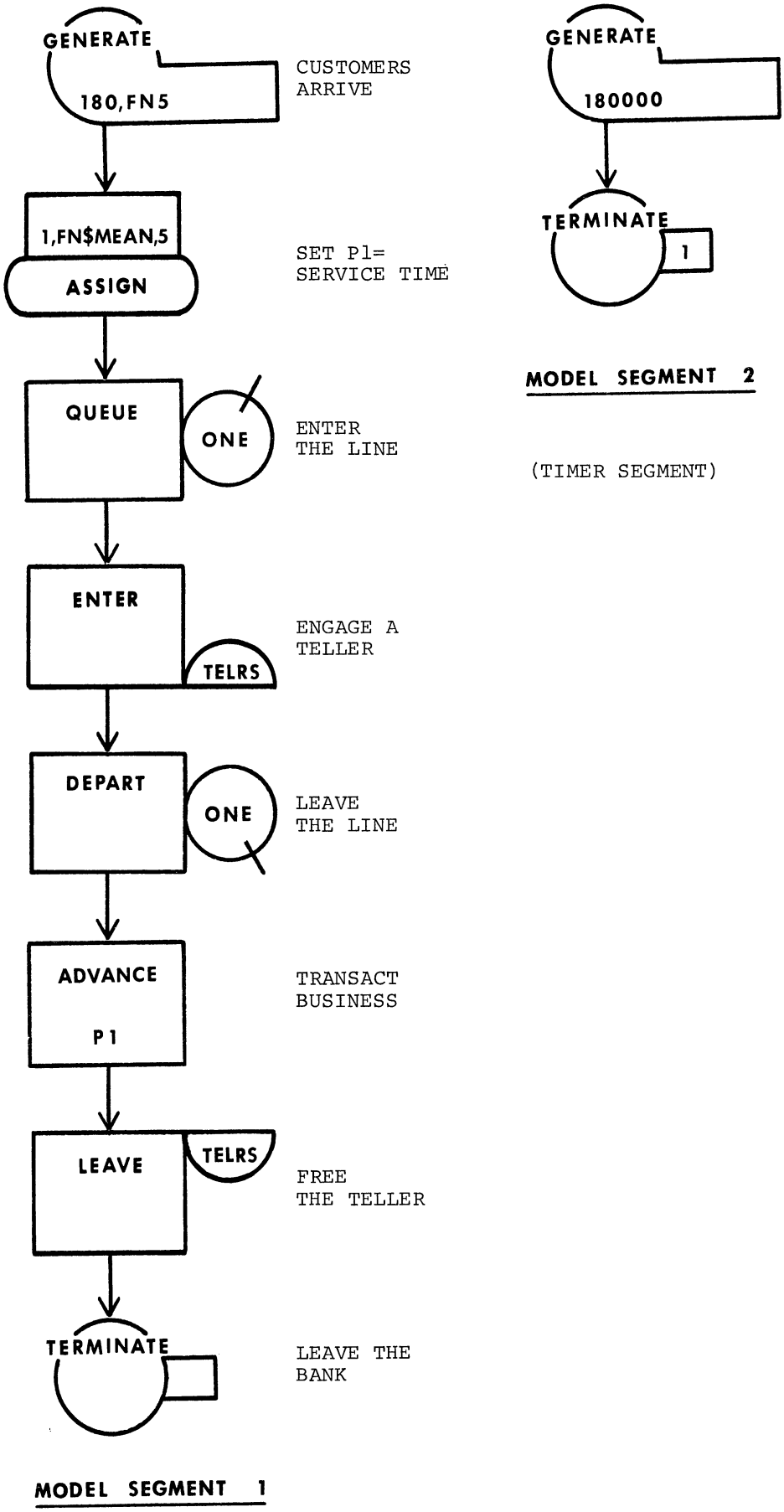


Figure 4C.2(a) Block Diagram for Case Study 4C (One-Line Model)

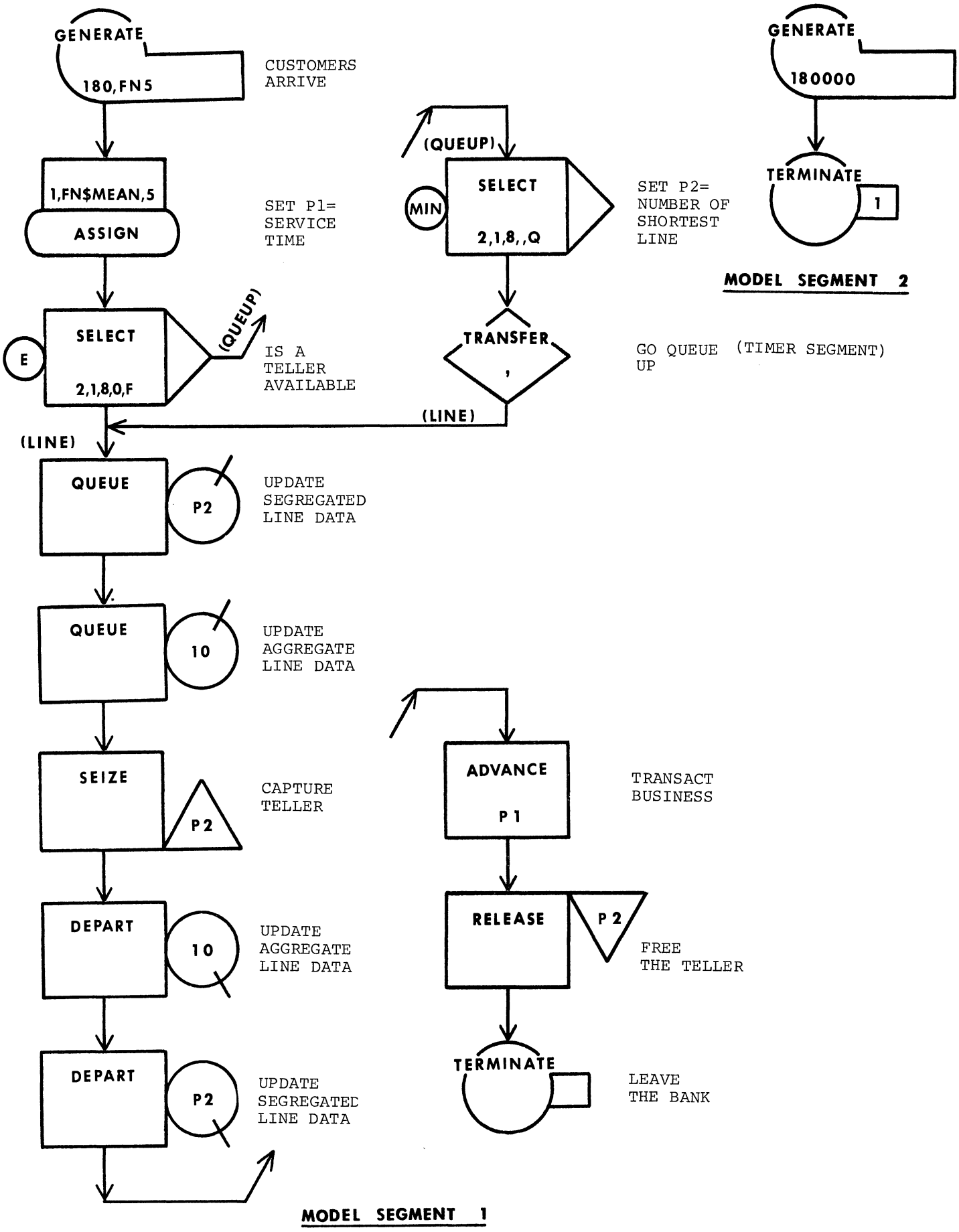


Figure 4C.2(b) Block Diagram for Case Study 4C (Multiple-Line Model)

(5) Extended Program Listings

```
BLOCK                                                                    CARD
NUMBER  *LOC   OPERATION  A,B,C,D,E,F,G            COMMENTS              NUMBER
               SIMULATE                                                      1
        *                                                                    2
        *      SET NON-STANDARD RANDOM NUMBER SEQUENCE FOR DAY 1             3
        *                                                                    4
               RMULT      111                                               5
        *                                                                    6
        *      DEFINE FUNCTIONS                                              7
        *                                                                    8
         5     FUNCTION   RN1,C24          EXPONENTIAL DISTRIBUTION FUNCTION 9
        0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38  10
        .8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2  11
        .97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8            12
         MEAN  FUNCTION   RN1,D5          DISTRIBUTION OF MEAN SERVICE TIME 13
        .1,450/.29,750/.61,1000/.85,1500/1,3000                             14
        *                                                                   15
        *      DEFINE STORAGE CAPACITY                                      16
        *                                                                   17
               STORAGE    S$TELRS,8                                        18
        *                                                                   19
        *      MODEL SEGMENT 1                                              20
        *                                                                   21
1              GENERATE   180,FN5,,,,1    CUSTOMERS ARRIVE                 22
2              ASSIGN     1,FN$MEAN,5     SET P1 = SERVICE TIME            23
3              QUEUE      ONE             ENTER THE LINE                   24
4              ENTER      TELRS           ENGAGE A TELLER                  25
5              DEPART     ONE             LEAVE THE LINE                   26
6              ADVANCE    P1              TRANSACT BUSINESS                27
7              LEAVE      TELRS           FREE THE TELLER                  28
8              TERMINATE                   LEAVE THE BANK                   29
        *                                                                   30
        *      MODEL SEGMENT 2                                              31
        *                                                                   32
9              GENERATE   180000          TIMER ARRIVES AFTER 5 HOURS      33
10             TERMINATE  1               SHUT OFF THE RUN                 34
        *                                                                   35
        *      CONTROL CARDS                                                36
        *                                                                   37
               START      1               START THE RUN FOR DAY 1          38
               RMULT      333             SET RANDOM NUMBER SEQUENCE FOR DAY 2 39
               CLEAR                       CLEAR FOR DAY 2                  40
               START      1               START THE RUN FOR DAY 2          41
               RMULT      555             SET RANDOM NUMBER SEQUENCE FOR DAY 3 42
               CLEAR                       CLEAR FOR DAY 3                  43
               START      1               START THE RUN FOR DAY 3          44
               RMULT      777             SET RANDOM NUMBER SEQUENCE FOR DAY 4 45
               CLEAR                       CLEAR FOR DAY 4                  46
               START      1               START THE RUN FOR DAY 4          47
               RMULT      999             SET RANDOM NUMBER SEQUENCE FOR DAY 5 48
               CLEAR                       CLEAR FOR DAY 5                  49
               START      1               START THE RUN FOR DAY 5          50
               END        RETURN CONTROL TO OPERATING SYSTEM                51
```

Figure 4C.2(a) One-Line Model

```
BLOCK                                                                    CARD
NUMBER  *LOC    OPERATION  A,B,C,D,E,F,G            COMMENTS             NUMBER
                SIMULATE                                                      1
        *                                                                     2
        *       SET NON-STANDARD RANDOM NUMBER SEQUENCE FOR DAY 1             3
        *                                                                     4
                RMULT      111                                                5
        *                                                                     6
        *       DEFINE FUNCTIONS                                              7
        *                                                                     8
         5      FUNCTION   RN1,C24     EXPONENTIAL DISTRIBUTION FUNCTION      9
        0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38   10
        .8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2   11
        .97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8             12
         MEAN   FUNCTION   RN1,D5      DISTRIBUTION OF MEAN SERVICE TIME     13
        .18,450/.29,750/.61,1000/.85,1500/1,3000                             14
        *                                                                    15
        *       MODEL SEGMENT 1                                              16
        *                                                                    17
1               GENERATE   180,FN5,,,,2    CUSTOMERS ARRIVE                18
2               ASSIGN     1,FN$MEAN,5     SET P1 = SERVICE TIME           19
3               SELECT E   2,1,8,0,F,QUEUP IS A TELLER AVAILABLE?           20
4        LINE   QUEUE      P2              UPDATE SEGREGATED LINE DATA     21
5               QUEUE      10              UPDATE AGGREGATE LINE DATA      22
6               SEIZE      P2              CAPTURE A TELLER                23
7               DEPART     10              UPDATE AGGREGATE LINE DATA      24
8               DEPART     P2              UPDATE SEGREGATED LINE DATA     25
9               ADVANCE    P1              TRANSACT BUSINESS               26
10              RELEASE    P2              FREE THE TELLER                 27
11              TERMINATE                   LEAVE THE BANK                  28
12       QUEUP  SELECT MIN 2,1,8,,Q        SET P2 = NUMBER OF SHORTEST LINE29
13              TRANSFER   ,LINE           GO QUEUE UP                     30
        *                                                                    31
        *       MODEL SEGMENT 2                                              32
        *                                                                    33
14              GENERATE   180000          TIMER ARRIVES AFTER 5 HOURS     34
15              TERMINATE  1               SHUT OFF THE RUN                35
        *                                                                    36
        *       CONTROL CARDS                                                37
        *                                                                    38
                START      1               START THE RUN FOR DAY 1         39
                RMULT      333             SET RANDOM NUMBER SEQUENCE FOR DAY 2 40
                CLEAR                       CLEAR FOR DAY 2                 41
                START      1               START THE RUN FOR DAY 2         42
                RMULT      555             SET RANDOM NUMBER SEQUENCE FOR DAY 3 43
                CLEAR                       CLEAR FOR DAY 3                 44
                START      1               START THE RUN FOR DAY 3         45
                RMULT      777             SET RANDOM NUMBER SEQUENCE FOR DAY 4 46
                CLEAR                       CLEAR FOR DAY 4                 47
                START      1               START THE RUN FOR DAY 4         48
                RMULT      999             SET RANDOM NUMBER SEQUENCE FOR DAY 5 49
                CLEAR                       CLEAR FOR DAY 5                 50
                START      1               START THE RUN FOR DAY 5         51
                END        RETURN CONTROL TO OPERATING SYSTEM              52
```

(b) Multiple-Line Model

Figure 4C.2 Extended Program Listings for Case Study 4C

(7) Discussion

Model Implementation

In the section discussing the approach taken in building the model, the 24-point continuous Function XPDIS was referred to several times. Note in Figure 4C.2(a) and (b) that the Function has been named numerically, as "5", rather than symbolically. This is because of the restriction that when the optional C Operand is used with the ASSIGN Block, it must be the number of a Function, and cannot be a Function's symbolic name. At Block Number 2, then, "5" has been used as the ASSIGN Block's C Operand. In like fashion, "FN5" is used as the B Operand at the GENERATE Block in Location 1.

Program Output

Pertinent simulation output is summarized in Table 4C.4. The table indicates

	Average Queue Residence Time, Seconds[a]		Percent Zero Entries	
Day	One-Line	Multiple Line[b]	One-Line	Multiple-Line[b]
1	821	1114	29.4	39.8
2	2482	2588	27.9	34.3
3	551	832	43.2	50.6
4	3233	3161	11.6	16.6
5	2103	1956	15.7	22.2
Average:	1838	1930	25.6	32.7
Standard Deviation:	1012	874	11.1	12.1

[a] Including zero entries

[b] Aggregate Queue

Table 4C.4 Summary of Pertinent Statistics in Case Study 4C

that, for the one-line model, average waiting time for customers for the 5 days simulated was 1838 time units, or about 3.05 minutes. In the multiple-line results, average waiting time for the same 5 simulated days was 1930 time units, or about 3.2 minutes. For this experiment, there is then about a 5% time savings in converting from a multiple-line to a one-line system. These points should be noted, however.

(1) The sample size used in the experiment (5 days) is so small, and the observed superiority of the one-line system is so slight (about 5%), that further experimentation is required before firm conclusions can be drawn.

(2) The multiple-line case modeled here is unrealistic, because no line-switching is permitted. (The ability to incorporate the line-switching feature into queuing systems models will be discussed in Chapter 6.)

(3) Average waiting time is probably not a sufficiently complete measure of customer waiting experience. In addition to its average value, other properties of the random variable "waiting time" are also important determinants of a customer's feeling about the waiting process. For example, the statistic "percentage of customers who have to wait longer than 4 minutes" may be important; and, even if the one-line and multiple-line queuing systems produce the same average waiting time, one system may be superior to the other in terms of this other measure. The models presented in this case study do not estimate the waiting time distribution. The ability to incorporate such estimates in GPSS models is taken up at the end of this chapter.

STORAGE	CAPACITY	AVERAGE CONTENTS	AVERAGE UTILIZATION	ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENTS	MAXIMUM CONTENTS
TELRS	8	7.047	.880	989	1282.745	8	8

(a) Storage Statistics (One-Line Model, Day 1)

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
ONE	20	4.552	997	294	29.4	821.883	1165.601		8

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(b) Queue Statistics (One-Line Model, Day 1)

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
1	.969	133	1311.563	27	
2	.938	136	1241.867	6	
3	.913	140	1174.864	22	
4	.909	125	1309.455	16	
5	.848	142	1075.464	21	
6	.818	100	1473.250	12	
7	.729	105	1251.133	26	
8	.809	106	1375.216	13	

(c) Facility Statistics (Multiple-Line Model Day 1)

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
1	3	1.094	135	42	31.1	1459.473	2118.591		2
2	3	.973	138	56	40.5	1270.354	2137.914		2
3	3	.948	141	60	42.5	1211.460	2108.839		1
4	3	.869	126	51	40.4	1242.269	2087.013		1
5	2	.669	143	61	42.6	842.901	1469.938		1
6	2	.601	101	42	41.5	1072.395	1835.796		1
7	2	.502	106	53	50.0	853.631	1707.263		1
8	2	.508	107	56	52.3	855.037	1793.901		1
10	20	6.169	997	421	42.2	1113.842	1927.953		10

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(d) Queue Statistics (Multiple-Line Model, Day 1)

Figure 4C.3 Selected Program Output for Case Study 4C

(4) Table 4C.4 indicates that the standard deviation in the day-by-day average waiting time is large, ranging from about 45% (multiple-line system) to 55% (one-line system) of the average. This means that, if experimental conditions were not controlled, comparison of results would be all the more difficult. As it is, the multiple-line system "outperformed" the one-line system for two of the five days simulated (Days 4 and 5). Without experimental control, and with nearly-identical average waiting times and large standard deviations, the multiple-line system might have appeared consistently superior in a 5-day comparison, thereby producing misleading conclusions.

The Table 4C.4 information shows that, for the 5 days simulated, the customers had about a 25.6% chance to get a teller immediately in the one-line system, and a 32.7% chance in the multiple-line system. This seems inconsistent with the observation that the one-line system is superior. Careful thought leads to the conclusion, however, that the percentage zero entries in the multiple-line case should be higher than for the one-line system, precisely because no line-switching is simulated in the model. Reasoning this out is left to the reader as an exercise.

In Figure 4C.3(c), it is evident from the utilization statistics that customers have shown a bias in favor of using the lower-numbered Facilities. There is a simple reason for this. When one or more Facilities are available, the SELECT Block always selects the smallest-numbered Facility not in use. Similarly, when all Facilities are in use and there are ties for minimum Queue contents, the SELECT Block always selects the smallest-numbered Queue as having the "minimum" content. Hence, the smaller-numbered teller-Queue combinations are favored in the long run over those with larger numbers.

Figure 4C.3(c) also indicates that Facility 8 has markedly fewer captures than Facilities 1 through 7. Based on the SELECT Block logic just described, it is true that teller 8 gets business only when there are at least 7 customers already in the bank. The small capture count for Facility 8, then, "makes sense".

4.13 Computer Memory Considerations

When introducing Transactions, Facilities, Storages, and Queues, it was indicated that the maximum number of these entities permissible in a model depends on the amount of available computer memory. These maximum values are given in Appendix B, along with other information pertinent to computer memory considerations in GPSS. The portion of the Appendix B information applying to the GPSS entities discussed so far is repeated in Table 4.5.

Each entity has a "fixed" memory requirement, whether it is actually "used" in a model or not. These fixed requirements are shown in the second-to-last column in Table 4.5. For example, each Transaction has a fixed memory requirement of 16 bytes, each Storage has a fixed requirement of 40 bytes, and so on.

In addition, when certain items are used, additional memory must be committed to them. The amount of this additional memory is "variable" in the sense that it depends on the properties of the particular entity in question. For example, as footnote (a) in Table 4.5 indicates, each Transaction actually in a model requires 20 bytes, plus additional bytes for its Parameter set. Blocks and Functions also have a "variable" memory requirement. Facilities, Storages, and Queues, however, only have fixed memory requirements.

When an entity is used, its "variable" memory requirement, if any, is met by using portions of otherwise-uncommitted memory available in a pool called COMMON. The number of bytes of COMMON is dependent on the total available computer memory, as indicated in Table 4.5. The limitation on COMMON, coupled with its use to support

Entity Type	Normally Available Quantities for Various Computer Memory Levels 64K	128K	256K	Fixed Memory Requirement per Item, Bytes	Reallocation Mnemonic
Transactions	200	600	1200	16[a]	XAC
Blocks	120	500	1000	12[b]	BLO
Facilities	35	150	300	28	FAC
Functions	20	50	200	32[c]	FUN
Queues	35	150	300	32	QUE
Storages	35	150	300	40	STO
COMMON (bytes)[d]	5600	14400	25600		COM

[a] Add 20 bytes of COMMON for every active Transaction, plus additional bytes for Parameters (2 bytes per halfword Parameter, 4 bytes per fullword Parameter)

[b] Add 4 bytes of COMMON for each Block using more than one Operand

[c] Add 8 bytes of COMMON for each pair of coordinates of a D type Function. Add 12 bytes of COMMON for each pair of coordinates of a C type Function.

[d] COMMON is an otherwise-uncommitted pool of computer memory, portions of which can be used for such purposes as storing the pairs of points used to define Functions, providing the memory needed for Transaction Parameters, and so on.

Table 4.5 Repetition of A Portion of the Appendix B Information

entities actually being used in a model, sometimes results in the supply of COMMON becoming exhausted during a simulation. When this happens, error message 599, "LIMITS OF GPSS/360 COMMON CORE EXCEEDED", is printed out and the simulation is terminated.

Consider a typical situation which can result in the "COMMON exceeded" error message. Assume that a certain model consists only of Blocks, Facilities, Storages, and Queues. For simplicity, ignore the amount of COMMON which the Blocks may require. As suggested in Table 4.5, the Facilities, Storages, and Queues place no demands on COMMON. Now, if each Transaction uses 12 halfword Parameters (the default case), how many Transactions can be active in the model when 64K bytes of computer memory are available? From Table 4.5, it is easy to think that the answer is "200". This is not the case, because COMMON would be exceeded long before the 200th Transaction could be brought into the model. Footnote (a) in Table 4.5 indicates that each active Transaction with 12 halfword Parameters requires 44 bytes from COMMON (20 + 2*12 = 44). Dividing 44 into 5600 (the total number of bytes in COMMON at the 64K level) shows there is only enough COMMON to support the active existence of 127 Transactions. Hence, even though there can theoretically be 200 Transactions at the 64K level, this can only come about in practice if the analyst adjusts the number of Transaction Parameters (or takes other steps described below) so that enough COMMON is available to support this many Transactions.

It is good practice, then, to use the F-Operand on GENERATE Blocks to limit Transaction Parameters to the number actually required. This was done in Case Study 4C (see Cards 22 and 18 in Figures 4C.2(a) and (b), respectively). If the number of Transaction Parameters had not been scaled down, the Case Study 4C models would not have run at the 64K memory level. In both models, conditions arose in which COMMON would otherwise have been exceeded while trying to bring another customer into the bank (i.e., while the Processor was trying to bring another Transaction into the model).

It may be, however, that a "COMMON exceeded" condition arises even when active Transactions only carry exactly the number and size of Parameters they require. When this happens, more COMMON can be obtained simply by going to a higher K-level (assuming a higher K-level is available).[4e] The disadvantage of this approach, however, is that models are more expensive to run at higher K-levels, for the simple reason that more computer memory is required to process the job.

Another way to solve the "COMMON exceeded" problem is to reallocate the use of computer memory at a given K-level. The "maximum normal quantities" shown in Table 4.5 (or, more generally, in Appendix B) are default quantities. By use of a special Control Card, the REALLOCATE Card, the analyst can redistribute available computer memory among the various entity types within a given K-level. For example, suppose that a given model at the 64K level makes use of only 2 Storages. The fixed memory requirement of each Storage is 40 bytes, whether the Storage is being used or not. Hence, with only 2 Storages in use, 1320 bytes of memory are being "wasted" (40 bytes times 33 unused Storages equals 1320). If the analyst re-defines the maximum number of Storages as 2, 1320 bytes are thereby "freed". In the absence of other considerations, these bytes could be added to COMMON. There would then be 6920 bytes in COMMON for the model at the 64K level, an increase of almost 25%.

Quite apart from the constraint imposed by COMMON, a model's need for a given type of entity or entities may exceed the "normally available quantities" at a given K-level. When this is true, the "maximum quantity" problem can be solved by following either approach available to solve the "COMMON exceeded" problem. If the option exists, it is possible to go to a higher K-level, although the disadvantage of increased run-cost should be kept in mind. The second approach, reallocation of the "maximum quantity" of entities within a given K-level, should be used before resorting to a higher K-level. In the above example, with only 2 Storages used in the model, 1320 bytes were freed via reallocation. These bytes could be added to COMMON. Suppose, however, that COMMON does not impose a constraint in the model in question. Instead, assume the maximum quantity of Transactions, 200, is a constraint. If it is necessary that up to 250 Transactions be in the model simultaneously, the additional 50 Transactions required can be "bought" with some of the 1320 bytes freed through the Storage reallocation. Because each Transaction has a fixed memory requirement of 16 bytes, the maximum quantity can be reallocated upward to 250, using 800 of the freed bytes in the process. The other 520 freed bytes might then be added to COMMON. By reallocating downward on some entities, then, it is possible to reallocate upward on other entiries. In summary, considerable flexibility can be achieved through the reallocation process.

Specifications for the REALLOCATE Control Card are shown in Table 4.6. The Location field is not used. The word REALLOCATE is entered in the Operation field. Pairs of entries are used in the Operands field. The first entry in a pair is a mnemonic for the entity type being reallocated. The second entry is the new "maximum quantity" to be in effect. Mnemonics for the Table 4.5 entities are shown in the right-most column in Table 4.5

(4e) The K-level desired by the analyst is indicated on the External Control Cards used in making up the job deck. See Section 18 in Chapter 2.

Punchcard Field	Information Supplied in Field
Location	Not used
Operation	Literally, the word REALLOCATE
Operands	
A,C,E,G, etc.	Each alternate Operand is a mnemonic indicating a type of entity whose maximum normal quantity is being re-defined; a partial list of mnemonics is shown in Table 4.5; a complete list appears in Appendix B
B,D,F,H, etc.	Each alternate Operand is a constant indicating the maximum normal quantity to be in effect for the entity type indicated by the preceding Operand

Table 4.6 Specifications for the REALLOCATE Control Card

Examples of REALLOCATE Cards appear in Figure 4.20. The A and B Operands in the

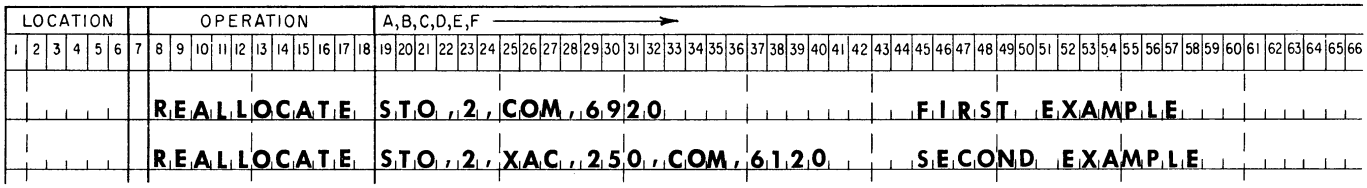


Figure 4.20 REALLOCATE Card Examples

first example specify that the number of Storages is to be reallocated downward to 2. As discussed above, if the model is being run in a 64K environment, 1320 bytes are freed by this reallocation. The C and D Operands in the first example specify that COMMON is to be reallocated upward to 6920 bytes. Hence, all of the bytes freed by the Storage reallocation are being added to COMMON.

The first example shows that, although reallocation specifications are provided pair-wise on the REALLOCATE Card, Operand pairs are not separated by a slash (/) as are pairs on Function Follower Cards and Storage Capacity Definition Cards. Only commas are used on the REALLOCATE Card to set apart consecutive Operands.

It is also implied in the first example that, if the bytes freed by downward reallocation are to be added to COMMON, this must be explicitly indicated through use of the COM mnemonic. It is not true, then, that any freed bytes not otherwise "used" by upward reallocation are "automatically" added to COMMON.

In the second example, the number of Storages is reallocated downward to 2. Assuming a 64K environment, the 1320 freed bytes are then used in part to reallocate upward to 250 Transactions (Operands C and D in the example). This commits 800 of the freed bytes. The remaining 520 bytes are added to the 5600 otherwise available in COMMON, producing a total of 6120 (Operands E and F in the example).

Three final observations concerning use of the REALLOCATE Card will now be made.

(1) As many Operand pairs can be entered in a REALLOCATE Card as desired, as long as the pairs do not extend beyond card column 72. More than one REALLOCATE Card can be used, if necessary or desirable.

(2) The REALLOCATE Card (or Cards) must be placed ahead of the SIMULATE Card in the job deck.

(3) It is recommended that the minimum normal quantity specified for any entity type be 1, not 0. Although 0 is logically a feasible minimum quantity, a 0 specification may result in an error condition because of the way the GPSS Processor has been constructed.

4.14 Computer Time Requirements

Until now, nothing has been said about the computer time required to run GPSS models. The time spent by the computer's Central Processing Unit (i.e., CPU time) in performing a simulation is important in a variety of ways. For example, it has direct dollar consequences. CPU time might be billed at rates ranging from 15¢ to $1.00 per second, depending on the hardware involved, and the context in which the hardware is used.[4f] Apart from cost considerations, the analyst must often estimate the maximum CPU time required to run a job. During the run, if that much CPU time has been used and the job is not yet done, the job is usually "time trapped" off the system. This is to be viewed in a positive sense; if it is established during a run that more time is required than was estimated, this may be a signal that something is wrong with the logic or implementation of the model. On the other hand, when the analyst is faced with deadlines of one kind or another, it is disappointing to find that the time estimate was set unrealistically low and, even though the model was performing as expected, less than complete results are available from the run. This is especially vexing when "turnaround time" (i.e., the time between turing in a job, and having it ready for pickup) is hours, rather than minutes.

With one or two exceptions, the simulations discussed so far in the book have each required fewer than 10 CPU seconds.[4g] This single CPU time measure is the sum of two major components. Each of the major components consists of sub-components, as itemized in Table 4.7. As an example of the relative contribution of the sub-components to the final CPU-time measure, Table 4.8 shows the size of the components when the Figure 4C.2(a) model was run. Execution time is usually the most significant time component, as Table 4.8 shows.

Major Component	Sub-Components
GPSS Processor Time	Assembly Time
	Input Time
	Execution Time
	Output Time
Operating System Time	Log in
	Load the GPSS Processor
	Unload the GPSS Processor
	Log out

Table 4.7 CPU-Time Components Involved in Running a GPSS Model

---

[4f] For example, the model in Figure 4C.2(a) used 48.7 CPU seconds on Michigan's IBM 360/67 multiprocessing system. Cost of the run was $6.65. Cost per CPU second, then, was about 15¢ for the run. This is admittedly a coarse computation, because Michigan's costing algorithm takes into account use of system resources other than CPU time alone. As another example, the author knows of an industrial user who pays $1.00 per CPU second to run GPSS in "remote batch" mode on a Univac 1108 computer. Of course, the figures of 15¢ and $1.00 cannot be compared directly, because about 1.5 to 2.0 times as much computation can be accomplished in 1 second on the Univac 1108 as on the IBM 360/67. Furthermore, the code for the Univac 1108 GPSS Processor is more efficient than it is for the GPSS/360 Processor. Such efficiency differences must also be factored in when comparing cost figures.

[4g] All the models in this book were run on the University of Michigan's IBM 360/67 multiprocessing system.

Major Component or Sub-Component	CPU Time, Seconds
Assembly	0.375
Input	0.421
Execution	44.731
Output	0.648
Operating System Time	2.534
Total CPU Time:	48.709

Table 4.8 CPU Time Breakdown for Running the Figure 4C.2(a) Model

Table 4.9 shows the total CPU times required to run the 12 GPSS case studies presented so far. From now on, in the "Program Output" section of each case study, a

Case Study	Total CPU Time, Seconds
2A	1.6
2B	1.8
2C	2.1
2D	2.0
2E	68.2
2F	4.1
3A	4.0
3B	3.2
3C	8.1
4A	4.8
4B	5.2
4C	86.5 (both models included)

Table 4.9 Total CPU Times for Case Studies Presented So Far

footnote will be used to indicate the total CPU time required to perform the case study simulation.

It was indicated in Section 2.8 that those using GPSS/360 on a System 360/67 computer can estimate the CPU time required for a simulation by using "1 millisecond per Block execution" as a rule of thumb. The 1 millisecond figure is a rough approximation at best. Actual execution time depends on the types of Blocks involved, the nature of the Functions used, the number and type of blocking conditions experienced by Transactions, the number of scans and re-scans of the Current Events Chain, and so on. As a "test" of this measure, consider again the Figure 4C.2 model. Using Block Counts not shown in the book, the "Total Count", summed over each Block, and over each of the 5 days simulated, comes to about 61,000. This "total Total Count" can be interpreted as the number of Block executions performed. Dividing 61,000 into the 48.7 CPU seconds used, a figure of about 0.8 milliseconds per Block execution results. In this case, then, the 1 millisecond rule of thumb tends to be on the conservative side.

If it is necessary to develop a rule of thumb for another GPSS Processor and/or another hardware system, a convenient approach is to run several widely different models, then use the "total Total Count" as just illustrated to arrive at a rough approximation for estimation purposes. When such a figure is available, it is possible to forecast, albeit in crude fashion, about how much CPU time a previously-untried model can be expected to consume.

In Chapter 6 of this book, methods for building "relatively efficient" GPSS models will be considered. If CPU time is expensive or limited, designing models so that they require as little CPU time as possible is obviously a matter of some interest.

4.15 A Note on Possible Redundancies in GPSS Modeling

During the model input phase, one of the tasks of the GPSS Processor is to assign each symbolically named entity in the model a corresponding numeric name. For the entity in question, the numeric name is then substituted for the symbolic name throughout the model. This is done on an internal basis and is something to which the analyst may give little thought, if any.

Of course, the correspondences established by the Processor between symbols and numbers appear in the model output in the form of symbol dictionaries. The various symbol dictionaries are labeled "BLOCK SYMBOLS AND CORRESPONDING NUMBERS","FACILITY SYMBOLS AND CORRESPONDING NUMBERS", "FUNCTION SYMBOLS AND CORRESPONDING NUMBERS", and so on. An example of a dictionary for symbolic Function names appears in Figure 4B.3(a).

It is important to understand how the Processor establishes correspondences between symbols and numbers in a model. A two-step approach is followed.

(1) For the entity type in question (e.g., Facilities, or Queues, or Functions, etc), the model is first scanned to find all such entities which have been named numerically by the analyst. These numbers are then understood to be already committed in the model. In particular, they are not available for being made the numeric equivalent of symbolically-named entities of that type.

(2) For the same entity type, the model is then scanned to find all such entities which have been named symbolically by the analyst. The scan of the model takes place from the front of the deck toward the back. As each not-previously-encountered symbol is found, its numeric equivalent is defined as the smallest number not yet committed to that entity type. Of course, as each equivalence is established, the number involved is removed from the list of numbers not yet committed.

As an example of this two-step approach, assume that Figure 4.21 shows the sequence of different Facility references as they are encountered (via SEIZE Cards) in

```
SEIZE  5
SEIZE  3
SEIZE  BETA
SEIZE  2
SEIZE  ALPHA
SEIZE  1
SEIZE  GAMMA
```

Figure 4.21 The Relative Order of Facility References as They Appear in a Hypothetical GPSS Model

a certain GPSS model. The references are shown in the order in which the SEIZE Cards appear in the model deck, from front to back. When the model is first scanned to find numeric references to Facilities, the numbers 5, 3, 2, and 1 are removed from the "not yet committed" list. The order of Facility numbers "still available" is then 4, 6, 7, 8, and so on. When the model is then subsequently scanned to find symbolic references to Facilities, the symbols BETA, ALPHA, and GAMMA are encountered, in that order. These symbols are consequently made equivalent to the numbers 4, 6, and 7, respectively.

In the Figure 4.21 example, the Facilities are directly specified in all cases, either numerically or symbolically. Now consider an example in which some Facilities are directly specified, whereas others are specified indirectly. Such an example is shown in Figure 4.22. In Figure 4.22, there is a potential problem in referencing the Facility (or Facilities) whose number is the value of P1. The problem is that, during

```
SEIZE  DELTA
SEIZE  7
SEIZE  P1
SEIZE  8
SEIZE  BETA
```

Figure 4.22 The Order of Facility References in a Hypothetical Model in Which Indirect Specification is Used

the model input phase, the GPSS Processor cannot possibly know what values P1 will have during the simulation. As a result, no correspondences can be drawn during the input phase between "P1" and corresponding Facility numbers. Following the usual two-step approach, the numbers 7 and 8 are first removed from the "still available" list. Then, DELTA and BETA are made equivalent to 1 and 2, respectively. No steps are taken with respect to "Facilities whose numbers are the value of P1".

Now suppose that the model is so designed that P1 can take on the values 1, 2, and 3 during the simulation. When P1 is 1, "SEIZE P1" is then equivalent to "SEIZE DELTA". Similarly, when P1 is 2, "SEIZE P1" is equivalent to "SEIZE BETA". In all likelihood, this redundancy is probably not what the analyst had in mind. If this was not the analyst's intention, then the model is invalid for all practical purposes. The analyst's thinking probably went something like this: "I have Facilities numbered 7 and 8 in the model. Furthermore, by design, the various values P1 can assume at the "SEIZE P1" Block are 1, 2, and 3. This means that I also have Facilities numbered 1, 2, and 3 in the model. The GPSS Processor will therefore make the Facility symbols DELTA and BETA equivalent to 4 and 5, respectively." The analyst thinks, then, that there are seven different Facilities in his model, whereas there are only five.

There are several ways to sidestep the problem just described.

(1) All Facilities in a model can be named directly, either numerically or symbolically. The disadvantage of this approach is that the power of indirect specification is thereby lost.

(2) Symbolic Facility names can be entirely avoided. If all Facility references are either numeric, or via indirect specification, the potential problem is avoided. In this approach, the memory-aiding convenience of using symbolic names is foregone.

(3) When both symbolic naming and indirect specification are used, the analyst can build his model so that, by design, the Facilities referenced indirectly have large numeric values. When the Processor makes symbolic names equivalent to numbers, then, the relatively smaller numbers on the "still available" list will be distinct from those that arise via indirect specification.

(4) Finally, the analyst can force the Processor to equate symbolic names with user-selected numbers. By doing this, unwanted redundancies can be avoided. The desired relationships between symbolic names and their numeric equivalents can be specified by use of the EQU Card. This card is explained in the GPSS/360 Users Manual. The need for it in fundamental GPSS modeling is rare. As a result, it will not be described or used in this book.

The above discussion has centered around possible redundancies that might arise relative to Facilities. Of course, the remarks made for Facilities apply to other entities as well, i.e., to Storages, Queues, and Functions, for example. Although they occur infrequently in fundamental GPSS modeling, the possibility of such redundancies should be kept in mind.

4.16 Exercises

4.16.1 Show SELECT Blocks to accomplish the following.

(a) Find the first Queue among Queues 2 through 7 with a zero-entry count of zero. Place the Queue number in Parameter 7 of the selecting Transaction, which is to continue to the sequential Block whether or not such a Queue is found.

(b) Find the first Storage among those numbered from 8 through 11 whose current content is less than 3. Place the Storage number in Parameter 1 of the selecting Transaction. If no such Storage is found, the Transaction is to move to the non-sequential Block at the Location RUTE4.

(c) Find the first Facility in the range from 1 to 5 which has been captured more than 10 times. Parameter 7 of the selecting Transaction indicates the Parameter in which to place the result. If no Facility meets the specified condition, the selecting Transaction should go to the Block in the Location BYPAS.

(d) Find the Facility in the range from 4 to 9 with minimum utilization. Place the Facility number in Parameter 1.

(e) Find the Storage in the range from 2 to 3 with maximum remaining capacity The Storage number is to be put in Parameter 3.

(f) Parameters 1 through 10 of a Transaction contain order quantities. The number of the Parameter containing the minimum order quantity is to be placed in Parameter 12 of the Transaction.

(g) Find the Queue in the range from P1 to P2 which has the minimum average residence time (including zero entries). Place the number of the Queue in Parameter 5. (All three Parameters mentioned belong to the Transaction entering the SELECT Block.)

4.16.2 Explain what happens when Transactions enter the various SELECT Blocks whose card images are shown below.

(a) SELECT E 5,3,7,5,Q,PATHA

(b) SELECT MIN P5,1,10,,FR

(c) SELECT GE 5,P1,P2,P3,SC

4.16.3 (a) Describe the two methods whereby the "normally available quantities" of GPSS entities can be increased in a model. Given a choice, which of the two methods is probably preferable, and why?

(b) At the 64K level, how many additional Storages can be "bought" if the number of Facilities is reallocated downward to 20? Show a REALLOCATE Card which makes the necessary downward and upward reallocations to accomplish this. State any assumptions you may have to make.

(c) At the 64K level, how many Transactions can be in a model if each Transaction has 10 fullword Parameters? Assume that the REALLOCATE Card has not been used, and that no other entities in the model require memory from COMMON. (Also show a GENERATE Block through which Transactions enter a model with 10 fullword Parameters.)

(d) Repeat (c), only now assume that the number of Blocks has been reallocated downward to 75, and the number of Functions to 5. Show the REALLOCATE Card which accomplishes these downward reallocations, as well as the corresponding upward reallocation for COMMON.

(e) How much COMMON is required by the Function XPDIS, shown in Figure 3.22?

4.16.4 (a) Submit the model in Figure 4C.2(a) for running on your computer. Compare the total CPU time required on your system with the 48.7 seconds required on an IBM 360/67. Also compute the "total Total Count" for your simulation, and use it to compute a "millisecond per Block execution" figure.

(b) Compute "total Total Counts" and the corresponding "millisecond per Block execution" statistics for any other three GPSS models which you have run. How much variation is there from model to model in the "millisecond per Block execution" statistic?

4.16.5 Explain the reason why, in Case Study 4C, the percent zero entries in the multiple-line system is larger than in the one-line system.

4.16.6 Figure P4.16.5 shows the sequence of different Queue references as they are encountered (via QUEUE Cards) in a certain GPSS model. The references are shown in the order in which the cards appear in the model deck, from front to back.

```
QUEUE  GAMMA
QUEUE  1
QUEUE  4
QUEUE  BETA
QUEUE  3
QUEUE  6
```

Figure P4.16.6

To what numbers are the symbols GAMMA and BETA made equivalent?

4.16.7 Figure P.4.16.7(a) shows the sequence of different Facility references in a model, in the order in which the Processor encounters them in the model deck. Figure P4.16.7(b) shows analogous information for Queues in the same model. As a Facility, what is the numeric equivalent of DELTA? As a Queue, what is the numeric equivalent of DELTA?

```
SEIZE  1                QUEUE  3
SEIZE  DELTA            QUEUE  2
SEIZE  2                QUEUE  DELTA
```

(a) (b)

Figure P4.16.7

4.16.8 See if you can develop or find an analytic solution for either queuing system considered in Case Study 4C. If you are successful, compare and contrast the analytic results with those produced in the case study.

4.16.9 Show how to modify the multiple-line model in Case Study 4C so that only one SELECT Block need be used. (Hint: in the case study, the customer first scans the tellers to find if one is immediately available; if not, the customer then scans the Queues to find the one with minimum content. If care is taken, the first of these SELECT Blocks can be eliminated.)

4.16.10 In Case Study 4C, suppose that service times are not exponentially distributed, but have been found to follow the distributions shown in Table T4.16.10. (Note: See Section 3.12 for an explanation of the interpretation to be given to Table T4.16.10.)

Service Time, Seconds	Cumulative Frequency as a Function of Various Business Categories				
	1	2	3	4	5
25	0.0				
50	.10				
75	.68	0.0			
100	1.0	.14	0.0		
125		.39	.18		
150		.72	.42	0.0	
175		1.0	.58	.21	
200			.83	.58	0.0
250			1.0	.79	.09
300				1.0	.54
350					.81
400					1.0

Table T4.16.10

(a) Show how to modify the models in Case Study 4C to take these service time into account. Experimental conditions should still be controlled in the same sense that they were in the case study.

(b) Assume now that the customer arrival rate increases from 200 to 240 per hour, but that a Poisson arrival process is still in effect. Simulate with the multiple-line model for the equivalent of an 8-hour day in trial-and-error fashion to determine the smallest number of tellers required if average teller utilization is to be 90% or greater. Before making any runs, develop an intelligent plan to follow in performing the trial-and-error investigation.

(c) Using the number of tellers determined from (b), run the Quickline version of the model. Compare and contrast system performance for the multiple-line and Quickline alternatives.

4.16.11 In Case Study 4C, the question was raised whether conflicts could arise if only one continuous 24-point Function were used to simulate both the Poisson arrival and the exponential servers. It was suggested that if there were any doubts about using only one source of random numbers, "two distinct 24-point Functions could be used in the model, each with its own 'private' random number source as argument".

(a) Show how to modify the models in Figure 4C.2 so that two distinct random number sources are used to simulate inter-arrival times and service times, respectively.

Your models in (a) are not valid if it is true that the same random number value is used to determine both a customer's inter-arrival time, and his service time. For example, suppose you use RN1 to determine inter-arrival times, and RN2 to determine service times. Referring to Table 3.1, a value of .000573 is used to compute the arrival time of customer 1 (first call on RN1). Similarly, Table 3.1 indicates that a value of .000573 is used to compute the service time requirement of customer 1 (first call on RN2). The result is one of correlation between customer 1's inter-arrival-time and his service time. The same remark holds for each customer entering the bank. Customers with small inter-arrival times (because of small RN1

values) have small service times (because of the correspondingly small RN2 values); and customers with large inter-arrival times have correspondingly large service times. Such a relation between inter-arrival and service times is of course unrealistic.

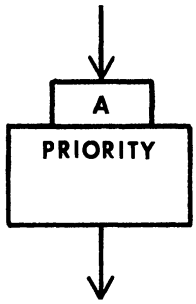
The problem of correlation did not arise in Case Study 4C, because a single source of random numbers was used to simulate inter-arrival and service times. It is also possible to avoid correlation when using separate sources of random numbers to simulate inter-arrival and service times. It is only necessary to use the RMULT Control Card to change the starting point of one of the random number generators.

(b) Re-run Case Study 4C, using two distinct random number generators and deliberately letting inter-arrival and service times be correlated in the sense described above. Contrast the results with those in Case Study 4C. Do the correlated experiments tend to favor the one-line system even more than the results in the case study? Would this be expected intuitively? Explain why or why not.

4.17 Modification of a Transaction's Priority Level: The PRIORITY Block

When a Transaction enters a model, its Priority Level is specified through the E Operand at its GENERATE Block. In all applications discussed so far, each Transaction has retained this initial Priority Level throughout its existence in the model. It is possible, however, to dynamically change a Transaction's Priority Level as a simulation proceeds. Altering a Priority Level has important implications, of course, in terms of the position a Transaction occupies when it is on the Current Events Chain. This, in turn, influences the chronological sequence in which various Transactions are moved forward in a model. Hence, the ability to dynamically change a Transaction's Priority Level can be of major consequence in GPSS modeling.

Whenever a Transaction moves into the PRIORITY Block, its Priority Level is changed. The PRIORITY Block, and its A Operand, are shown in Figure 4.23. When a Transaction moves



Operand	Significance	Default Value or Result
A	The value to be assigned as the Priority Level of Transactions which enter the PRIORITY Block	Error

Figure 4.23 The PRIORITY Block and Its A Operand

into the PRIORITY Block, the value of the A Operand is assigned as the Transaction's new Priority Level. Of course, if the "new value" happens to be identical to the "old value", no change in the Priority Level occurs.

Transactions are never denied entry to the PRIORITY Block. When such a Block is entered, this sequence of events occurs.

(1) The new value of the Transaction's Priority Level is assigned.

(2) The Transaction's location on the Current Events Chain is changed. The Transaction is put behind all other Current Events Chain Transactions in its new priority class.

(3) The Processor then continues moving the Transaction forward in the model until it comes to rest.

(4) Finally, after the Transaction comes to rest, the Processor re-starts its scan of the Current Events Chain. There is a reason for this. At the PRIORITY Block, the Priority Level of the Transaction may have been made smaller. The result would be to shift both the Transaction and the point of the Current Events Chain scan to a position toward the back of the chain. If the scan simply continued from that point, intermediate Transactions on the Current Events Chain might be bypassed in the process. Re-starting the scan means that this possibility is avoided.

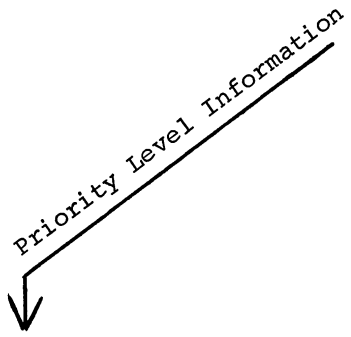
On occasion, the analyst may want to have the Current Events Chain scan re-started immediately after the PRIORITY Block subroutine has been executed. This can be accomplished by using the word BUFFER as the B Operand on the PRIORITY Block. When this

BUFFER option is used, steps (1) and (2) above take place, and the scan is then re-started. As the scan proceeds, it eventually encounters the Transaction still at the PRIORITY Block. At that time, the Transaction continues its normal forward movement in the model. Use of the BUFFER option arises in the next case study.

The Priority Level of a Transaction is a Standard Numerical Attribute in GPSS. The name of the attribute is PR. The name PR in itself constitutes a complete data reference, because a Transaction only has one PR value. Whenever PR is used to indirectly specify data, the Priority Level of the active Transaction is used as the data.

As previously explained, PR (Priority Level) can assume integer values from 0 to 127, inclusive. Because it is a Standard Numerical Attribute, PR can be used as Block Operands, Function arguments, and so on, in the model-building process.

When the Current and/or Future Events Chains are printed out, the Priority Level of Transactions on those chains appear as part of the printout. A portion of the Figure 4B.4 Future Events Chain printout is repeated in Figure 4.24. As pointed out in the Figure,



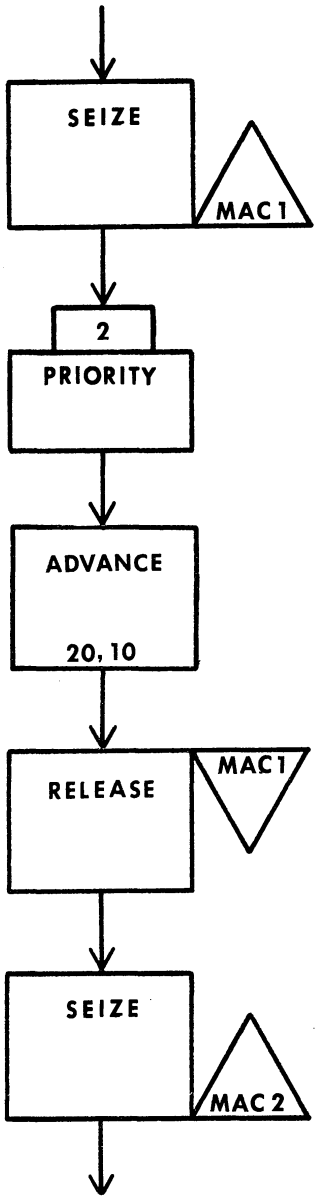
FUTURE EVENTS CHAIN

TRANS	BDT	BLOCK	PR	SF	NBA	SET	MARK-TIME	P1
16	28806	4			5	16	28632	0
								0
								0
12	28830	4			5	12	28766	0
								0
								0
7	28836	10			11	7	28380	8
								0
								0
15	28840				1	15	-366	0
								0
								0
3	57600				20	3	-1	0
								0
								0

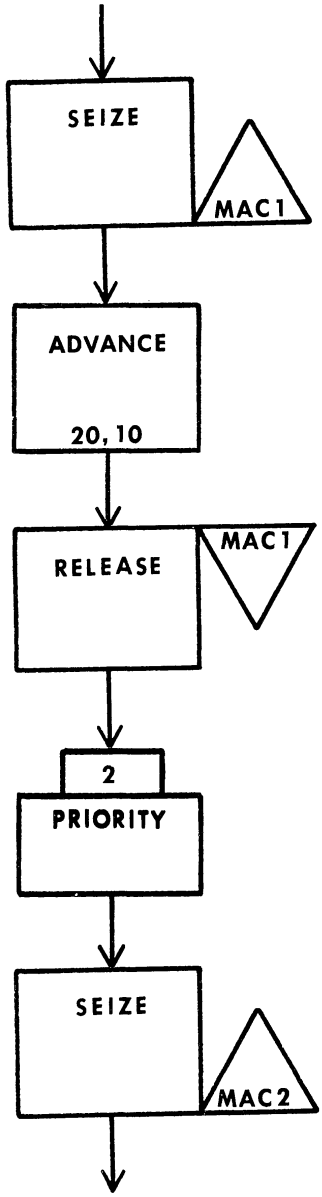
Figure 4.24 Repetition of a Portion of the Figure 4B.4 Future Events Chain

Column 4, labeled PR, contains the Priority Level information. All of the column 4 entries in Figure 4.24 are "blank", indicating that the Transactions have a Priority Level of zero.

In concluding this section, an example will now be given showing placement of the PRIORITY Block in a model. Suppose that a Transaction represents a part on which work is performed at various machines in a shop. One machine is simulated with the Facility MAC1. Another machine is simulated with the Facility MAC2. Work is to be performed on the part at MAC1 first, then at MAC2. The part is to have a Priority Level of 3 for use of MAC1; its Priority Level for use of MAC2 is to be 7. Figure 4.25 shows



(a) A Valid Block Sequence



(b) An Invalid Block Sequence

Figure 4.25 Valid and Invalid Examples for Placement of the PRIORITY Block in a Model

two possible Block sequences which have been proposed to produce the desired effect. It is assumed that the Priority Level of the Transaction in question is 3 when the Transaction moves into either one of the proposed sequences.

In the Figure 4.25(a) sequence, the Transaction has its Priority Level stepped from 3 to 7 after it has succeeded in capturing MAC1, but before it moves into the ADVANCE Block where the time required to use MAC1 is simulated. Of course, when the Transaction enters the ADVANCE Block, it is hooked onto the Future Events Chain. When it is later returned to the Current Events Chain (scheduled to release MAC1), it is put behind other CEC Transactions with a Priority Level of 7. As a result, it is in a "first-come, first-served, within Priority Level" position relative to its use of MAC2. This is as it should be.

In the Figure 4.25(b) sequence, the Transaction is immediately put on the Future Events Chain after it captures MAC1. When it is later brought back to the Current Events Chain (scheduled to release MAC1), it is still a member of Priority Class 3. Only after the CEC scan reaches Priority Level 3 Transactions will it release MAC1, have its Priority Level stepped to 7, and attempt to capture MAC2. The Transaction does not compete for MAC2, then, until all Transactions of Priority Level 6, 5, and 4 (and possibly some of Priority Level 3) have been processed. If one of these other Transactions captures MAC2, the intended logic has been violated. The Figure 4.25(b) sequence consequently can be invalid.

In the context of this example, there is an easy conclusion to be reached. When a Transaction is to have its Priority Level changed in moving between two consecutive servers, the change should be made immediately after capturing the first of the servers, before use of that server is simulated with the ADVANCE Block.

4.18 Case Study 4D: Dynamic Priority Distinctions in a Job Shop

(1) Statement of the Problem

A certain small job shop uses only two different types of machines. It has 4 machines of type A, and 6 machines of type B. Three different categories of jobs move through the job shop. Each job requires machining on a type A machine and a type B machine. Whether a job visits a type A machine first, then a type B, or visits the machines in the reverse sequence, depends on the job's category. Table 4D.1 shows the "machine usage sequence" for the three categories of jobs involved. As indicated, jobs in category

		Type of Machine Used	
		First	Second
Job Category	1	A	B
	2	A	B
	3	B	A

Table 4D.1 Machine Usage Sequences

1 or 2 each visit an A machine first, then a B machine. Jobs in category 3 visit a B machine first, then an A machine.

Processing times at the two types of machines are exponentially distributed, with means depending on the category of job involved. Table 4D.2 shows these mean processing times for the various possible combinations of machine type and job category.

		Machine Type	
		A	B
Job Category	1	60	35
	2	40	75
	3	20	55

Table 4D.2 Mean Processing Times, Minutes

A job arrives at the shop every 10 minutes, on average, in a Poisson pattern. Job categories 1, 2, and 3 constitute 30%, 50%, and 30%, respectively, of the arriving jobs.

The queue discipline used in the shop is "shortest imminent operation". That is, when jobs of two or more types are waiting for a machine which has just become available, that job whose expected processing time is the smallest is put on the machine next. Queue discipline is first-come, first-served within each job category.

Build a GPSS model for the job shop. Design the model to estimate the average residence time in the shop for each job category. Also gather average utilization statistics for the group of 4 machines of type A, and the group of 6 machines of type B. Simulate with the model until 1,000 jobs have been completed. Output statistics not only at the end of the simulation, but also at times corresponding to 250, 500, and 750 job completions.

(2) Approach Taken in Building the Model

The model could be built in three separate segments. Jobs in categories 1, 2, and 3 would move through the first, second, and third segments, respectively. In such an approach, all Block Operands would be directly specified, as constants. Although this approach would require more Blocks than might be necessary, it would have the virtue of simplicity. Instead of following such a simple strategy, the approach taken here will be to build a compact model consisting of a single Block sequence. The simpler approach is left to the exercises.

The model core will be ENTER-ADVANCE-LEAVE-ENTER-ADVANCE-LEAVE. A Transaction moving through this six-Block sequence simulates a job visiting each of the two machine groups. Of course, the six-Block sequence must be embellished to make the visitation order depend on the type of job involved, and to make the job's Priority Level at each group depend on its mean processing time. The following approach can be taken.

Jobs entering the shop are tagged with a value of 1, 2, or 3 in Parameter 1 by sampling from a job-category distribution. Hence, the P1 value can be directly interpreted as the type of job represented by the corresponding Transaction. This Parameter, in turn, can be used as Function arguments (a) to tag each job with its proper relative Priority Level ahead of each machine group, and (b) to reference the proper machine group (type A or type B machines) at the first ENTER-LEAVE Block pair, and again at the second ENTER-LEAVE pair.

The only remaining task is to arrange for an estimate of job residence time in the model. This is accomplished by passing Transactions through a QUEUE Block when they enter the model, and through a DEPART Block just before they leave the model. If the Queue is referenced through a Transaction's first Parameter, then Queues numbered 1, 2, and 3 will maintain statistics for jobs in categories 1, 2, and 3, respectively, per the Parameter-tagging scheme just described.

(3) Table of Definitions

Time Unit: 0.1 Minutes

GPSS Entity	Interpretation
Transactions	Jobs Parameter Use P1: Code for Job Category

Value	Job Category Indicated
1	1
2	2
3	3

Functions	
FIRST	Function indicating the number of the Storage simulated at the first ENTER Block; depends on the category of job involved
JTYPE	Function describing the distribution of job categories
SECON	Function indicating the number of the Storage simulated at the second ENTER Block; depends on the category of job involved
TAG1, TAG2	Functions indicating a job's Priority Level for the first and second machines it uses, respectively
TIME1, TIME2	Functions indicating a job's mean processing time at the first and second machines it uses, respectively
XPDIS	Function for sampling from the exponential distribution with a mean value of 1
Storages	
10, 11	Storages used to simulate the groups of type A and type B machines, respectively

Table 4D.3 Table of Definitions for Case Study 4D

(4) Block Diagram

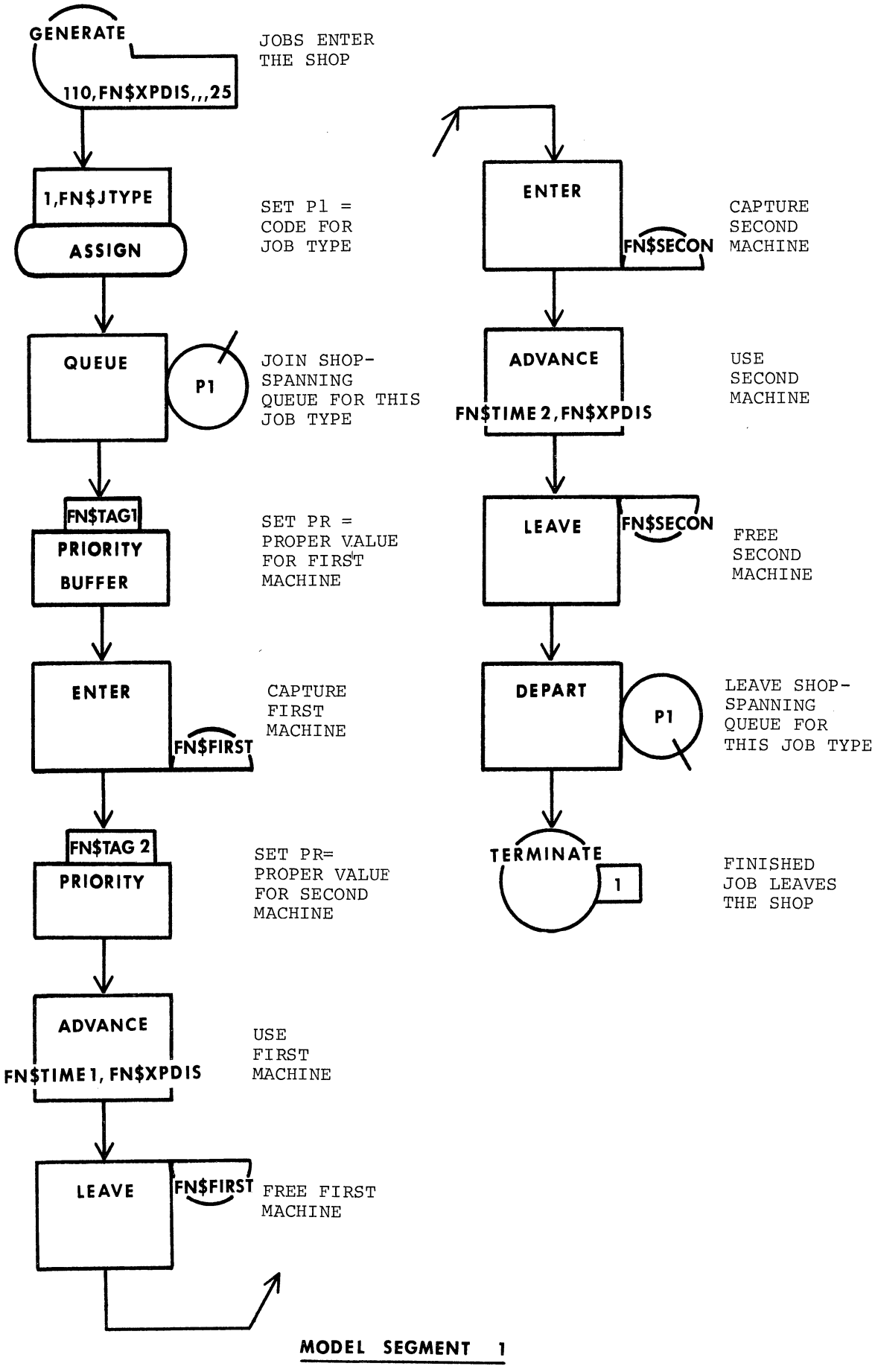


Figure 4D.1 Block Diagram for Case Study 4D

(5) Extended Program Listing

```
BLOCK                                                                    CARD
NUMBER  *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS                NUMBER
               SIMULATE                                                     1
        *                                                                   2
        *      DEFINE FUNCTIONS                                             3
        *                                                                   4
         FIRST FUNCTION   P1,D3       GIVES NO. OF 1ST MACHINE JOB "P1" USES 5
        1,10/2,10/3,11                                                      6
         JTYPE FUNCTION   RN1,D3      DISTRIBUTION OF JOB-TYPES             7
        .2,1/.7,2/1,3                                                       8
         SECON FUNCTION   P1,D3       GIVES NO. OF 2ND MACHINE JOB "P1" USES 9
        1,11/2,11/3,10                                                      10
         TAG1  FUNCTION   P1,D3       SETS PRIORITY FOR JOB AT ITS 1ST MACHINE 11
        1,5/2,6/3,6                                                         12
         TAG2  FUNCTION   P1,D3       SETS PRIORITY FOR JOB AT ITS 2ND MACHINE 13
        1,7/2,5/3,7                                                         14
         TIME1 FUNCTION   P1,D3       MEAN TIME FOR JOB "P1" AT ITS 1ST MACHINE 15
        1,600/2,400/3,550                                                   16
         TIME2 FUNCTION   P1,D3       MEAN TIME FOR JOB "P1" AT ITS 2ND MACHINE 17
        1,350/2,750/3,200                                                   18
         XPDIS FUNCTION   RN1,C24     EXPONENTIAL DISTRIBUTION FUNCTION     19
        0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38  20
        .8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2  21
        .97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8            22
        *                                                                   23
        *      DEFINE STORAGE CAPACITIES                                    24
        *                                                                   25
               STORAGE    S10,4/S11,6                                       26
        *                                                                   27
        *      MODEL SEGMENT 1                                              28
        *                                                                   29
1              GENERATE   110,FN$XPDIS,,,25  JOBS ENTER THE SHOP           30
2              ASSIGN     1,FN$JTYPE         SET P1 = CODE FOR JOB TYPE    31
3              QUEUE      P1                 JOIN SHOP-SPANNING QUEUE FOR  32
        *                                    THIS JOB TYPE                 33
4              PRIORITY   FN$TAG1,BUFFER     SET JOB'S PR FOR ITS 1ST MACHINE 34
5              ENTER      FN$FIRST           CAPTURE FIRST MACHINE         35
6              PRIORITY   FN$TAG2            SET JOB'S PR FOR ITS 2ND MACHINE 36
7              ADVANCE    FN$TIME1,FN$XPDIS  USE FIRST MACHINE            37
8              LEAVE      FN$FIRST           FREE FIRST MACHINE            38
9              ENTER      FN$SECON           CAPTURE SECOND MACHINE        39
10             ADVANCE    FN$TIME2,FN$XPDIS  USE SECOND MACHINE           40
11             LEAVE      FN$SECON           FREE SECOND MACHINE           41
12             DEPART     P1                 LEAVE SHOP-SPANNING QUEUE     42
        *                                    FOR THIS JOB TYPE             43
13             TERMINATE  1                  FINISHED JOB LEAVES THE SHOP  44
        *                                                                   45
        *      CONTROL CARDS                                                46
        *                                                                   47
               START      1000,,250          START THE RUN; GET SNAP PRINTOUTS 48
               END        RETURN CONTROL TO OPERATING SYSTEM                49
```

Figure 4D.2 Extended Program Listing for Case Study 4D

STORAGE	CAPACITY	AVERAGE CONTENTS	AVERAGE UTILIZATION	ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENTS	MAXIMUM CONTENTS
10	4	3.138	.784	266	367.582	3	4
11	6	5.175	.862	256	629.910	6	6

(a) 250 Jobs Completed

STORAGE	CAPACITY	AVERAGE CONTENTS	AVERAGE UTILIZATION	ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENTS	MAXIMUM CONTENTS
10	4	3.223	.805	515	367.962	3	4
11	6	5.508	.918	508	637.564	6	6

(b) 500 Jobs Completed

STORAGE	CAPACITY	AVERAGE CONTENTS	AVERAGE UTILIZATION	ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENTS	MAXIMUM CONTENTS
10	4	3.250	.812	775	364.135	1	4
11	6	5.479	.913	755	630.169	5	6

(c) 750 Jobs Completed

STORAGE	CAPACITY	AVERAGE CONTENTS	AVERAGE UTILIZATION	ENTRIES	AVERAGE TIME/TRAN	CURRENT CONTENTS	MAXIMUM CONTENTS
10	4	3.332	.833	1007	379.788	4	4
11	6	5.449	.908	1004	622.992	4	6

(d) 1000 Jobs Completed

Figure 4D.3 Storage Statistics for Case Study 4D

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
1	8	2.331	54		.0	1345.296	1345.296		2
2	19	7.487	126		.0	1851.626	1851.626		17
3	7	2.682	90		.0	928.655	928.655		1

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(a) 250 Jobs Completed

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
1	8	2.294	92		.0	1466.532	1466.532		2
2	19	8.859	248		.0	2100.286	2100.286		13
3	9	3.060	181		.0	994.010	994.010		6

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(b) 500 Jobs Completed

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
1	8	2.275	151		.0	1308.622	1308.622		1
2	25	9.620	365		.0	2288.440	2288.440		24
3	10	3.121	261		.0	1038.562	1038.562		2

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(c) 750 Jobs Completed

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
1	10	2.872	211		.0	1562.270	1562.270		7
2	27	9.558	477		.0	2300.037	2300.037		5
3	10	2.896	325		.0	1023.000	1023.000		1

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(d) 1000 Jobs Completed

Figure 4D.4 Queue Statistics for Case Study 4D

(7) Discussion

Model Implementation

Table 4D.2 had been used to determine what the relative job priorities should be at the two machine groups. The resulting priority assignments are shown in Table 4D.4.

		Machine Type	
		A	B
Job Category	1	5	7
	2	6	5
	3	7	6

Table 4D.4 Priority Levels Used in Case Study 4D

In column 1 of the table, Priority Levels at machine type A are 5, 6, and 7, corresponding to the mean processing times of 60, 40, and 20, respectively, shown in column 1 of Table 4D.2. Hence, the relative Priority Levels are inversely related to the mean processing times. The same remarks hold for columns 2 in Tables 4D.4 and 4D.2.

Choice of 5, 6, and 7 as Priority Levels is arbitrary. Any three different numeric values in the range from 0 to 127 could have been used. In this case, the values 1, 2, and 3 were avoided to eliminate confusion which might result because job categories have been coded as 1, 2, and 3.

Consistent with Table 4D.4, the Functions TAG1 and TAG2 (Cards 11 through 14 in Figure 4D.2) have been defined. Consider the Function TAG1. For job categories 1, 2, and 3, it returns values of 5, 6, and 6, respectively. A job in category 1 visits an A machine first; hence, from Table 4D.4, its first Priority Level is 5. Similarly, jobs in category 2 or 3 use A and B machines first, respectively; their first Priority Levels are consequently both 6, as indicated in Table 4D.4. By following similar reasoning, the Function TAG2 has been designed to return values of 7, 5, and 7 for jobs in categories 1, 2, and 3, respectively.

Storages 10 and 11 have been chosen to simulate the groups of type A and type B machines, respectively. In choosing these numbers, values of 1, 2, 3, 5, 6, and 7 were avoided to eliminate the confusion which could result if some numbers had more than one role in the model.

The logic used to define the Functions TAG1 and TAG2 is highly analogous to that used to define the Functions FIRST, SECON, TIME1, and TIME2 (see Figure 4D.2). Choose any one of these four Functions, then study its definition in Figure 4D.2 to satisfy yourself that the Function definitions are valid.

In line with the discussion in Section 4.17, note that just after entering the Storage at Block 5, a job's Priority Level is modified at Block 6 in anticipation of its visit to the next machine. Then use of the current machine is simulated at Block 7.

At Block 1, Transactions entering the model are given an initial Priority Level of 25. Then, still at the time of model entry, that Priority Level is changed to its "currently correct" value at Block 4. The reason for bringing Transactions into the model with a Priority Level higher than any of those which are "meaningful", per Table 4D.4, is subtle. It can probably best be explained through a simple example. Suppose that a category 2 job is finished with machine A just at the time a category 3 job enters the shop. Both jobs are headed for the group of B machines. Assume that exactly one of the B machines is available. The category 3 job should capture it, because its Priority

Level at that machine is 6, whereas that of the type 2 job is only 5 (see Table 4D.4). Now if the category 3 job were to enter the model with a default Priority Level of 0, it would be placed on the back of the Current Events Chain. By the time the Processor eventually got to it, then, the category 2 job would already have been permitted to capture the B machine. Hence, the problem logic would have been subverted. This invalid logic is sidestepped by giving incoming jobs a "highest" Priority Level (any value larger than 7 would be acceptable), so that they are put at the front of the Current Events Chain. The Processor therefore handles them first, moving them as far as Block 4, where their true Priority Level is assigned. The result is to re-position the job on the Current Events Chain, as the last member in its true Priority Class. Then, through use of the BUFFER option at Block 4, the scan is re-started. If this were not done, the newly-arrived job would immediately try to move into the ENTER Block at Location 6. But, if a machine in the corresponding group is available, and toward the front of the Current Events Chain a job with a higher Priority Level is poised to capture it, it is invalid to permit the new entrant to take the machine. Hence, if the BUFFER option were not used at Block 4, an attempt to make one aspect of the model be valid would result in another aspect becoming invalid.

Finally, note how the C Operand on the START Card (Card 48 in Figure 4D.2) has been used so that model output will be produced each time another 250 jobs have moved through the shop until, with the 1,000th job completion, the model shuts off.

Program Output[4h]

Figures 4D.3 and 4D.4 show Storage and Queue statistics, respectively, for the cases of 250, 500, 750, and 1,000 jobs having moved through the job shop. In Figure 4D.3, the Storage utilizations show little variation between parts (c) and (d). It is left as an exercise to compute the long-run utilizations which will be in effect for these Storages.

In Figure 4D.4, the extent to which job residence time in the shop has stabilized seems to depend on the sample size, as would be expected. Hence, for category 2 jobs, average residence time varies by less than 2% (from about 229 minutes to 230 minutes) between parts (c) and (d) in the figure. The corresponding sample sizes are 365 and 477 type 2 jobs, respectively. On the other hand, for category 1 jobs, average residence time varies by about 20% (from 131 minutes to 156 minutes) between parts (c) and (d) in the figure. The corresponding sample sizes for jobs in category 1 are 151 and 211, respectively.

The shop is apparently "stable". That is, the number of jobs in the shop does not grow without bound as the simulation proceeds. When the 250th, 500th, 750th, and 1,000th jobs were completed, there were 20, 21, 27, and 13 jobs, respectively, in the shop [sum of "CURRENT CONTENTS" in parts (a) through (d) of Figure 4D.4]. The large number of resident jobs at the time of the 750th completion has been substantially reduced by the time of the 1,000th completion.

[4h] Total CPU time for the simulation was 16.6 seconds.

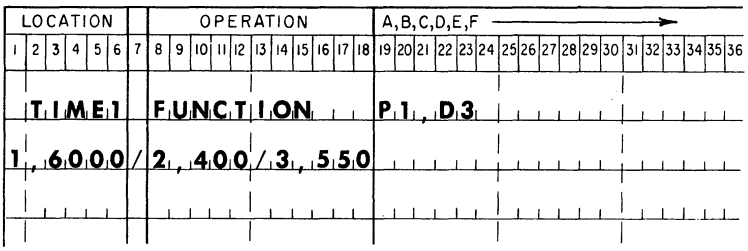
4.19. List Functions

Referring to the Function definitions in Case Study 4D (see the extended program listing in Figure 4D.2), note that all Functions except JTYPE have two properties.

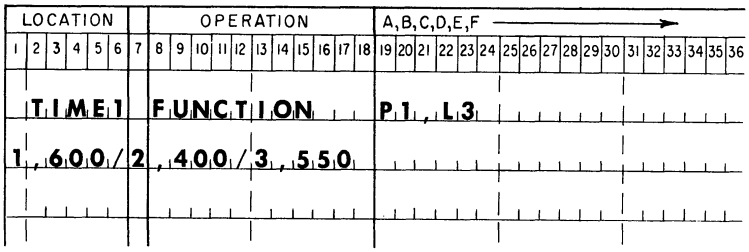
(1) They are defined as discrete.

(2) The first members of the ordered pairs used to define the Functions form an unbroken sequence of integers beginning with 1. (For the indicated Functions in Figure 4D.2, the unbroken sequence of integers in each case is simply 1, 2, 3.)

Whenever a Function has these two properties, it can be defined as a list Function. Definitionally, the B Operand of a list Function takes the form Lj, where j is the number of ordered pairs supplied to describe the Function. In every other way, list Functions are defined like discrete Functions. For example, Figure 4.26(a) shows the definition of the discrete Function TIME1 repeated from Figure 4D.2. Figure 4.26(b) shows TIME1 defined as a list Function. Except for the B Operand on the Function card, the definitions are identical.



(a) The Discrete Function



(b) The Equivalent List Function

Figure 4.26 Example of a Discrete Function, and Its Equivalent As a List Function

Computationally, less execution time is required to evaluate a list Function than to evaluate its discrete Function equivalent.(4i) It is therefore advantageous to use list Functions whenever possible.

If the value of a list Function's argument is found to be out of range, the GPSS Processor prints an error message to that effect and terminates the simulation. Compare this with what happens when a discrete Function's argument is out of range.

(4i) The reason for this is that the Processor uses the Function argument to index directly into the array of feasible Function values. Hence, no "table lookup" is required.

4.20 Exercises

4.20.1 (a) What happens when a Transaction moves into the Block "PRIORITY PR"?

(b) After a Transaction which entered a PRIORITY Block has come to rest, what does the GPSS Processor do next?

(c) Discuss the difference between the Block "PRIORITY P2" and the Block "PRIORITY P2,BUFFER".

(d) Critically discuss this statement: "When a Transaction with a Priority Level of 3 moves into the Block 'PRIORITY 3', the Block might just as well not have been there, for all practical purposes."

4.20.2 (a) Show how to change the discrete Function TIME2, defined in Figure 4D.2, to convert it to a list Function.

(b) When P4 has the values 3, 4, or 5, a Function is to have the values -2, -8, and 22, respectively. Show the definition of a discrete Function named MAP which describes this relationship.

(c) Can the discrete Function defined in (b) be converted to a list Function? Why or why not?

(d) What happens when the argument of a discrete Function is out of range?

(e) What happens when the argument of a list Function is out of range?

4.20.3 The questions in this exercise all refer to Case Study 4D.

(a) Compute the long-run utilizations for Storages 10 and 11. Compare your results with the corresponding statistics shown in Figure 4D.3.

(b) Show the Block Diagram for a model which uses the "three-segment approach" described in Section 4.16 for modeling the job shop problem. Is it necessary in this approach to give jobs an artificially high initial Priority Level when they enter the model? Thoroughly discuss the reasoning behing your answer.

(c) Suppose that the mean processing times in effect are those shown in Table T4.20.3, rather than those in Table 4D.2

		Machine Type	
		A	B
Job Category	1	20	75
	2	60	35
	3	40	35

Table T4.20.3

(i) Compute the number of machines of type A and B which must be in the shop for the utilization of the two machine groups to be as high as possible, but less than 100%.

(ii) Show how to change the definitions of the Functions TAG1, TAG2, TIME1, and TIME2 in the Figure 4D.2 model so that they will be valid under these new circumstances.

(d) Suppose that the machine-usage sequence in effect is the exact opposite of that shown in Table 4D.1. That is, suppose jobs in categories 1 and 2 visit a type B machine first, and then a type A machine; and that category 3 jobs use type A first, and then type B. Show how to change the definitions of the Functions FIRST and SECON in the Figure 4D.2 model so that they will be valid under these new circumstances.

4.20.4 In Case Study 4D, suppose it is proposed to change the queue discipline in the job shop from shortest imminent operation to first-come, first-served. Design a GPSS model, or models, with which to compare and contrast the average shop residence time of the three job types, depending on which queue discipline is in effect. Make the comparison on the basis of 1,000 jobs having moved completely through the shop. Control experimental conditions, so that any observed differences can be attributed to the two queue disciplines, given whatever specific inter-arrival and service time sequences are in effect for the experimentation.

4.20.5 Re-run the model in Figure 4D.2 two times, increasing the capacities of Storages 10 and 11 by 1 for the first run, and then by 2 for the second run. Compare and contrast the effect of the additional machines on average job residence time in the shop, and the number of jobs resident in the shop. Is the improvement as great in going from 1 to 2 extra machines of each type, as in adding an extra machine of each type in the first place?

4.20.6 In Case Study 4D, invalid modeling could result, in part, if the BUFFER B Operand were not used at Block 4 (Figure 4D.2), and/or if an E Operand of 25 were not supplied at the GENERATE Block (Block 1). How likely do you think it is that circumstances producing an invalid model would come about in a run in which 1,000 jobs move completely through the shop? Eliminate the GENERATE Block's E Operand, and the PRIORITY Block's B Operand, in the Figure 4D.2 model, and submit the resulting modified model for running. Does the output suggest that the modified model is partially invalid? Do you get the same output for the GPSS implementation at your computing center as that in Figure 4D.3 and 4D.4 when you run the Figure 4D.2 model without modification?

4.20.7 Compose a job shop problem in which the shop consists of three different types of machines, A, B, and C. Postulate that four different kinds of jobs, labeled as 1, 2, 3, and 4, move through the shop. Suggest a machine-usage sequence for these jobs (see Table 4D.1), and a set of mean machine-usage time (see Table 4D.2). Determine a job-type mix, i.e., what percentage of jobs moving through the shop fall into each of the four categories? Compute the minimum number of machines of types A, B, and C required in the shop if it is to be stable, i.e., if the number of jobs in the shop is not to grow without bound. Modify the model in Figure 4D.2 so that it corresponds to your job shop. Also consider the alternative possibility of using a "four-segment approach" to modeling the shop [see Problem 4.20.3(b)]. Which approach would you prefer to use in practice? About how much more COMMON is required in the "four-segment approach" than in the one-segment approach?

4.21 Residence Time and Transit Time as Transaction Properties

The length of time a Transaction has been in a model is its model residence time. The time that elapses while a Transaction moves between two arbitrarily-designated points within a model is its transit time between the two points. There are two Standard Numerical Attributes in GPSS whose values are a Transaction's model residence time, and transit time, respectively. They will be discussed in this section.

4.21.1 Model Residence Time

Each time a Transaction enters a model, the Processor makes a record of its time of entry. The time of entry is referred to as "Mark Time". Two points should be noted carefully here.

(1) A Transaction is understood to "enter" a model at the time it succeeds in moving out of its GENERATE Block. If a Transaction is delayed in a GENERATE Block, then, its time of model entry differs from its time of arrival at the GENERATE Block.

(2) The Absolute Clock is used in determining the time of entry, not the Relative Clock. (If RESET Cards are used in a model, the readings of the Absolute and the Relative Clocks may differ.)

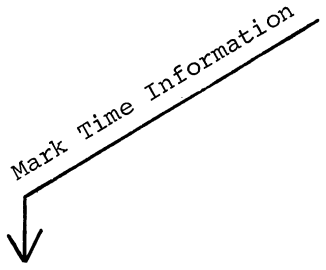
The record of a Transaction's time of model entry is not directly available during a simulation, and therefore cannot be directly used as part of the logic in a model. In other words, there is no Standard Numerical Attribute whose value is a Transaction's Mark Time.

There is, however, a Standard Numerical Attribute which is closely related to a Transaction's time of model entry. The name of the Standard Numerical Attribute is M1. Its value is the difference between the current reading of the Absolute Clock, and the Transaction's time of model entry. Hence, M1 measures the number of time units a Transaction has been in the model. To put it differently, M1 is a Transaction's residence time in a model.

Of course, for a given Transaction the value of M1 changes as a simulation proceeds. Just after it enters a model, a Transaction's M1 value is zero. Ten time units later, its M1 value is 10. After another ten time units have elapsed, its M1 value is 20, and so on.

For M1, the same question arises as that for Pj, and for PR. Namely, when the value of M1 is to be data, which Transaction is to be used in determining M1's value? The answer, of course, is that the M1 of the Transaction currently being processed is to be used.

Transaction Mark Time appears in the printout of the Current and Future Events Chain in a model. A portion of the Figure 4B.4 Future Events Chain printout is repeated in Figure 4.27. As pointed out in the figure, column 8, labeled "MARK-TIME", contains the Mark Time information. The Mark Time value is not made meaningful until a Transaction succeeds in leaving its GENERATE Block. If a Transaction is delayed in its GENERATE Block, its Mark Time in the Current Events Chain is actually a negative number. This is also true of Transactions on the Future Events Chain which are "on their way" to a GENERATE Block. Inspection of Figure 4.27 shows that Transactions 15 and 3 both have negative entries in their MARK-TIME column. Transaction 15 is a customer, on his way to the grocery store; Transaction 3 is the "timer", on his way to shut off the model. These values are negative for reasons which are internal to the GPSS Processor, and which will not be discussed here.



FUTURE EVENTS CHAIN

TRANS	BDT	BLOCK	PR	SF	NBA	SET	MARK-TIME	P1	P2
16	28806	4			5	16	28632	0	0
								0	0
								0	0
12	28830	4			5	12	28766	0	0
								0	0
								0	0
7	28836	10			11	7	28380	8	0
								0	0
								0	0
15	28840				1	15	-366	0	0
								0	0
								0	0
3	57600				20	3	-1	0	0
								0	0
								0	0

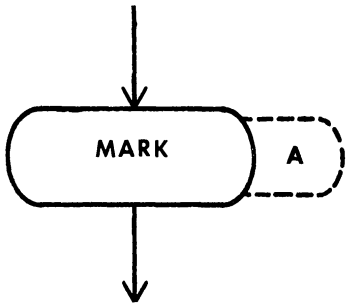
Figure 4.27 Repetition of a Portion of the Figure 4B.4 Future Events Chain

4.21.2 Transit Time

M1 measures the time that has elapsed since a Transaction first entered a model. It is frequently of interest to know how much time elapses while a Transaction is in transit between two points interior to a model. Suppose the time used by a Transaction in moving between interior Points A and B is to be determined. A two-step process is involved. When the Transaction is at Point A, a record must be made of the Absolute Clock. Then, when the Transaction arrives at Point B, the difference between the value recorded at Point A, and the current value of the Absolute Clock, must be computed. This difference is then the desired transit time. A method for accomplishing this two-step process in GPSS will now be described.

The first step involves using the GPSS MARK Block. When a Transaction enters the MARK Block, the Absolute Clock's value is copied into one of its Parameters. This is referred to as "marking the Transaction".

The MARK Block and its A Operand are shown in Figure 4.28. The A Operand specifies



Operand	Significance	Default Value or Result
A	Number of the Parameter into which the Absolute Clock's value is to be copied	See Footnote 4j

Figure 4.28 The MARK Block and Its A Operand

the number of the Parameter in which the Transaction is to be marked. The sole purpose of the MARK Block subroutine is to copy the Absolute Clock's value into the specified Parameter. By placing a MARK Block at "Point A" in the interior of a model, the first step in the two-step process consequently takes place.

When the Transaction later arrives at "Point B", the second step is to determine how much time has elapsed since the Transaction was marked. This elapsed time is the value of a Standard Numerical Attribute. The name of the Standard Numerical Attribute is MPj, where j is the number of the Parameter in which the Transaction was marked. To compute the value of MPj, the Processor subtracts the Transaction's j-th Parameter value from the Absolute Clock's current reading. When the Transaction reaches "Point B", then, MPj's value is the transit time between Points A and B.

As pointed out earlier, there is no Standard Numerical Attribute whose value is the Absolute Clock. The MARK Block can be used, however, to sidestep any difficulties that might arise as a consequence of this. For example, assume a model requires that the value of the Absolute Clock be known when a Transaction arrives at a certain point. If the Transaction moves into a MARK Block at that point, the value of the Parameter in which it is marked is the Absolute Clock's reading. If the Transaction is marked in Parameter 3, for instance, then the value of P3 is the Absolute Clock at the time of marking.

(4j) Defaulting is permitted. The result of defaulting is that the Absolute Clock's value replaces the Transaction's previously-recorded time of model entry. This means that no Parameter is involved. Furthermore, it means that M1 no longer measures the time a Transaction has been resident in a model. Instead, it measures elapsed time since the Transaction moved into the MARK Block used in "no-A-Operand" mode. Nothing is really gained by defaulting on the A Operand. On the contrary, the record of a Transaction's time of model entry is lost. The "no-A-Operand" mode will not be further discussed or used in this book.

4.22 Exercises

4.22.1 (a) What is meant by a Transaction's "time of entry" to a model?

(b) What is a Transaction's "Mark Time"?

(c) Why is it important to distinguish between Absolute and Relative Clock time when discussing the concept of Mark Time?

(d) When M1 is used to indirectly specify data, how does the GPSS Processor compute the value of the data?

4.22.2 (a) A Transaction enters a model at Absolute Clock time 21. At Absolute Clock time 52 it moves into the ASSIGN Block whose punchcard image is "ASSIGN 3,M1". What is the value of P3 after the ASSIGN Block subroutine has been executed?

(b) Later, at Absolute Clock time 72, the Transaction moves into the Block "ASSIGN 5,FN$DAVID". Figure P4.22.2 shows the definition of the Function DAVID.

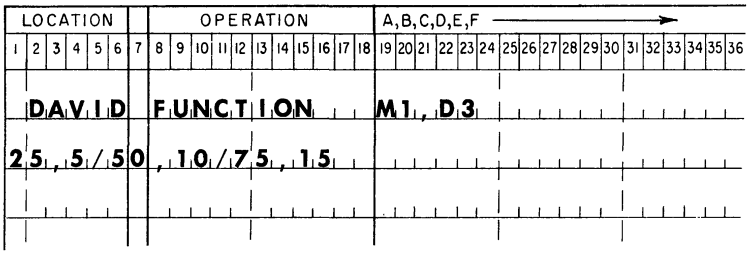


Figure P4.22.2

What is the value of P5 after the ASSIGN Block subroutine has been executed?

4.22.3 (a) What task does the MARK Block subroutine accomplish? (Assume that the Block's A Operand is used.)

(b) What does it mean to "mark a Transaction in Parameter 2"?

(c) How does the GPSS Processor compute the value of MP5?

(d) What is the difference between M1 and MP1?

4.22.4 At Absolute Clock time 5, a Transaction moves into the Block "MARK 6".

(a) After the MARK Block subroutine has been executed, what is the value of P6?

(b) Later, at Absolute Clock time 25, the Transaction moves into the Block "ASSIGN 4,MP6". What is the value of P4 after the ASSIGN Block subroutine has been executed?

(c) The Transaction then moves into the Block "ENTER JOE,P6". By what amount is R$JOE decreased as a result?

4.22.5 A Transaction is to capture the Facility whose number equals the current value of the Absolute Clock. Show a Block sequence with which the desired result can be accomplished. (Note: although this problem may not make sense logically, it can be "solved".)

4.22.6 At a certain point in a model, Transactions must move through Queue 7. To support the model logic, the time spent by each Transaction in the Queue must be made the value of that Transaction's Parameter 5. Show a Block sequence which accomplishes this.

4.23 Concept of the GPSS Table Entity

Suppose that a collection of numbers is produced by observing and recording the values taken on by some random variable. For example, each number in the collection might be the time required by a Transaction to move between Points A and B in a model. During a simulation, many Transactions move along the path from A to B. If the travel time of some or all of these Transactions is recorded, the result is a collection of values of the "travel time random variable". The collection is termed "a sample". The values in the sample are said to have been "drawn from a random variable's population".

It is of interest to analyze the numbers in such a sample to estimate the properties of the population from whence they came. Population properties of interest include the mean, the standard deviation, and the relative frequency with which population values fall within prescribed ranges. These prescribed ranges are sometimes called intervals, or frequency intervals, or frequency classes. Population properties are often estimated by computing the corresponding properties for the sample drawn from the population. Given a sample, this requires performing tasks such as these.

(1) Compute the sample mean.

(2) Compute the sample standard deviation.

(3) Count the number of sampled values which fall into each of a series of prescribed ranges.

(4) Compute the percentage of values in the sample which fall into the various ranges in the series.

Such computations, although straightforward, are tedious and time consuming if done by hand. As a result, the computations have been automated in GPSS. To have them performed automatically, use must be made of the Table entity. The steps involved are described in the next section.

4.24 Defining and Using Tables: The TABLE Card and the TABULATE Block

Use of the Table entity involves two steps. First, each of the one or more Tables to be used in a model must be defined. Second, arrangements must be made to have sampled values entered, one by one, in the various Tables of interest. The details involved are discussed below, and illustrated with an example.

4.24.1 The TABLE Card

There can be many different Tables in a model. Each Table must be defined before it can be used. A Table is defined by supplying these five pieces of information.

(1) The name (numeric or symbolic) of the Table.

(2) The name of the random variable whose values are to be tabulated.

(3) A number known as the "first boundary point". Sampled values less than or equal to this number fall into the left-most interval (frequency class) in the Table.

(4) The width common to all the Table intervals, excepting the left-most (lowest) and the right-most (highest).

(5) The total number of intervals in the Table, including the lowest and the highest.

Items (3) through (5) call for some clarification. The values being tabulated can, in general, range from "minus infinity" to "plus infinity".[4k] To use the Table entity, the span from minus infinity to plus infinity (i.e., the real number axis) must be

---

[4k] The maximum magnitude of integer values on the System/360 is 2,147,483,647. In practical terms, then, this is the value of "plus infinity". "Minus infinity" is -2,147,483,647.

divided into a series of consecutive intervals. The relative frequency with which tabulated values fall into each of these intervals can then be computed by the Processor. In defining a Table, items (3) through (5) indicate where the user wants the intervals to be located on the real number axis. Figure 4.29 provides a pictorial interpretation of this idea. Note that the left-most interval consists of the values from minus infinity

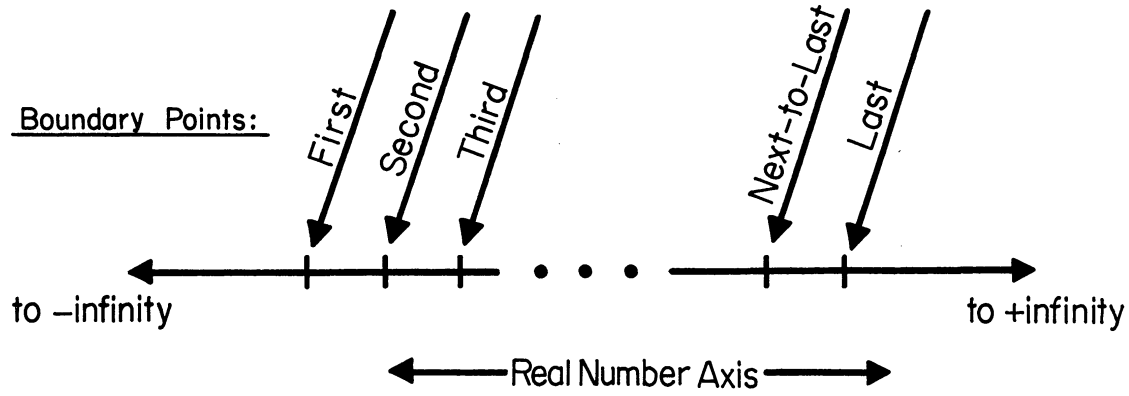


Figure 4.29 Pictorial Interpretation of the Real Number Axis and Its Division into a Series of Consecutive Intervals

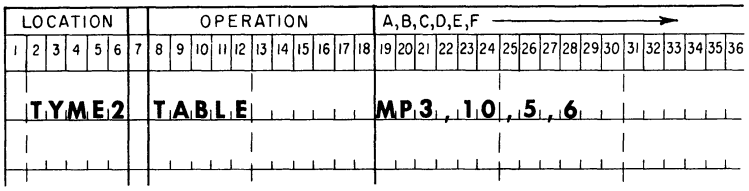
up to and including the "first boundary point". The second interval consists of values greater than the first boundary point, but less than or equal to the second boundary point. All the intermediate intervals are of uniform width. Finally, the right-most interval covers all values greater than the "last boundary point".

The five items of information are provided on a single punchcard, the Table Card. Like most other cards, the Table Card is divided into the Location, Operation, and Operands fields. The information entered in these fields is shown in Table 4.10.

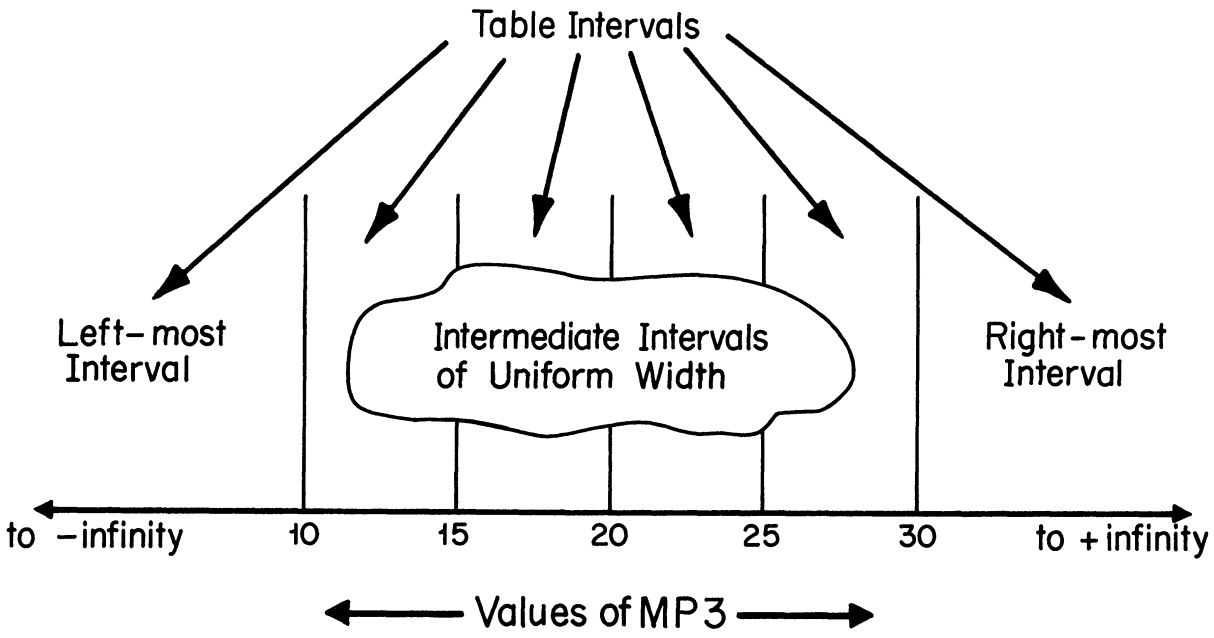
Punchcard Field	Information Supplied in Field
Location	The name (numeric or symbolic) of the Table
Operation	Literally, the word TABLE
Operands	
A	The name of the random variable whose values are to be entered in the Table. In particular, the A Operand will be the name of some Standard Numerical Attribute
B	The first boundary point
C	The width of each intermediate Table interval
D	The total number of intervals in the Table, including the lowest and the highest

Table 4.10 Table Card Specifications

An example of Table definition appears in Figure 4.30. In part (a) of the figure, the Table Card is shown. In part (b), a pictorial interpretation of the intervals into which the real number axis is divided is given. The Table is named TYME2. The Standard Numerical Attribute being tabulated is MP3 (A Operand). The first boundary point is 10 (B Operand). The width of intermediate intervals in the Table is 5 (C Operand). This means that the second boundary point is 15 (that is, 10 + 5), the third boundary point is 20 (that is, 15 + 5), ... , and the last boundary point is 30. The real number axis is divided into a total of 6 consecutive intervals (D Operand). This means, in addition to the leftmost and rightmost intervals, there are 4 intermediate intervals.



(a) Definition of the Table



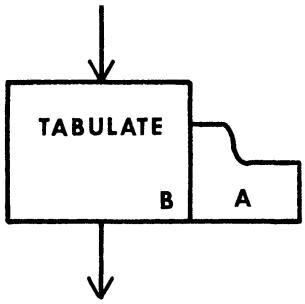
(b) Pictorial Interpretation of the Table Intervals

Figure 4.30 Definition and Interpretation of a Particular Table

The values of the B, C, and D Operands on the Table Card must be specified as constants. Neither the boundary points nor the number of intervals can be changed as a simulation proceeds. As usual, the constants must be integers. The effect of this is that all the boundary points are integers. For that matter, all sampled values entered into a Table are also integers.

4.24.2 The TABULATE Block

Sampled values are entered into a Table one by one, as a simulation proceeds. A value is entered into a Table each time a Transaction moves into a TABULATE Block. This Block, and its A and B Operands, are shown in Figure 4.31. The A Operand names the Table into which a value is to be placed. A given Table can be referenced from many different TABULATE Blocks in a model, when it is logical to do so. Note that the Standard Numerical Attribute being tabulated is not specified at the TABULATE Block. It is provided only on the Table Card.



Operand	Significance	Default Value or Result
A	Name (numeric or symbolic) of the Table into which a value is to be entered	Error
B	The number of times the value is to be entered in the Table each time a Transaction moves into the TABULATE Block	1

Figure 4.31 The TABULATE Block, and Its A and B Operands

The TABULATE Block's B Operand indicates the number of times the SNA's value is to be entered in the Table each time the TABULATE Block subroutine is executed. Normally, the analyst defaults on the B Operand. As a result, when a Transaction moves into the TABULATE Block, the Standard Numerical Attribute's current value is entered in the Table one time. If the B Operand is 2 or greater, the Table is said to be weighted. When a Table is weighted, the letter W must follow the comma separating the C and D Operands on the Table Card. The potential utility of weighting a Table will be described later.

Now consider how the distribution of Transaction transit time between Points A and B in a model can be estimated. At Point A, the Transactions move through a MARK Block, where Parameter 3 (for example) is marked with a copy of the Absolute Clock's value. At Point B, the Transactions move through a TABULATE Block, where the Table TYME2 (defined in Figure 4.30) is referenced via the A Operand. By definition, the values entered in the Table TYME2 are MP3. But MP3 is the transit time of Transit time of Transactions between the MARK Block, and the TABULATE Block. Each time a Transaction enteres the TABULATE Block, then, another sampled value of the transit time is added to the collection of such values which are being gathered in the Table.

4.24.3 An Example of Table Use

Relative to Case Study 4B (A Grocery Store Model), suppose it is of interest to tabulate the distribution of shopper residence time in the store. Only a modest extension of the Figure 4B.2 model is required to do this. Figure 4.32 shows the extended program listing for the modified model. Card 22 defines a Table, named RTIME, and having M1 as its A Operand. Card 44 is the punchcard image for a TABULATE Block which has been placed between the Blocks "LEAVE CARTS" and "TERMINATE". Just before each shopper-Transaction leaves the store, then, it passes through the TABULATE Block, and its model residence time is entered in the Table RTIME.

Figure 4.33 shows the output produced at the end of the simulation run with the Figure 4.32 model. In total, 363 shoppers had completed their shopping at the end of the simulation ("ENTRIES IN TABLE"). The average shopper residence time was 318 seconds ("MEAN ARGUMENT" was 317.8). The standard deviation in this sample of size 363 was 121

```
BLOCK                                                                    CARD
NUMBER  *LOC   OPERATION  A,B,C,D,E,F,G          COMMENTS                NUMBER
               SIMULATE                                                     1
        *                                                                   2
        *      DEFINE FUNCTIONS                                             3
        *                                                                   4
         AYL1  FUNCTION   RN1,C2       ITEMS-SELECTED DISTRIBUTION, AISLE 1  5
        0,2/1,5                                                             6
         AYL2  FUNCTION   RN1,C2       ITEMS-SELECTED DISTRIBUTION, AISLE 2  7
        0,3/1,6                                                             8
         AYL3  FUNCTION   RN1,C2       ITEMS-SELECTED DISTRIBUTION, AISLE 3  9
        0,4/1,7                                                            10
         COTYM FUNCTION   P1,C2        CHECKOUT-TIME DISTRIBUTION           11
        1,3/18,54                                                          12
         IMPUL FUNCTION   RN1,C2       DISTRIBUTION OF ITEMS TAKEN ON IMPULSE 13
        0,1/1,4                                                            14
         XPDIS FUNCTION   RN1,C24      EXPONENTIAL DISTRIBUTION FUNCTION    15
        0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38 16
        .8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2 17
        .97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8           18
        *                                                                  19
        *      DEFINE TABLES                                               20
        *                                                                  21
         RTIME TABLE      M1,100,100,9 TABLE FOR DIST'N OF TIME SPENT IN STORE 22
        *                                                                  23
        *      MODEL SEGMENT 1                                             24
        *                                                                  25
1               GENERATE   75,FN$XPDIS     SHOPPERS ARRIVE               26
2               ENTER      CARTS           TAKE A CART                   27
3               TRANSFER   .25,,TRY2       WANT TO SHOP AISLE 1?         28
4               ADVANCE    120,60          TIME IN AISLE 1               29
5               ASSIGN     1,FN$AYL1       SET P1 = ITEMS SELECTED IN AISLE 1 30
6        TRY2   TRANSFER   .45,,TRY3       WANT TO SHOP AISLE 2?         31
7               ADVANCE    150,30          TIME IN AISLE 2               32
8               ASSIGN     1+,FN$AYL2      SET P1 = TOTAL ITEMS SELECTED SO FAR 33
9        TRY3   TRANSFER   .18,,OUT        WANT TO SHOP AISLE 3?         34
10              ADVANCE    120,45          TIME IN AISLE 3               35
11              ASSIGN     1+,FN$AYL3      SET P1 = TOTAL ITEMS SELECTED SO FAR 36
12       OUT    QUEUE      GIRL            QUEUE FOR CHECKOUT COUNTER    37
13              ASSIGN     1+,FN$IMPUL     ADD TO P1 ITEMS TAKEN ON IMPULSE 38
14              SEIZE      GIRL            CAPTURE THE CHECKER           39
15              DEPART     GIRL            LEAVE THE CHECKOUT QUEUE      40
16              ADVANCE    FN$COTYM        CHECK-OUT TIME                41
17              RELEASE    GIRL            FREE THE CHECKER              42
18              LEAVE      CARTS           RETURN THE CART               43
19              TABULATE   RTIME           PLACE RESIDENCE TIME SAMPLE IN TABLE 44
20              TERMINATE                   LEAVE THE STORE              45
        *                                                                  46
        *      MODEL SEGMENT 2                                             47
        *                                                                  48
21              GENERATE   28800           TIMER ARRIVES AT END OF 8-HOUR DAY 49
22              TERMINATE  1               SHUT OFF THE RUN              50
        *                                                                  51
        *      CONTROL CARDS                                               52
        *                                                                  53
               START      1               START THE RUN                 54
               END        RETURN CONTROL TO OPERATING SYSTEM               55
```

Figure 4.32 Figure 4B.2 Extended to Include Tabulation of Shopper Residence Time

seconds ("STANDARD DEVIATION" is 120.562).

TABLE RTIME
ENTRIES IN TABLE 363 | MEAN ARGUMENT 317.831 | STANDARD DEVIATION 120.562 | SUM OF ARGUMENTS 115373.000

UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIAT FROM M
100	9	2.47	2.4	97.5	.314	-1.
200	59	16.25	18.7	81.2	.629	-.
300	93	25.61	44.3	55.6	.943	-.
400	93	25.61	69.9	30.0	1.258	.
500	89	24.51	94.4	5.5	1.573	1.
600	20	5.50	100.0	.0	1.887	2.

REMAINING FREQUENCIES ARE ALL ZERO

Figure 4.33 Table Output Produced When the Figure 4.32 Model Was Run

Nine customers spent 100 seconds or less in the store ("OBSERVED FREQUENCY" is 9 for the Table interval with an UPPER LIMIT of 100). These 9 customers constituted 2.47% of the points in the sample ("PERCENT OF TOTAL"). The CUMULATIVE PERCENTAGE of data points falling in the interval between "minus infinity" and an UPPER LIMIT of 100 is 2.4. The CUMULATIVE REMAINDER of points yet to be accounted for (because they fall into intervals further to the right in the Table) is 97.5.

The entries in the right-most two columns in Figure 4.33 provide information about the UPPER LIMIT of the corresponding frequency class, not about the entries in the frequency class. For example, the UPPER LIMIT of the left-most frequency class is 100; this value equals 0.314 times the average Table entry ("MULTIPLE OF MEAN" is 0.314). The difference between this UPPER LIMIT of 100, and the average Table entry, is -1.806 Table standard deviations ("DEVIATION FROM MEAN" is -1.806). The same remarks hold for the other intervals.

The Table Card in the Figure 4.32 model (Card 22) specifies that there are 9 Table intervals. In Figure 4.33, however, information appears for only 6 intervals. The last of these 6 intervals, which covers a range of values from 500+ to 600, contains 20 entries (OBSERVED FREQUENCY). Because none of the sampled values were larger than 600, the additional 3 Table intervals covering a range from 600+ to "infinity" were all empty. The GPSS Processor consequently suppressed printing out information for these intervals, terminating the Table with the message REMAINING FREQUENCIES ARE ALL ZERO.

When one or more sampled values do fall into the right-most Table interval, the Processor prints the word OVERFLOW in the UPPER LIMIT column, then fills out the rest of that row with the usual information about entries in that interval. As an example of this, Figure 4.34 shows the output produced when the Figure 4.32 model was run with the D Operand on the Table Card changed from 9 to 6. Although the OVERFLOW line indicates how many values fell into the right-most interval, the analyst has no idea of how large these values might have been, because the interval covers the range up to "infinity". As a result, the AVERAGE VALUE OF OVERFLOW is printed out by the Processor at the bottom of the Table. In Figure 4.34, for example, the average of the 20 values falling into the right-most interval is 525.84.

TABLE RTIME

ENTRIES IN TABLE	MEAN ARGUMENT	STANDARD DEVIATION	SUM OF ARGUMENTS
363	317.831	120.562	115373.000

UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIATION FROM MEAN
100	9	2.47	2.4	97.5	.314	-1.806
200	59	16.25	18.7	81.2	.629	-.977
300	93	25.61	44.3	55.6	.943	-.147
400	93	25.61	69.9	30.0	1.258	.681
500	89	24.51	94.4	5.5	1.573	1.510
OVERFLOW	20	5.50	100.0	.0		
AVERAGE VALUE OF OVERFLOW		525.84				

Figure 4.34 Example of a Table in which Overflow Occurs

As the estimate of a probability distribution, the information in a Table is sometimes better revealed graphically, in the form of a histogram, than it is in tabular form. Figure 4.35 shows a hand produced histogram corresponding to the Figure 4.33 Table. The

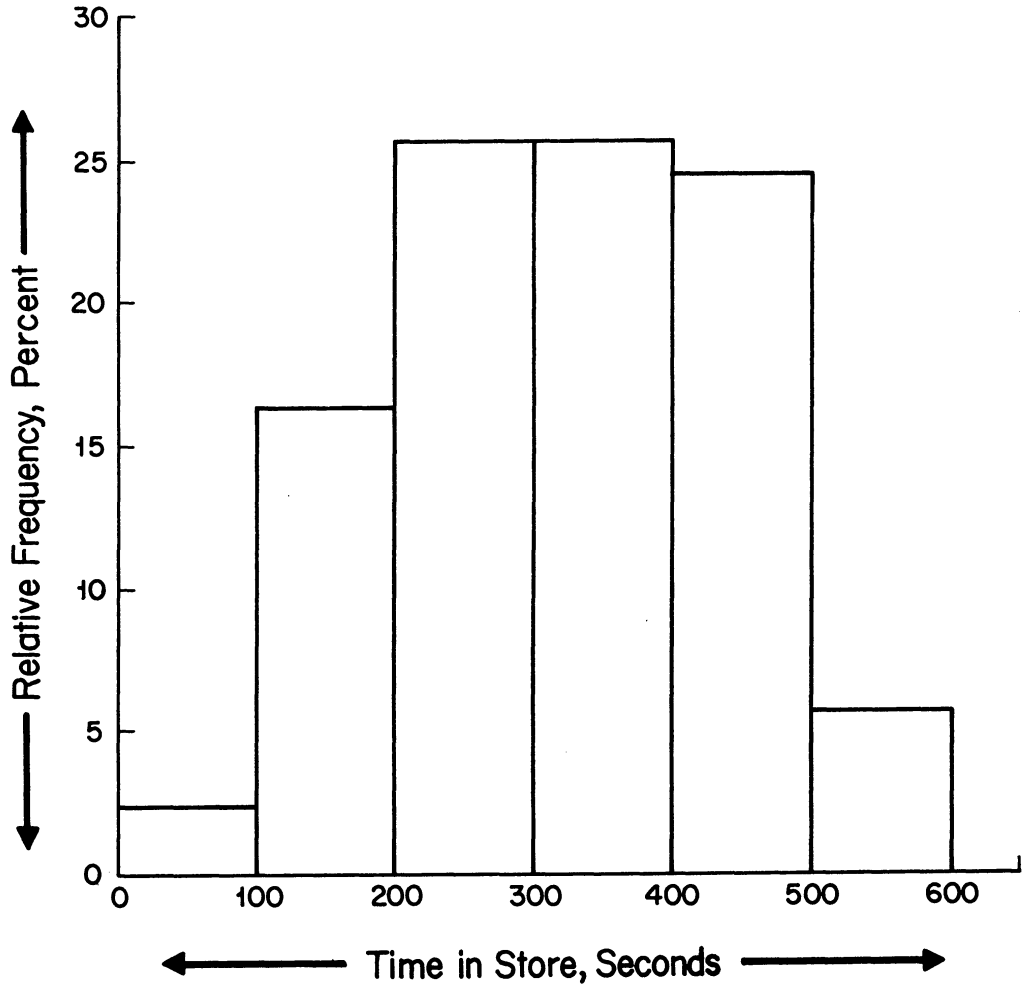


Figure 4.35 A Hand-Produced Histogram for the Figure 4.33 Table

horizontal line forming the "top" of each rectangular section in the histogram spans the same range of values as does the corresponding Table interval. The "height" of each rectangle is proportional to the percentage of sampled values which fall into the interval in question. For example, the left-most rectangle in Figure 4.35 is 2.47 units high (corresponding to a relative frequency of 2.47%), and spans the horizontal range from 0 to 100. [The base starts at 0 in this case, not "minus infinity", because residence time cannot take on negative values.]

It is possible to have the GPSS Processor produce histograms for Tables in a model. To do so requires use of the GPSS Output Editor. Figure 4.36 shows the histogr

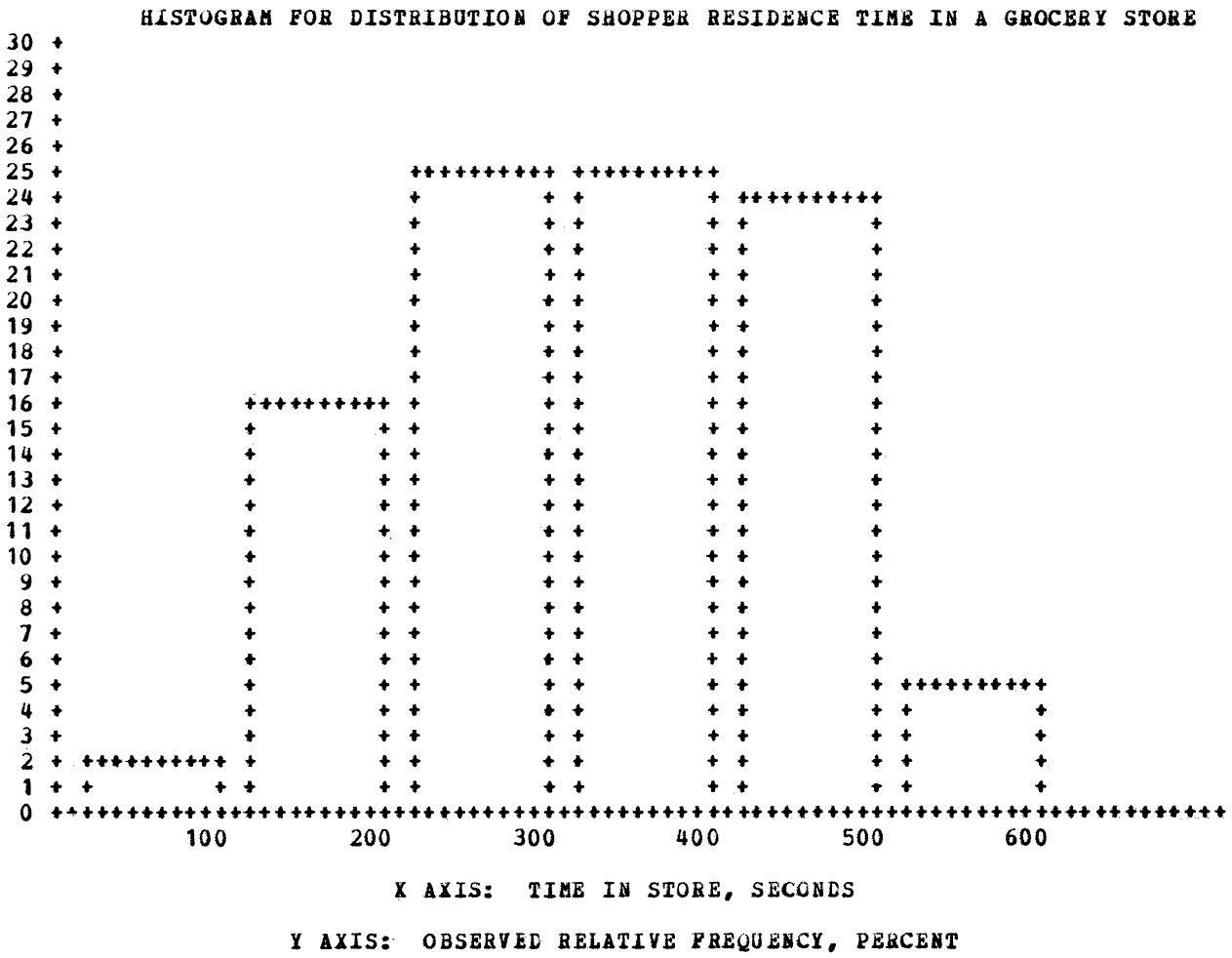


Figure 4.36 Histogram Produced for the GPSS Table in Figure 4.33 with Use of the GPSS Output Editor

produced by the GPSS Processor for the Figure 4.33 Table when the Output Editor is used with the Figure 4.32 model, and appropriate specifications are provided. Actually, the Output Editor can be used to perform a variety of services, including (a) providing analyst-supplied explanatory text and labels as part of the Processor's output, (b) sel ting and printing only that statistical information which is of interest, and (c) outputting information about user-selected Standard Numerical Attributes in a graphical format, including a series of possibilities for the Table entity. Although it can be useful on occasion, the Output Editor involves a series of mechanical details and does not, in itself, contribute to or enhance the conceptual aspects of simulation modeling. For this reason, specifics of its use will not be taken up in this book. The intereste

reader is referred to the Users Manual.

4.25 Table Standard Numerical Attributes

Several properties of Tables are available as Standard Numerical Attributes. This means that selected Table information can be used as part of the logic of a model. The Standard Numerical Attributes, and the significance of their values, are shown in Table 4.11.

Name[4e1]	Value
TBj, or TB$sn	The average value of the Table entries
TCj, or TC$sn	The number of sampled values entered in the Table
TDj, or TD$sn	The standard deviation of the Table entries

Table 4.11 Table Standard Numerical Attributes

The values of the Table Standard Numerical Attributes are time-dependent. TCj, of course, is a direct count of the number of values which have been placed in the Table. As TCj increases, the values of TBj and TDj generally tend to vary less than they did initially. In fact, it could be of interest to use the value of TBj, say, to control the time span covered by a simulation. A "stopping condition" such as this might be imposed: "When TCj exceeds 1,000, and a 10% increase in TCj produces less than a 5% change in TBj, turn off the simulation".[4m] To implement such logic in a model, it would be necessary to test the value of TBj dynamically, as the simulation proceeds. The ability to construct and use such tests in GPSS models is discussed in Chapter 5.

In Section 4.2, it was emphasized that the numeric form of certain entity properties depends on whether the values appear in model output, or are used as data within the model. This remark holds for the Table Standard Numerical Attributes as well. In the output for the Table RTIME in Figure 4.33, for example, Table mean and standard deviation appear as 317.831 and 120.562, respectively. The values of TB$RTIME and TD$RTIME, however, are 317 and 120. That is, when these two Table SNA's are used internally to indirectly specify data, only the integer portion of the true values is in effect. No such distinctions in numeric form need be made with respect to TCj (or TC$sn), because this attribute is inherently integer-valued.

(4e1) As already indicated in footnote 4c, "j" is the number of the particular Table in question; or, if it has been named symbolically, "sn" is its symbolic name.

(4m) There would be obvious dangers in this approach. The analyst would probably also design the model to turn off if the simulated clock reached some specified large value before this other condition was yet satisfied.

4.26 Exercises

4.26.1 (a) How is the number of intermediate intervals in a Table related to the D Operand on the Table Card?

(b) Show a pictorial interpretation of the Table intervals when the Table Card is "JOE TABLE P3,-50,25,8".

(c) What range of values is spanned by the left-most interval in the Table JOE, as defined in (b)? What range of values is spanned by the third interval?

(d) Into which interval of the Table JOE, as defined in (b), does the value 25 fall?

(e) What is the smallest value that could ever be entered in the Table SOFAR, for which the Table Card is "SOFAR TABLE M1,-100,50,15"?

4.26.2 In a certain model, two different Tables are defined with Q$LONG used as the A Operand in the Table Cards. The Table Cards are shown in Figure P4.26.2.

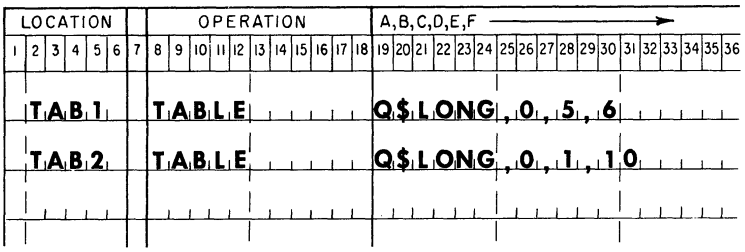


Figure P4.26.2

One TABULATE Block is used to reference TAB1; another is used to reference TAB2. The two TABULATE Blocks are placed one after the other. Give a possible reason why the model might have been built this way. (Hint: the reason does not involve the choice of Q$LONG as the Table Card A Operand.)

4.26.3 Explain the difference between the two Tables whose Table Cards are shown in Figure P4.23.3.

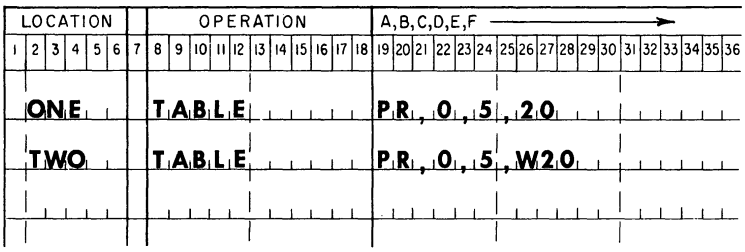


Figure P4.26.3

4.26.4 Three Transactions, with P2 values of 2, 3, and 4, respectively, move through the Block "TABULATE TABL4". In the same model, two other Transactions, each with a P2 value of 5, move through the Block "TABULATE TABL4,2". Assuming no other references have been made to the Table, what are the values of TB$TABL4 and TC$TABL4 after this has happened? What must be true of the D Operand on TABL4's Table Card?

4.26.5 In Case Study 4D, the GPSS Queue entity was used to estimate the average residence time of jobs in a job shop. Use of the Queue entity for this application was somewhat artificial. In the real system, there is no actual Queue "surrounding" the shop; furthermore, the Queue entity provided only an estimate of the average shop residence time for jobs in each category, and gave no additional information about the distribution of the "residence

time" random variable. Eliminate the Queue entity from the Case Study 4D model. Show how to use the Table entity in the model to estimate the distribution of shop residence time for the three job categories.

4.26.6 Customers may have to wait for a "constrained resource" at Point A in a model. Later, they may have to wait for another "constrained resource" at Point B. A random variable of interest in the model is "total time customers spend waiting for constrained resources". For example, if a particular customer waits 5 minutes at Point A, and another 3 minutes at Point B, then his total time spent waiting is 8 minutes. Describe how the distribution of the "total waiting time" random variable can be estimated in the model.

4.26.7 Modify the widget-manufacturing model in Case Study 2D so that it will provide estimates of the distributions of these three random variables.

(1) Total working time per widget assembly (consists of two components: assembly time; and firing time).

(2) Total clock time elapsed per widget assembly (consists of three components: assembly time; time spent waiting for the oven; and firing time).

(3) Assembly time.

Run the model for the case of 4, 5, and 6 assemblers, simulating for 5 working days for each configuration. Then do the following.

(a) The average value of the random variable in (1), and its smallest and largest values, can easily be determined from data used in the model. Compare these known population values with the distribution estimate produced by the model.

(b) Using the mean value estimated by the model for the random variable in (2), compute the number of widgets that should have been completed during each simulation. Are your computed values in agreement with the number of completed widgets as determined through Block Counts?

(c) How closely does the estimate of the assembly time distribution in (3) compare with its known distribution from configuration to configuration (uniform over the integers from 25 through 35, inclusive)?

4.26.8 Service time at a point in a system is exponentially distributed with a mean value of 100 time units. Devise a GPSS model to draw 1,000 samples from the service time distribution and enter the resulting values in a Table. Print the Table after the number of entries in it is 100, 200, 300, ..., 1,000. Compare the Table mean and standard deviation, printout-by-printout, with the population mean and standard deviation, each of which is known to be 100. Which of these two statistics appears to approach its expected value the most rapidly as sample size increases? Design the Table intervals so that the relative frequency with which values fall into the range between 0+ and 20, 80+ and 100, and 140+ and 160, will be tabulated. Compare the observed relative frequencies, printout-by-printout, with the relative frequencies expected in the long run. [The long-run relative frequencies can be determined by inspecting Figure 3.22.] For which of these intervals does the estimate of relative frequency seem to converge to the known theoretical value the most rapidly? Can you explain why?

4.26.9 In a certain application, an analyst must estimate the relative frequency with which a random variable takes on values in the range between 0+ and 20, and 20+ and 40. At the same time, he must estimate the relative frequency with which the same random variable takes on values in the range between 19+ and 21, 21+ and 23, and 23+ and 25. Assuming that randomly-sampled values of the random variable are contained in Parameter 2 of Transactions moving past Point A in a model, suggest a plan whereby he can obtain the necessary estimates.

4.26.10 The following problem is taken, with permission, from Chapter 3 in Operations Research, Methods and Problems, by Maurice Sasieni, Arthur Yaspan, and Lawrence Friedman [Copyright (c) 1959 by John Wiley & Sons, Inc; by permission of John Wiley & Sons, Inc.]

An airline has 15 flights leaving a given base per day, each with one stewardess. The airline has a policy of keeping three reserve stewardesses on call to replace stewardesses scheduled for flights who become sick. The probability distribution for the daily number of sick stewardesses follows.

Number Sick	Probability
0	0.20
1	0.25
2	0.20
3	0.15
4	0.10
5	0.10

Build a GPSS model to simulate this situation for the purpose of estimating the utilization of reserve stewardesses, and the distribution of the random variable "number of flights canceled each day because no stewardess is available". Simulate for 1,000 days of operation, printing out the estimates every 100 days. By hand, compute the theoretical values of the statistics under investigation. Compare the behavior of the estimates, as a function of sample size, with their known long-run values.

4.26.11 The inter-arrival time of ships at a harbor follows the distribution shown in Table T4.26.11. Thirty percent of the ships are of a newer type which

Inter-arrival Time, Minutes	Cumulative Frequency
Less than 125	0.0
175	0.10
225	0.22
275	0.40
325	0.55
375	0.78
425	0.90
475	1.0

Table T4.26.11

uses high speed unloading equipment. These ships move to Unloading Complex 1 (see Figure P4.26.11), which consists of two berths, each able to unload a ship in 1±0.1 days. The older type ships move to Unloading Complex 2, which consists of 6 berths, each able to unload a ship in 1.5±0.2 days. Before a ship can dock, it requires the services of a tug. Docking time requires 30±10 minutes. Similarly, before a ship can clear a berth after

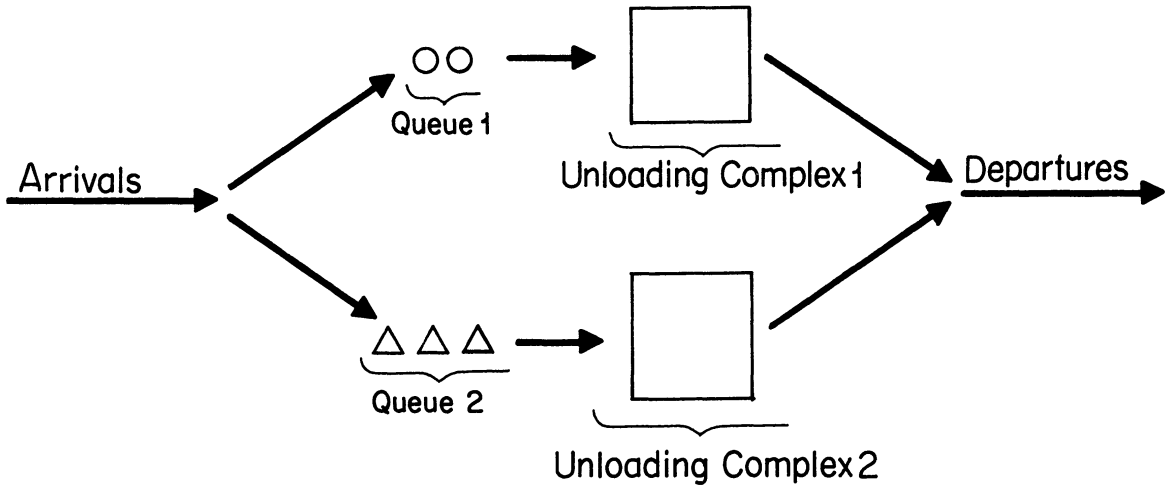


Figure P4.26.11

it has been unloaded, the services of a tug are required. A berth can be cleared in 15±5 minutes.

Show how to build a GPSS model of the system which uses only a single sequence of Blocks. Design the model to estimate both the distribution of ship residence time at the harbor, and the distribution of the random variable "total time each ship spends waiting at the harbor." Simulate until 100 ships have moved through the harbor. Then answer these questions, re-configuring the model and simulating again only if necessary.

(a) If the cost of a waiting ship is $500 per hour, at what estimated berth cost per day could addition of another berth in Unloading Complex 2 be justified?

(b) Repeat (a) with respect to addition of another berth in Unloading Complex 1. Are the answers to (a) and (b) independent of each other?

(c) If the cost of providing a tug is $250 per day, would it be better to have 2 tugs at the harbor, instead of just 1?

4.26.12 Problem 3.17.4 requires building a model or models to compare the average waiting time (including service) for each of three types of jobs at a work center when the queue discipline is (a) first-come, first-served, and (b) first-come, first-served within priority class. Build and run another model or models for the work center to estimate the distribution of waiting time for each job type, depending on the queue discipline used. What influence does queue discipline have on such properties of the waiting time random variable as standard deviation, and the relative frequency with which it takes on values falling within prescribed intervals? For jobs of "type 1", prepare a histogram for the waiting time distribution when queue discipline is first-come, first-served. Then, using a different pen color, superimpose on this plot the histogram for the "job type 1" waiting time distribution when priority distinctions are made. Discuss as many aspects of the differences in the two distributions as you can.

4.27 Additional Table Modes

There are three special modes in which Tables can be used. These modes are known as IAT-mode, RT-mode, and QTABLE-mode, respectively. The modes will be discussed in this section.

4.27.1 IAT-Mode Tables

Whenever a GENERATE Block is used, the distribution of inter-arrival times at that point in the system is required as data. Such a distribution describes one of the ways the system being modeled interfaces with its external environment. Information about these inter-arrival times must either be postulated, or gathered via direct measurement.

There are other inter-arrival time distributions which may also be important to the analyst. These distributions apply to Transactions as they arrive at points interior to a system. The GPSS Table entity can be conveniently used to provide estimates of such distributions as part of a simulation.

The logic of gathering data on Transaction inter-arrival times at a point interior to a model is straightforward. A record must be made of the Absolute Clock's value when the first Transaction reaches the point. Later, when the second Transaction arrives, the time recorded earlier must be subtracted from the current reading of the Absolute Clock. This difference is an interarrival time. It can be entered in a GPSS Table. Then, when the next arrival occurs at the point, its interarrival time relative to its predecessor can be computed and tabled. In this fashion, the distribution of interarrival times at the point can be estimated.

Interarrival times are automatically computed and entered in GPSS Tables when they are used in a special mode, known as IAT (for interarrival time) mode. If interarrival times at a point are to be the values entered in a Table, two steps must be taken.

(1) The A Operand on the Table Definition Card must be IAT. (The B, C, and D Operands have their usual significance.)

(2) A TABULATE Block referencing the Table (i.e., with the Table's name as its A Operand) must be placed at the point of interest in a model.

Nothing further is required to use a Table in IAT-mode.

4.27.2 RT-Mode Tables

Closely related to interarrival time is arrival rate. The analyst may want to estimate the distribution of the rate at which Transactions arrive at a point. There is a special Table mode with which this can be done, the mode is RT (for rate) mode. If arrival rates are to be tabulated, three steps must be taken.

(1) RT must be entered as the A Operand on the Table Definition Card. (The B, C, and D Operands carry the usual information.)

(2) An E Operand must be provided on the Table Definition Card. The E Operand is the time span relative to which the rate will be measured.

(3) A TABULATE Block referencing the Table must be placed at the point of interest in the model.

Note that a rate is specified relative to some time interval. Such phrases as "cars per hour", "people every 15 minutes", and "orders every 5 days" are used to describe rates. It is necessary that the GPSS Processor know how large this time interval is to be when rate information is gathered. The user supplies this information with the E Operand on the Table Definition Card. The implicit time unit used in the model is understood to apply to the E Operand's value as well. If the implicit time unit is, say, 1 second, and the Table BRIAN is defined with the card "BRIAN TABLE RT,0,5,10,30",

then the rate information is expressed "per 30 seconds".

The logic of gathering rate data is worth some mention. Each Table used in RT mode has a special counter associated with it. At the start of a simulation, this counter is given a value of zero. Whenever the Table is referenced at a TABULATE Block, the counter is incremented by 1 (actually, by an amount equal to the TABULATE Block's B Operand value). When a time interval equal to the E-Operand's value has elapsed for the first time, the value of the counter is entered in the Table, and the counter is set back to zero. The process of incrementing the counter then continues while the specified time interval is again being spanned. When the interval has been spanned the next time, the counter's value is again placed in the Table, the counter is restored to zero, and so on.

The method used by the Processor to "know" when it is time to enter the counter's value into an RT-mode Table should be pointed out. When the Processor encounters an RT-mode Table Card during the model input phase, it fetches a Transaction from the latent pool and hooks it onto the Future Events Chain, with its Move Time equal to the E Operand value on the Table Card. When that time is reached, the Transaction is then transferred to the Current Events Chain, as would be expected. Note that this Transaction is a "dummy"; it is not scheduled to enter the model through a GENERATE Block. When the CEC scan encounters the Transaction, the corresponding counter value is placed in the Table, and the counter is restored to zero. The Table Card's E Operand value is then added to the clock, and the "dummy" Transaction is put back on the Future Events Chain, scheduled to return to the CEC at that next time. And so on.

The consequence of this use of a "dummy" Transaction should be kept in mind. For each RT-mode Table in a model, such a Transaction will appear in any listing of the Future or Current Events Chain. The analyst should not think "something is wrong" when he finds such Transactions on chain listings.

4.27.3 QTABLE-Mode Tables

In Queue statistics printed out at the end of a simulation, the average residence time in Queues is provided. Other properties concerning the distribution of Queue residence time are not automatically measured and printed out. These other properties may, however, actually be more important measures of system performance than is the average residence time itself. For example, "the percentage of the Queue entries having to wait longer than 10 minutes in the Queue" might be critical in, say, a banking application.

By using the Table entity, the analyst can easily provide logic with which to estimate the distribution of Queue residence time. The two-step sequence involved would be (1) mark Transactions in Pj when they enter the Queue, then (2) when they depart the Queue, move the Transactions through a TABULATE Block referencing a Table whose A Operand is MPj. Precisely the same effect can be achieved by using the Table entity in a special mode, known as QTABLE-mode. The advantages of using QTABLE-mode, instead of providing explicit logic to accomplish the same thing, are (1) there is no need to have a MARK Block at the point the Queue is entered, (2) there is no need to have a TABULATE Block at the point where the Queue is departed, and (3) the execution time required is not as great.

To estimate Queue residence time distribution, all the analyst need do is provide a corresponding Table Definition Card in the model. Such a card differs from other Table Cards in two respects.

(1) The word entered in the Operation Field is QTABLE, not TABLE.

(2) The A Operand on the card is the name of the Queue for which the residence time distribution is to be gathered.

The B, C, and D Operands on the card play their accustomed role; and, as before, the Location Field contains the name of the Table itself.

In Case Study 4C, the average waiting time of customers in a bank was estimated. Except for indicating the relative frequency with which the waiting time was zero, no other information was gathered in that case study about the distribution of the "waiting time random variable". Simply by adding a QTABLE Card to the model, the entire distribution can now be estimated. Figure 4.37 repeats the Figure 4C.2(a) extended program listing for the one-line queuing system, except that now a QTABLE Card has been included (Card 21), and only one day ("Day 1" in Case Study 4C) will be simulated. Figure 4.38(a) shows the Queue statistics produced when the Figure 4.37 model was run; Figure 4.38(b) shows the output for the Table, which has been symbolically named INQUE.

```
BLOCK                                                                          CARD
NUMBER  *LOC    OPERATION  A,B,C,D,E,F,G            COMMENTS                   NUMBER
                SIMULATE                                                          1
        *                                                                         2
        *       SET NON-STANDARD RANDOM NUMBER SEQUENCE FOR DAY 1                 3
        *                                                                         4
                RMULT      111                                                    5
        *                                                                         6
        *       DEFINE FUNCTIONS                                                  7
        *                                                                         8
         5      FUNCTION   RN1,C24          EXPONENTIAL DISTRIBUTION FUNCTION     9
        0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38       10
        .8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2       11
        .97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8                 12
         MEAN  FUNCTION   RN1,D5           DISTRIBUTION OF MEAN SERVICE TIME     13
        .1,450/.29,750/.61,1000/.85,1500/1,3000                                  14
        *                                                                        15
        *       DEFINE STORAGE CAPACITY                                          16
        *                                                                        17
                STORAGE    S$TELRS,8                                             18
        *                                                                        19
        *       DEFINE TABLE                                                     20
         INQUE  QTABLE     ONE,0,600,7      TABLE FOR DIST'N OF TIME IN THE LINE 21
        *                                                                        22
1               GENERATE   180,FN5,,,,1     CUSTOMERS ARRIVE                     23
2               ASSIGN     1,FN$MEAN,5      SET P1 = SERVICE TIME                24
3               QUEUE      ONE              ENTER THE LINE                       25
4               ENTER      TELRS            ENGAGE A TELLER                      26
5               DEPART     ONE              LEAVE THE LINE                       27
6               ADVANCE    P1               TRANSACT BUSINESS                    28
7               LEAVE      TELRS            FREE THE TELLER                      29
8               TERMINATE                    LEAVE THE BANK                       30
        *                                                                        31
        *       MODEL SEGMENT 2                                                  32
        *                                                                        33
9               GENERATE   180000           TIMER ARRIVES AFTER 5 HOURS          34
10              TERMINATE  1                SHUT OFF THE RUN                     35
        *                                                                        36
        *       CONTROL CARDS                                                    37
                START      1                START THE RUN FOR DAY 1              38
                END        RETURN CONTROL TO OPERATING SYSTEM                    39
```

Figure 4.37 Figure 4C.2(a) Extended to Include Tabulation of Queue Residence Time

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
ONE	20	4.552	997	294	29.4	821.883	1165.601	1	8

$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(a) Queue Statistics

TABLE INQUE

ENTRIES IN TABLE	MEAN ARGUMENT	STANDARD DEVIATION	SUM OF ARGUMENTS	
989	826.651	861.000	817558.000	NON-WEIGHTED

UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIATION FROM MEAN
0	294	29.72	29.7	70.2	-.000	-.960
600	226	22.85	52.5	47.4	.725	-.263
1200	143	14.45	67.0	32.9	1.451	.433
1800	164	16.58	83.6	16.3	2.177	1.130
2400	101	10.21	93.8	6.1	2.903	1.827
3000	55	5.56	99.3	.6	3.629	2.524
OVERFLOW	6	.60	100.0	.0		
AVERAGE VALUE OF OVERFLOW		3164.16				

(b) Qtable Statistics

Figure 4.38 Selected Output Produced When the Figure 4.37 Model Was Run

Following the procedure described in Section 4.15, the Processor has assigned 1 as the numeric equivalent of INQUE. In Figure 4.38(a), then, in the column labeled TABLE NUMBER, the entry "1" appears. This is an indication that Qtable statistics are being maintained in Table 1 for the Queue symbolically named ONE. (Up until now, the TABLE NUMBER column in Queue statistics has always been blank.)

Table 4.12 shows a comparison of information taken from the Queue statistics in Figure 4.38(a), and the Qtable statistics in Figure 4.38(b). Some differences in what

	Average Time in Queue	Standard Deviation	Percent Zero Entries
From Figure 4.38(a);	821.883	Not Provided	29.4
From Figure 4.38(b):	826.651	861.000	29.72(4n)

Table 4.12 Comparison of Queue and Qtable Statistics in Figure 4.38

should be "identical values" are evident. These differences should not be surprising. They arise because the computations are performed in finite precision, with the equivalent of about 7 decimal digits carried in single precision on a System/360 computer. When many computations are involved (989 Queue entries, in this case), the error associated with finite precision can accumulate to a noticeable extent. The difference in average Queue residence time in Table 4.13, for example, is about 5 parts in 820, or about 0.6%. Given the other uncertainties inherent in working with non-deterministic simulation models, this discrepancy can easily be tolerated.

Although IAT, RT, and QTABLE Tables are special in mode, they are Tables in every sense. In particular:

(1) They appear as part of standard model output.

(2) They can be outputted dynamically by use of the PRINT Block.

(3) The three Table Standard Numerical Attributes apply to them.

(4) They can be defined to be weighted Tables.

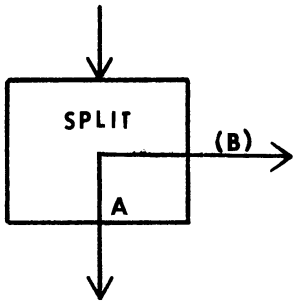
No "special mode" Table names can be identical to those chosen for "normal mode" Tables in a given model. If Table 7, say, is an IAT-mode Table, then 7 cannot be chosen as the name of any other Table in the model.

(4n) The "percent zero entries" information is available in the Qtable as the PERCENT OF TOTAL in the Table interval with an UPPER LIMIT of 0.

4.28 Propagation of Transactions: The SPLIT Block

The primary method for bringing Transactions into a model is to use one or more GENERATE Blocks. There is also a secondary method for bringing Transactions into a model. The method requires use of the SPLIT Block. Whenever a Transaction already in a model moves into a SPLIT Block, one or more additional Transactions enter the model as a consequence of the execution of the SPLIT Block subroutine. Hence, Transactions have the capability of "propagating" themselves.

The SPLIT Block, and its A and B Operands, are shown in Figure 4.39. It is



Operand(4oh)	Significance	Default Value or Result
A	The number of additional transactions to be brought into the model	Error
B	Name of the Block Location to which these additional Transactions are to be sent	Error

Figure 4.39 The SPLIT Block, and Its A and B Operands

convenient to think of the Transaction entering the SPLIT Block as the "parent". The additional Transactions brought into the model are termed the "offspring". When a parent moves into the SPLIT Block, the A Operand's value is the number of offspring to be brought into the model as a result. These offspring are identical to the parent in most respects.

(1) Their Priority Level is the same.

(2) They have the same number of Parameters; the Parameter type (i.e., halfword or fullword) is identical to that of the parent; and the Parameter values are identical to those of the parent.

(3) Their Mark Time is identical to that of the parent.

The offspring, however, are different Transactions. In particular, the Transaction number of each offspring differs from that of the parent, and from that of the other offspring.

When the parent leaves the SPLIT Block, it moves unconditionally to the sequential Block. The offspring, however, are routed to the Block in the Location whose name is used as the SPLIT Block's B Operand. This is normally a non-sequential Block.(4p) It is possible, though to send the offspring to the sequential Block, along with the parent. To do this, a symbolic Location Name is given to the sequential Block, and this name is specified as the SPLIT Block's B Operand.

(4oh) Two additional, optional SPLIT Block Operands will not be introduced until Chapter 6.

(4p) Actually, the SPLIT Block's B Operand is evaluated separately, offspring-by-offspring. Hence, it is possible to make the Block to which an offspring is sent be a function of the offspring itself. There is no need, however, to consider these complications in this chapter.

Consider the operation of the SPLIT Block from the point of view of chains. First of all, the SPLIT Block cannot deny entry to a Transaction. When the parent enters the Block, its processing is temporarily delayed while all the offspring are fetched from the latent pool and brought into the model. The offspring are fetched one by one. Each next offspring is put immediately on Current Events Chain, as the last member in its Priority Class. When all offspring have been hooked onto the CEC in this fashion, the processing of the parent is resumed. Later, as the scan of the Current Events Chain continues, the offspring Transactions are encountered by the Processor. Each of the offspring is then picked up, in turn, and moved as far forward in the model as possible, in the usual fashion.

For the SPLIT Block, the Standard Numerical Attributes Wj (Current Block Count) and Nj (Total Block Count) are incremented by one for the parent and for each offspring Transaction when the Block subroutine is executed. The Current Block Count is decremented by 1 each time any one of these Transactions succeeds in moving from the SPLIT Block to another Block. If the parent or any offspring are denied entry to their next intended Block, they are held at the SPLIT, where they continue to contribute to the Current Block Count.

Case Study 4E illustrates use of the SPLIT Block in context.

4.29 Case Study 4E: A Spare Parts Problem

(1) Statement of the Problem

A certain machine uses a type of part which is subject to periodic failure. Whenever the in-use part fails, the machine must be turned off. The failed part is then removed, a good spare part is installed if available, or as soon as one becomes available, and the machine is turned on again. Failed parts can be repaired and re-used.

The lifetime of a part follows the distribution shown in Table 4E.1. The time

Lifetime, Hours	Cumulative Frequency
Less than 100	0.0
125	0.10
150	0.26
175	0.51
200	0.76
225	0.90
250	1.0

Table 4E.1 Distribution of Part's Running Time Before Failure

required to remove a failed part, and to install a replacement, is negligibly small. It takes 16±4 hours to repair a failed part.

There is only one repairman available. His responsibilities include repair of items routed to him from another source. These other items arrive in a Poisson stream with a mean inter-arrival time of 100 hours. Their service time requirement is 75±15 hours, and they always have the highest repair priority.

The system is shown graphically in Figure 4E.1. Each open circle represents either a good part in "spare" status, or a part currently in the machine. Circles with an X in them represent parts in "failed" status. The other items which compete for the repairman's services are shown as squares with an X in them. As indicated above and as the flow schematic suggests, these other items have a higher repair priority than the failed parts; they enter the repair queue ahead of any failed parts which may be waiting for repair.

Build a GPSS model for this system, then use the model to estimate the fractional utilization of the machine as a function of the total number of parts which circulate through the system. Study the system behavior for the case of 5, 6, and 7 circulating parts. Make the experimental conditions as otherwise nearly identical to each other as possible.

(2) Approach Taken in Building the Model The Supply of Parts, and the Machine

It was indicated in Chapter 2 that a useful modeling approach is to first identify system constraints, then decide which GPSS entities to use to simulate these constraints. In the "spare parts problem", the only constraint (except for the single repairman) is the number of parts which circulate through the system. Because these parts can be thought of as "moving", as suggested in Figure 4E.1, it would be natural to let a Transaction simulate a part. Using the GENERATE Block's Limit Count feature, the required number of Transactions could then be brought into the model at the start of the simulation, and so on. For variety, this "natural" approach is not followed

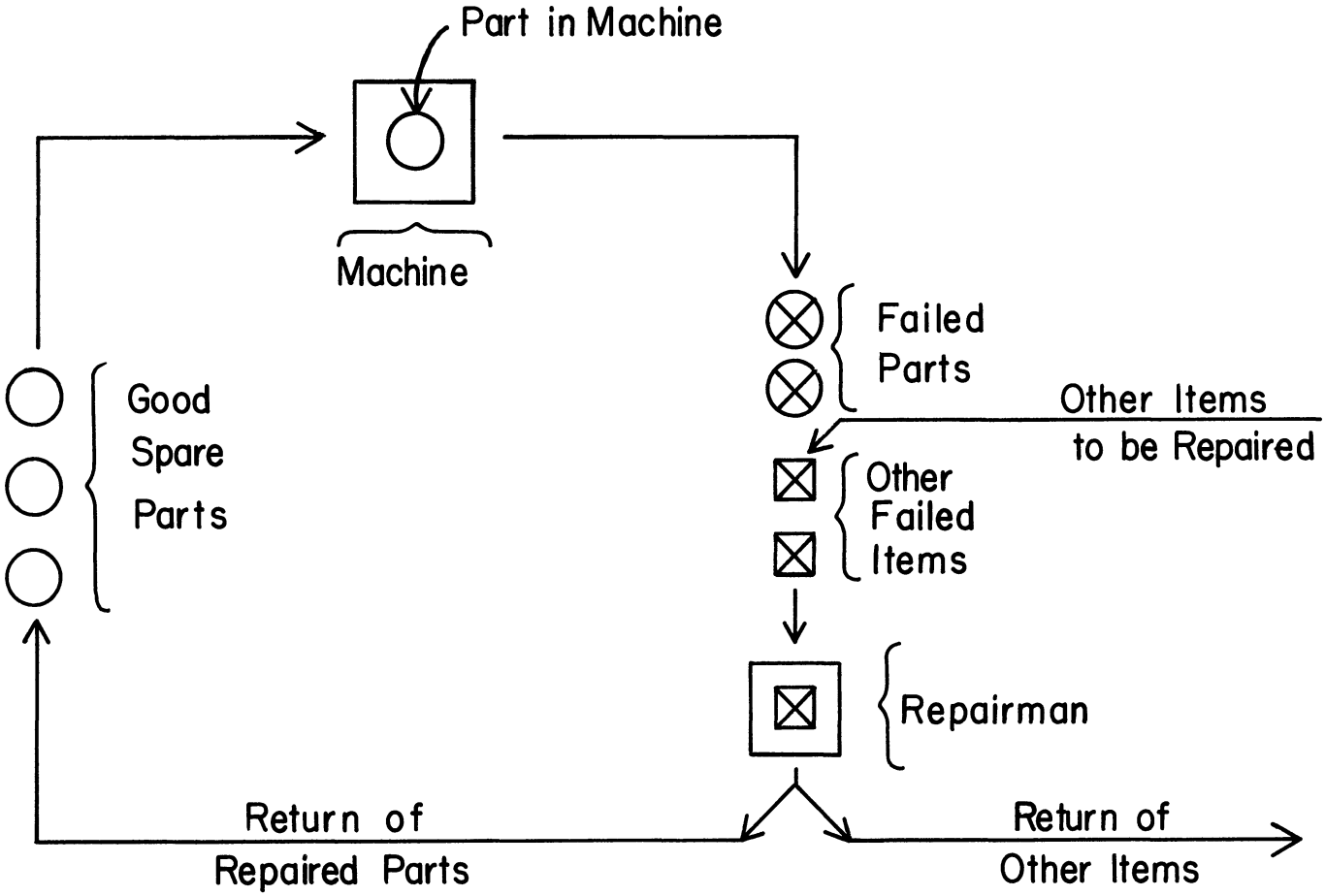


Figure 4E.1 Flow Schematic for Case Study 4E

here, but is left to the exercises [see Problems 4.30.9 and 4.30.10(b)].

The approach taken here is to let a Storage simulate the circulating parts. The name chosen for the Storage is PARTS. The total capacity of the Storage equals the total number of parts in the system. The remaining capacity of the Storage (R$PARTS) is interpreted as the number of "available parts", i.e., those ready to be put into immediate use. The current content of the Storage (S$PARTS) is understood to equal the number of "not available" parts, i.e., the part currently in the machine (if any), and the parts at the repair facility.

The scheme for use of the Storage PARTS is rounded out by interpreting a Transaction in the model as a "worker". To remove an available spare part from the shelf, the worker-Transaction must move into an "ENTER PARTS" Block. If the remaining capacity of the Storage is zero, he is denied entry to the Block (and is thereby prevented from moving further to "install the part and turn on the machine"). On the other hand, if R$PARTS is 1 or more, the worker-Transaction successfully moves into the "ENTER PARTS" Block, thereby decreasing R$PARTS by 1 (and updating the "available parts" situation as a result). Moving on from the ENTER Block, the worker can then "install the part", "turn on the machine", and so on.

When the repair of a failed part has been completed, "available parts" information is updated by having the worker-Transaction move through a "LEAVE PARTS" Block. This increases R$PARTS by 1, reflecting an increase in the number of available parts. It simultaneously decreases S$PARTS by 1, reflecting a decrease in the number of "unavailable parts".

The machine itself is simulated with the Facility MAC. A worker-Transaction "turns on the machine" by capturing the Facility, and "turns it off" by releasing it. This means that the utilization of the Facility equals the machine utilization. Hence, use of a Facility to simulate the machine provides a convenient method for measuring the requested utilization statistic.

At first glance, it might seem that only one worker-Transaction would be required in the model segment simulating the supply of parts and the machine (Model Segment 1). This is true when the machine is running, and no parts (except for the one in use) are in "not available" status. When the in-use part fails, however, two independent chains of events are triggered in the system. First of all, the failed part must be taken to the repair facility, where repair eventually takes place, and records are updated to indicate the repaired part is again ready for use. Meantime, the supply of spares must be checked to determine whether a spare is available, the spare must be installed, the machine must be turned back on, etc. Because these separate chains of events should logically proceed simultaneously, they cannot be caused by a single Transaction. As a result, when an in-use part fails, the worker-Transaction moves into a SPLIT Block, thereby "calling in a co-worker". The co-worker is routed to the "ENTER PARTS" Block to check for an available spare, etc. Meantime, the "parent" worker takes responsibility for having the failed part fixed. After eventually moving through the "LEAVE PARTS" Block when the repair has been completed, this "parent" then terminates, leaving the co-worker to carry on.

The Repair Facility

The repairman is simulated with the Facility FIXER. In the machine-and-parts model segment, a worker-Transaction puts a failed part into repair by capturing this Facility. He must compete, however, with "other items" for use of the Facility. This other use is simulated in a separate, self-contained model segment (Model Segment 2). In the separate segment, Transactions simulating the "other items" enter the model with a high priority, queue for use of the repairman and, after this use, leave the model. The separate segment, then, is completely straightforward. (The possibility of having a single Facility be accessible from each of two distinct, self-contained model segments was first introduced in Case Study 2B.)

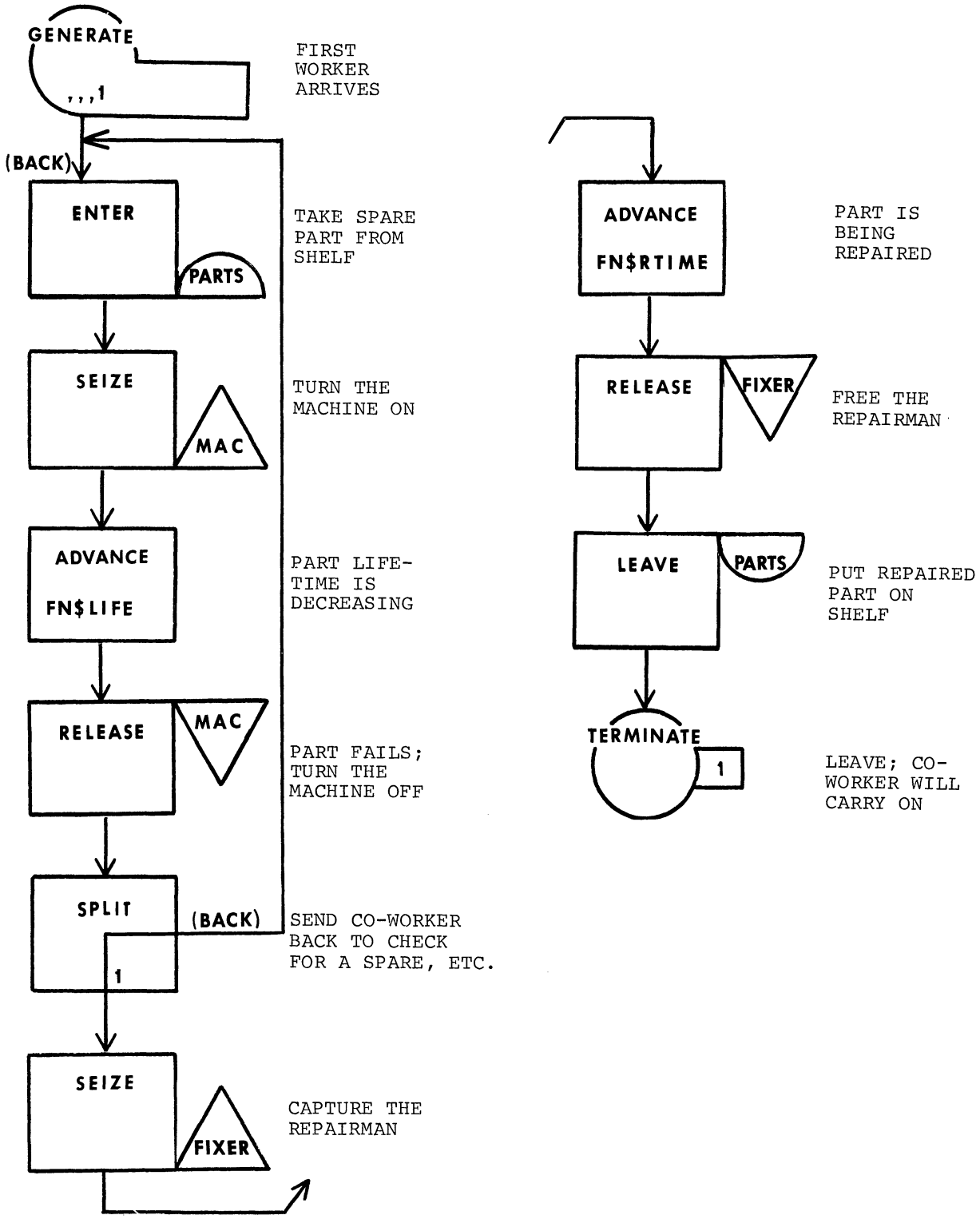
(3) Table of Definitions

Time Unit: 1 Hour

GPSS Entity	Interpretation
Transactions	
Model Segment 1	A "worker"
Model Segment 2	An "other item"
Facilities	
MAC	The machine whose utilization is being measured
FIXER	The repairman
Functions	
LIFE	Function describing the lifetime distribution for the type of part used by the machine
OTHER	Function describing the repairtime distribution of "other items" at the repair facility
RTIME	Function describing the repairtime distribution for the type of part used by the machine
XPDIS	Function for sampling from the exponential distribution with a mean value of 1
Storages	
PARTS	Storage simulating the parts used by the machine. Total Storage capacity equals the total number of parts in the system. The "remaining capacity" and "current content" equal the number of currently "available" and "unavailable" parts, respectively

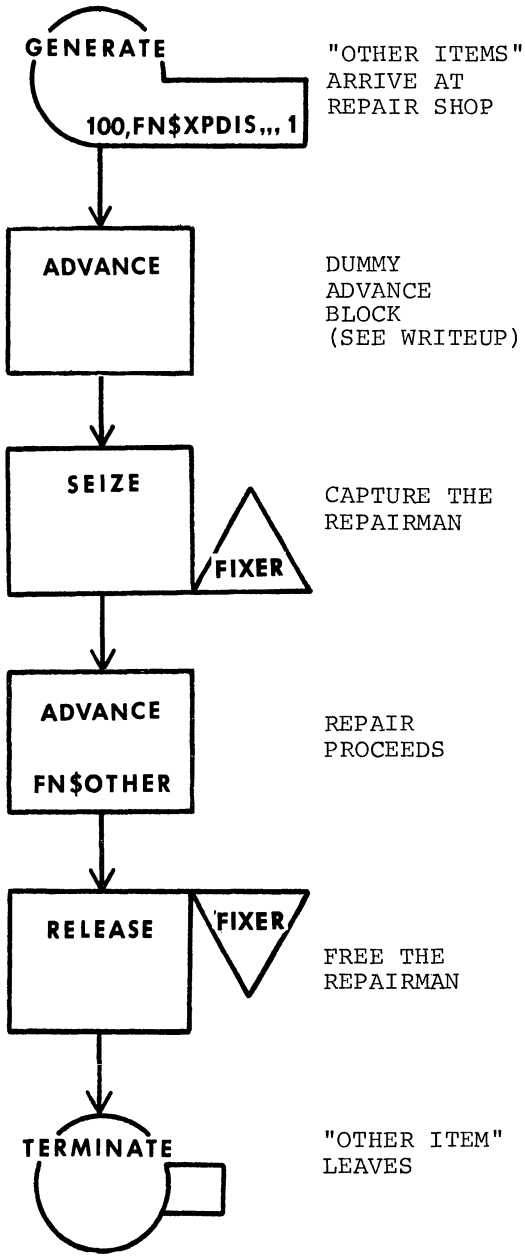
Table 4E.2 Table of Definitions for Case Study 4E

(4) Block Diagram



MODEL SEGMENT 1

Figure 4E.1 Block Diagram for Case Study 4E



MODEL SEGMENT 2

(5) Extended Program Listing

```
BLOCK                                                                    CARD
NUMBER  *LOC   OPERATION  A,B,C,D,E,F,G            COMMENTS              NUMBER
               SIMULATE                                                      1
        *                                                                    2
        *      SET RANDOM NUMBER SEQUENCES FOR 1ST RUN                       3
        *                                                                    4
               RMULT      321,123,753,357                                    5
        *                                                                    6
        *      DEFINE FUNCTIONS                                              7
        *                                                                    8
         LIFE  FUNCTION   RN1,C7      LIFETIME DISTRIBUTION OF PART-TYPE     9
        0,100/.1,125/.26,150/.51,175/.76,200/.9,225/1,250                    10
         OTHER FUNCTION   RN2,C2      REPAIRTIME DISTRIBUTION OF "OTHER ITEMS" 11
        0,60/1,91                                                            12
         RTIME FUNCTION   RN3,C2      REPAIRTIME DISTRIBUTION OF PART-TYPE   13
        0,12/1,21                                                            14
         XPDIS FUNCTION   RN4,C24     EXPONENTIAL DISTRIBUTION FUNCTION      15
        0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38   16
        .8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2   17
        .97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8             18
        *                                                                    19
        *      DEFINE STORAGE CAPACITY                                       20
        *                                                                    21
               STORAGE    S$PARTS,5                                         22
        *                                                                    23
        *      MODEL SEGMENT 1                                               24
        *                                                                    25
1              GENERATE   ,,,1            FIRST WORKER ARRIVES              26
2       BACK   ENTER      PARTS           TAKE SPARE PART FROM SHELF        27
3              SEIZE      MAC             TURN THE MACHINE ON               28
4              ADVANCE    FN$LIFE         PART LIFETIME IS DECREASING       29
5              RELEASE    MAC             PART FAILS;  TURN THE MACHINE OFF 30
6              SPLIT      1,BACK          SEND CO-WORKER BACK TO CHECK      31
        *                                 FOR A SPARE PART, ETC.            32
7              SEIZE      FIXER           CAPTURE THE REPAIRMAN             33
8              ADVANCE    FN$RTIME        PART IS BEING REPAIRED            34
9              RELEASE    FIXER           FREE THE REPAIRMAN                35
10             LEAVE      PARTS           PUT REPAIRED PART ON SHELF        36
11             TERMINATE                   LEAVE;  CO-WORKER WILL CARRY ON   37
        *                                                                    38
        *      MODEL SEGMENT 2                                               39
        *                                                                    40
12             GENERATE   100,FN$XPDIS,,,1 "OTHER ITEMS" ARRIVE AT REPAIR SHOP 41
13             ADVANCE                     DUMMY ADVANCE BLOCK (SEE WRITEUP) 42
14             SEIZE      FIXER           CAPTURE THE REPAIRMAN             43
15             ADVANCE    FN$OTHER        REPAIR PROCEEDS                   44
16             RELEASE    FIXER           FREE THE REPAIRMAN                45
17             TERMINATE                   "OTHER ITEM" LEAVES               46
        *                                                                    47
        *      MODEL SEGMENT 3                                               48
        *                                                                    49
18             GENERATE   25000          TIMER ARRIVES AFTER 25,000 HOURS    50
19             TERMINATE  1               SHUT OFF THE RUN                  51
        *                                                                    52
        *      CONTROL CARDS                                                 53
        *                                                                    54
               START      1               START 1ST RUN (5 PARTS IN SYSTEM) 55
               RMULT      321,123,753,357 RESTORE SEQUENCES FOR 2ND RUN     56
               STORAGE    S$PARTS,6       RE-CONFIGURE FOR 2ND RUN          57
               CLEAR                       CLEAR FOR 2ND RUN                 58
               START      1               START 2ND RUN (6 PARTS IN SYSTEM) 59
               RMULT      321,123,753,357 RESTORE SEQUENCES FOR 3RD RUN     60
               STORAGE    S$PARTS,7       RE-CONFIGURE FOR 3RD RUN          61
               CLEAR                       CLEAR FOR 3RD RUN                 62
               START      1               START 3RD RUN (7 PARTS IN SYSTEM) 63
               END        RETURN CONTROL TO OPERATING SYSTEM                64
```

Figure 4E.3 Extended Program Listing for Case Study 4E

(6) Program Output

No program output is shown directly. Results are presented in the next section.

(7) Discussion

Model Logic

In the machine-and-parts portion of the model (Model Segment 1), note that the Facility MAC does not impose a constraint. That is, whenever a worker-Transaction "wants" to capture the Facility (by seizing at Block 3, Figure 4E.3), the capture can always occur. It is the "ENTER PARTS" Block (Block 2) preceding the SEIZE that simulates the constraint.

In Model Segment 2, the Block following the GENERATE is an ADVANCE with no Operands specified (Block 13, Figure 4E.3). Because of the implied zero holding time, this "dummy" ADVANCE Block does not delay "other items" entering the model through the preceding GENERATE Block; on the other hand, its presence guarantees that whenever an "other item" arrives, it can move immediately from the GENERATE Block, causing the Processor to pre-schedule arrival of the next "other item". If the dummy ADVANCE Block were not included, the inter-arrival time pattern of "other items" could be distorted, as discussed in Chapter 2.

The problem statement indicates that experimental conditions are to be replicated to the extent possible. To do this, a separate Function has been defined to describe each of the four sources of randomness in the model (part lifetime; part repairtime; "other item" arrivals; and "other item" repairtime). Each Function uses a separate source of random numbers as its argument. With the RMULT Card (Card 4, Figure 4E.3), the random number generators involved have been set to randomly-chosen starting points. Whenever the system is re-configured by changing the capacity of the Storage PARTS (Cards 57 and 61, Figure 4E.3), the generators have also been re-set to their initial starting points (Cards 56 and 60).

Program Output[4q]

Table 4E.3 summarizes the program output. The machine utilization, shown in

		Facility Utilization		Statistics for the Storage PARTS	
		MAC	FIXER	Average Content	Average Utilization
Number of Parts in System	5	0.849	0.755	2.318	0.463
	6	0.867	0.757	2.476	0.412
	7	0.891	0.759	2.640	0.377

Table 4E.3 Summary of Program Output for Case Study 4E

column 1 in the table, increases slowly as the number of circulating parts increases. Utilization of the repairman, shown in column 2, is quite insensitive to the number of circulating parts. The number of "unavailable parts" (S$PARTS, shown in column 3 of the table), increases slightly as the total number of circulating parts increases. The number of "available parts" (the capacity-complement of the column 3 entries) is about 2.7, 3.6, and 4.4 on average for the three configurations, respectively. These results suggest that it is the low repair priority which constrains machine utilization, not the number of circulating parts. In Problem 4.30.8, the queue discipline used by the repairman, and its cost implications, are to be further investigated.

[4q] Total CPU time for the simulation was 7.1 seconds.

4.30 Exercises

4.30.1 Solve Problem 4.26.5 using a Qtable instead of a Table. Which of the two alternative solutions is relatively better, and why?

4.30.2 (a) Referring to Figure 4.38(b), what percentage of the "Day 1" customers had to wait more than 4 minutes in the one-line bank model?

(b) Show how to modify the multiple-line bank model in Figure 4C.2(b) to estimate the aggregate customer waiting time distribution.

(c) Re-run the Case Study 4C simulations for the one-line and multiple-line queuing systems to estimate the waiting time distributions. Use the output to compute (by hand) the mean and standard deviation of the random variable, "percentage of customers who had to wait longer than 4 minutes". Which queuing system appears to be superior with respect to this statistic?

4.30.3 Show what changes to make in the indicated models to estimate the distributions of the indicated random variables.

(a) Case Study 2A, distribution of time between consecutive service completions.

(b) Case Study 2A, distribution of rate of service completions.

(c) Case Study 2B, time-in-service for "shave-and-haircut" customers.

(d) Case Study 2C, inter-arrival time of mechanics at the tool crib (without making distinctions as to the type of mechanic).

(e) Case Study 2E, rate of arrival of failed sewing machines at the repair facility.

(f) Case Study 2E, inter-arrival time of failed sewing machines at the repair facility.

(g) Case Study 2F, inter-arrival time of television sets at the last inspection station.

4.30.4 Develop a GPSS model which uses the Table entity to determine the distribution of the first 1,000 values produced by RN1, for 5 different initial settings of the random number generator. (The RN1 values tabulated should be three-digit integers.) The Table should consist of the intervals from 0 to 49, 50 to 99, 100 to 149, ..., and 950 to 999. Run the model, then for each of the 5 Tables produced compare (a) the Table average, (b) the Table standard deviation, and (c) the observed frequencies for each Table interval with the corresponding theoretical expectations.

4.30.5 Whenever a Transaction reaches Point A in a model, another Transaction is to enter the model 15 ± 7 time units later, moving to the Block at THERE when it arrives. Show how this can be accomplished.

4.30.6 Case Study 2E involves a system in which a group of machines operate in parallel, with each machine subject to periodic failure. In the spirit of Case Study 4E, show how to use the SPLIT Block in building a model for that system.

4.30.7 For the Case Study 4E system, build and run GPSS models to determine what the cost per hour of a down machine would have to be so that downtime cost would match the waiting costs (excluding service) of "other items" at the repair facility, computed as $10 per hour per item, for the alternative queue disciplines indicated below. Assume throughout that 6 parts circulate through the system.

(a) "Other items" have a higher priority than parts.

(b) No priority distinctions are made.

(c) Parts have higher priority than "other items".

4.30.8 In Case Study 4E, the parts circulating through the system are simulated with a GPSS Storage. A Transaction is used to simulate a "worker" who moves about in the system, and who occasionally calls upon a co-worker when system conditions require that independent activities proceed in two different portions of the system. As Figure 4E.1 suggests, it may be more natural to think of a part as moving through the system, instead of a worker. This leads to the possibility of letting a Transaction simulate a part. Aside from Transactions representing the "other items", the number of Transactions in the model would then equal the total number of available parts; the "status" of a part would depend on where the corresponding Transaction happened to be in the model. Based on this idea, build an alternative GPSS model for the Case Study 4E problem. Show the model in the form of a Block Diagram. To the extent possible, make use of the Functions already defined for the case study.

4.30.9 (a) Show how to modify the Case Study 4E model under the assumption that the time required to remove a failed part from the machine is 3 ± 1 hours and the time to install another part is 5 ± 2 hours. The machine must be off when a part is being removed or installed. In the long run, what is the maximum machine utilization that can possibly be realized?

(b) Repeat (a) for the model developed as a solution to Problem 4.30.8. Does the case study model, or the solution to Problem 4.30.8, lend itself more readily to this simple extension?

4.30.10 (a) A machine uses two parts of a particular type which is subject to periodic failure. When either part fails, the machine must be turned off. The part having the shortest lifetime, then, determines how long the machine will run after it has been turned on.
The machine is to be modeled in GPSS. A Transaction simulates a worker. Parameters 1 and 2 contain the lifetimes of the two parts which have just been installed in the machine. The worker is to capture the Facility MAC (thereby simulating the event, "turn on the machine"), then release the Facility as soon as a failure occurs. Show a Block sequence which accomplishes this effect.

(b) Assume that the machine in Case Study 4E is replaced with the machine described in (a). Also assume that the information in Table 4E.1 still describes the lifetime distribution of the part type in question. Show all details for a GPSS model to simulate the Case Study 4E system under these circumstances. (Caution: when the machine is on, the "remaining lifetime before failure" of each installed part is decreasing. Suppose, for example, that the machine is turned on

when the two installed parts have 154 and 189 running hours remaining, respectively, before failure. Then, when the machine is turned off 154 hours later because one part fails, only 35 running hours remain for the other installed part before it will fail.)

4.30.11 A certain machine uses one part of Type A, and one part of Type B. Each of these types of parts is subject to periodic failure. Whenever one of the parts fails, the machine is shut down, the failed part is removed and replaced if or as soon as a spare is available, and the machine is turned on again. Failed parts are repaired and can then be re-used.

The data in Table T4.30.11 are available for the two part types. All of

	Lifetime	Repair Time*	Installation Time
Type A	235±75	336±50	4±2
Type B	375±95	280±40	3±2

*Actually, time lag before the part is again available. Failed parts are sent to a central repair department, where long delays are encountered.

Table T4.30.11

the times are in hours. Time required to remove a failed part from the machine is negligibly small.

Construct a GPSS model of this system to determine how the machine utilization varies with the number of Type A and Type B parts which circulate through the system. (Compare and contrast this problem with Problem 4.30.10.)

Chapter 5
Intermediate GPSS Modeling Concepts, Part II

5.1 Introduction

In the GPSS modeling introduced so far, the analyst has not had to indicate how computations are to be performed. It has been assumed that the Processor "knows how" to maintain and update such things as Facility utilization, Table standard deviation, average Queue residence time, and so on. Such computations are straightforward, and it is highly appropriate that the GPSS Processor does them for the analyst, automatically. On the other hand, many computations are highly context-dependent, i.e., are peculiar to the system being modeled and the corresponding logic being implemented. The Processor cannot be designed to anticipate all of these customized situations and handle them automatically. As a result, provisions are made for the analyst to construct arithmetic expressions of his own choosing, then use them in the logic of his model. These user-defined Arithmetic Variables will be discussed in this chapter.

Furthermore, all models have been constructed thus far without the benefit of user-supplied tests being part of the models. Granted, a wealth of testing is required to support the logic of the models presented in the earlier chapters. But, from a Block-oriented viewpoint, this testing has taken place exclusively at SEIZE and ENTER Blocks, at the SELECT Block, and at TRANSFER Blocks in statistical mode and BOTH mode. The type of testing that takes place at these Blocks is pre-defined and is relatively inflexible. Such Blocks cannot support a full range of customized decision-making. Suppose, for example, that a Transaction is to move forward in a model only if the sum of two numbers is less than or equal to the value of some other number. The ability to include such conditions in GPSS models by use of the TEST Block will be described in this chapter.

Not all tests the analyst might like to build into a model involve numeric relationships. It is often convenient to make model behavior depend on the logical status of certain entities within the GPSS language. It might be that a Transaction is not to move forward at one point in a model until some Facility elsewhere in the model has been captured. Use of the GATE Block to directly and efficiently model conditions such as this, which involve the logical status of selected GPSS entities, will be included in this chapter.

Finally, the possibility of introducing arbitrarily complex logical conditions into a model will be considered. Suppose, for example, that a ship cannot move into a harbor until (a) no storm is in progress, (b) an empty berth is available, and (c) a tug can be committed to it. Truth-valued variables can be used to determine whether conditions such as these are simultaneously satisfied. Definition and use of such user-supplied Boolean Variables is described here.

Among the items taken up in this chapter, then, are Arithmetic Variables, the TEST Block, the GATE Block, and Boolean Variables. In addition to these aspects of the language, the GPSS entities knows as Savevalues and Logic Switches will also be introduced.

5.2 Arithmetic Variables

In GPSS, an Arithmetic Variable is a user-defined Standard Numerical Attribute. The name of an Arithmetic Variable is Vj, or V$sn, where "j" is the number of the Variable if it has been named numerically, and "sn" is its symbolic name if it has been

named symbolically. The value of an Arithmetic Variable is the value of the user-supplied arithmetic expression which defines that Variable. An arithmetic expression, in turn, is a collection of data specifications connected by arithmetic operators. In succeeding parts of this section, the discussion will focus on the arithmetic operators available in GPSS; the specification of data in arithmetic expressions; the Variable Definition Card; examples of Variables and the rules followed in evaluating them; and, finally, a special case of Variable definition available at the user's option.

5.2.1 The Arithmetic Operators

The arithmetic operators available in GPSS are shown in Table 5.1. The first

Symbol	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Integer Division
@	Modulus Division

Table 5.1 The Arithmetic Operators Available in GPSS

three operators in the list are well known. The last two, integer division and modulus division, may require some elaboration.

In integer division, only the integer portion of the quotient is retained as the result of the operation. The fractional part of the quotient, if any, is discarded. For example, 17/4 equals 4. The whole quotient resulting when 17 is divided by 4 is 4.25. The fractional part of this whole quotient, 0.25, is discarded; the integer portion, 4, is retained as the result of the operation. Similarly, 25/3 equals 8; -21/6 equals -3; and 10/5 equals 2. As suggested by the second-to-last example, the algebraic sign carried by the result is determined by the usual laws of signs. The last example shows a case in which the result is exact, because the quotient has no fractional part.

The fifth and last operator in Table 5.1, @, denotes modulus division. In modulus division, the integer portion of the quotient is discarded and the remainder is retained as the result of the division. For example, 13 divided by 3 is 4, with a remainder of 1. In modulus division, the 4 is discarded and the 1 is retained as the result of the division. This is usually expressed more compactly by saying that "13 modulus 3 is 1", or by writing "13@3 = 1". The examples of modulus division in Figure 5.1 should be reviewed to be certain that the modulus division concept is clearly understood.

	Spoken Form	Written Form
(a)	52 modulus 25 is 2	52@25 = 2
(b)	17 modulus 6 is 5	17@6 = 5
(c)	3 modulus 5 is 3	3@5 = 3
(d)	8 modulus 10 is 8	8@10 = 8
(e)	7 modulus 7 is 0	7@7 = 0
(f)	54 modulus 9 is 0	54@9 = 0

Figure 5.1 Examples of Modulus Division

5.2.2 Data Specification in Arithmetic Expressions

In the expressions used to define Arithmetic Variables, data can be specified either directly, or indirectly. Data specified directly takes the form of integer constants. Data specified indirectly takes the form of Standard Numerical Attributes. All Standard Numerical Attributes which have been described can be validly used in the construction of arithmetic expressions. In fact, because they are also SNA's, the expression defining an Arithmetic Variable can include references to one or more other Arithmetic Variables in the model.

Consider what happens when a Function is used to indirectly provide data in an arithmetic expression. As pointed out earlier, only the integer portion of a Function's true value is used as data in most contexts. This is also true when Functions are used in constructing arithmetic expressions.

This means that all the data involved in an arithmetic expression are integers. When constants are used to directly specify data in such expressions, they must be integers. All the SNA's that might be used to indirectly specify data in arithmetic expressions are also integer-valued. When the operators +, -, and * are used to combine integer data, the results are other integers. As explained above, when the operators / and @ operate on pairs of integers, integer results are also produced. In short, at each step of the way in evaluating an arithmetic expression, only integers are involved. Arithmetic Variables, then, are themselves inherently integer-valued.

5.2.3 The Variable Definition Card

An Arithmetic Variable is defined by use of a Variable Definition Card, sometimes simply called a Variable Card. Like most other cards, the Variable Card is divided into the Operation, Location, and Operands fields. The information entered in these fields is shown in Table 5.2. The name of the Variable appears in the Location Field, and the word

Punchcard Field	Information Supplied in the Field
Location	The name (numeric or symbolic) of the Arithmetic Variable
Operation	Literally, the word VARIABLE
Operands	The arithmetic expression which defines the Arithmetic Variable

Table 5.2 Variable Definition Card Specifications

VARIABLE is entered in the Operation Field. The arithmetic expression is placed in the Operands Field. It must begin in card column 19 and continue in consecutive card columns, without any intervening blanks. The first blank column encountered in the Operands Field terminates the expression. The expression cannot extend beyond card column 71. Furthermore, if an expression is too long for a single card, it cannot be continued on a next card. In such cases, the expression must be broken up into a series of shorter expressions and defined through two or more Arithmetic Variables.

5.2.4 Examples and Rules

Four examples of Arithmetic Variables are shown in Figure 5.2. The Variable JOE has as its value the remaining capacity of Storage 5, plus the current content of Storage 5. Of course, "remaining capacity" plus "current content" of a Storage equals its user-defined capacity. There is no Processor-supplied Standard Numerical Attribute whose value is the user-defined capacity of a Storage. As a result, this example shows how the user can construct an Arithmetic Variable having Storage capacity as its value.

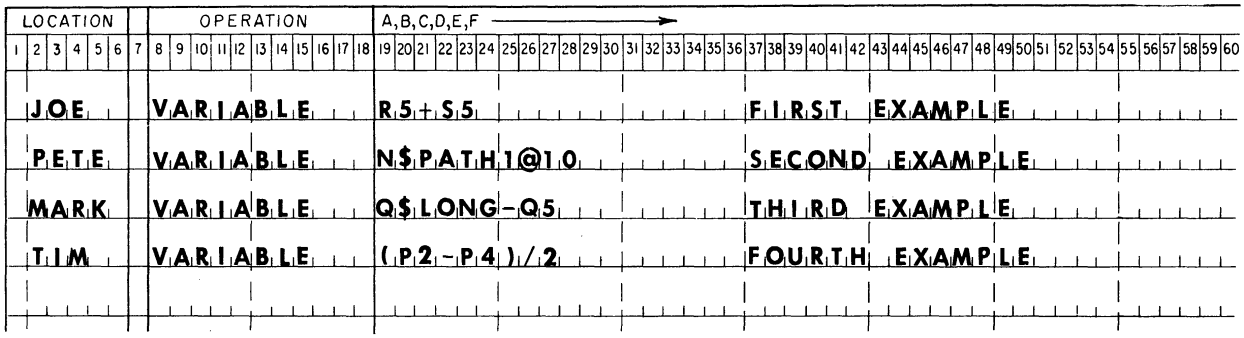


Figure 5.2 A First Set of Examples of Arithmetic Variables

The Variable PETE in Figure 5.2 is the Total Block Count at the Block in Location PATH1, modulus 10. Note that the range of values V$PETE can assume consists of the ten integers 0, 1, 2, 3, ..., 8, and 9. This restriction on the range of values is a natural consequence of modulus division.

The Variable MARK is defined as the current content of the Queue LONG, minus the current content of the Queue 5. The Variable TIM is defined as the value of Parameter 2, minus the value of Parameter 4, divided by 2. Two points are raised by the Variable TIM.

(1) When Parameters indirectly specify data for Arithmetic Variables, it is the active Transaction whose Parameters are used. An Arithmetic Variable is only evaluated at the time a Transaction moves into a Block whose Operand values depend in some way on the Variable. It is that Transaction whose Parameters are used.

(2) Parentheses can be used in the expressions defining Arithmetic Variables. When parentheses are used, everything within them is evaluated first, then the result is used in the rest of the expression. This is the so-called "rule of parentheses".

Now consider the set of examples of Arithmetic Variables in Figure 5.3. All three examples raise a question with respect to the time-sequence in which the various

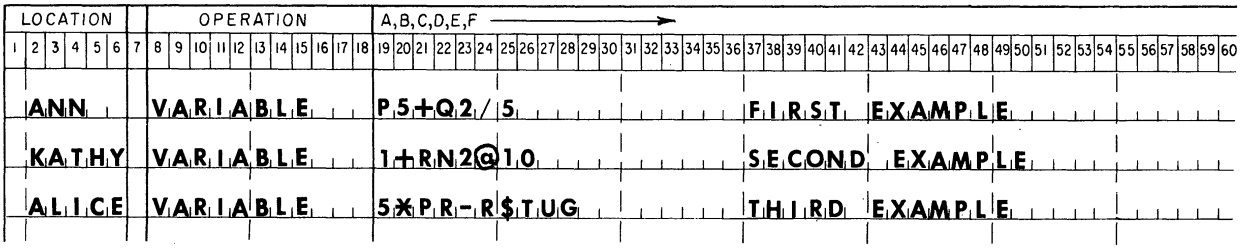


Figure 5.3 A Second Set of Examples of Arithmetic Variables

arithmetic operators have their effect. For example, if P5 is 5 and Q2 is 15, then V$ANN might be 4 (add P5 to Q2 first, then divide by 5); or, V$ANN might be 8 (divide Q2 by 5 first, then add the result to P5). The question, then, is whether one adds first, then divides; or divides first, and then adds. The "rule of precedence" resolves this question. A precedence order is imposed on the operators, with the understanding that their effect takes place in that order. Table 5.3 shows the precedence ordering for the five operators available. The "rule of precedence" is that, when unaltered by parentheses, the order of arithmetic operations performed within an expression is in descending order of precedence. Hence, from Table 5.3, multiplication, division, and modulus division each precede addition and subtraction in the absence of parentheses. This means that the Variable ANN in Figure 5.3 is evaluated by "dividing

Q2 by 5 first, then adding the result to P5".

Operator	Precedence Level
*, /, and @	Highest
+ and -	Lowest

Table 5.3 Precedence Levels of the Arithmetic Operators

Similarly, the Variable KATHY is evaluated by "modulus-dividing RN2 by 10 first, then adding 1 to the result". Recall that, except when used as a Function argument, RNj returns a random number from the uniform distribution of integers between 000 and 999, inclusive. In the Variable KATHY, RN2@10 is consequently a random number from the uniform distribution of integers between 0 and 9, inclusive. Adding 1 to this intermediate result, V$KATHY is a random variable taking on values uniformly distributed over the closed interval of integers between 1 and 10.

In the last Figure 5.3 example, the value of V$ALICE is formed by taking 5 times the PR value of the active Transaction, then subtracting from this product the remaining capacity of the Storage named TUG.

Note that the "rule of precedence" is conditioned on arithmetic expressions being unaltered by parentheses. When parentheses are used, everything within them is evaluated first, then this intermediate result is used in the rest of the expression. Of course, when evaluating expressions within parentheses, the rule of precedence applies. The fourth example in Figure 5.2 has already provided an example of how parentheses can be used to force subtraction to occur before division takes place.

A third set of examples of Arithmetic Variables is given in Figure 5.4. In

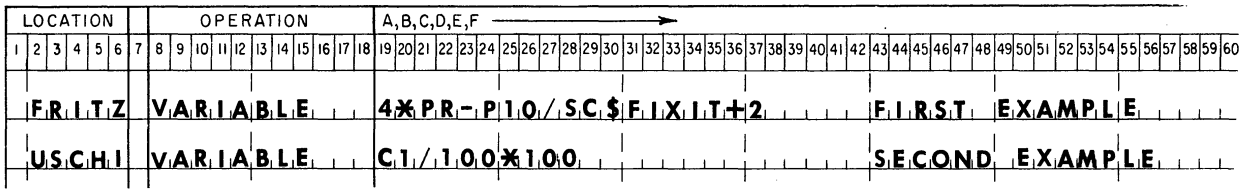


Figure 5.4 A Third Set of Examples of Arithmetic Variables

Table 5.3, note that *, /, and @ share a common precedence level. So do + and -. In both of the Figure 5.4 examples, two or more operators sharing a common precedence level appear. The question then arises, in case of precedence-ties, which operations are performed first? The "tie-breaking rule" is used to answer this question. The rule states that, within precedence level, operations are performed on a left-to-right basis. Hence, the value of V$FRITZ is determined this way.

(1) Multiply 4 by PR.

(2) Divide P10 by SC$FIXIT, retaining the integer portion of the quotient as the result.

(3) From the step (1) result, subtract the step (2) result.

(4) To the step (3) result, add 2.

(5) The step (4) result is the value of V$FRITZ.

Similarly, the value of V$USCHI is found this way.

(1) Divide C1 by 100, retaining the integer portion of the quotient as the result.

(2) Multiply the step (1) result by 100.

(3) The step (2) result is the value of V$USCHI.

Note that the value of V$USCHI equals C1 only if C1 is an integral multiple of 100. For example, if C1 is 400, then 400/100*100 equals 400. In like fashion, if C1 is 2700, then 2700/100*100 equals 2700. On the other hand, if C1 is 425, then 425/100*100 equals 400, not 425. In particular, 425/100 equals 4, not 4.25.

5.2.5 Real Variables vs. Integer Variables

The Variables discussed so far in this section are sometimes termed Integer Variables, because their values are computed by performing integer arithmetic on integer values. It is also possible to define Real Variables (i.e., floating-point Variables) in GPSS. Definitionally, Real Variables differ from Integer Variables only in that the word FVARIABLE, rather than VARIABLE, is entered in the Operation Field on the Variable Definition Card. Computationally, the GPSS Processor determines the value of a Real Variable this way.

(1) The entire value of any Functions used to indirectly specify data is used in the computation. The fractional portion, if any, is not discarded.

(2) Use of the division operator, /, does not result in the fractional portion of the quotient, if any, being discarded.

(3) Real (i.e., floating-point) arithmetic is used in the step-by-step evaluation of the Variable-defining expression. This simply means that any fractional values which arise during intermediate parts of the computation are retained and carried throughout the entire computation. Only after the expression's final value has been determined is the fractional part eliminated, with the integer portion being retained as the value of the Real Variable.

The modulus division operator, @, cannot be used in defining Real Variables. Furthermore, constants used in the defining expressions are still restricted to being integers. No decimal points can be used in the defining expressions. Also, the only Standard Numerical Attributes whose true values are used in evaluating Real Variables are Functions; for example, only the integer portion of Table means (TBj, or TB$sn) and Table standard deviations (TDj, or TD$sn) are carried in the computations leading to the value of a Real Variable. Finally, it should be noted, per (3) above, that Real Variables are integer-valued.

Despite the restrictions just stated, Real Variables have useful applications in GPSS modeling. Figure 5.5 shows two such applications. In the first example, the value

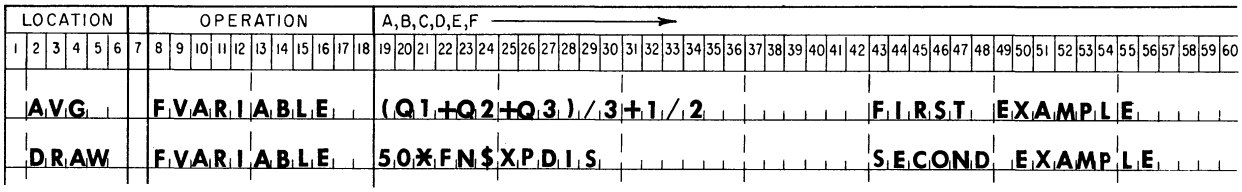


Figure 5.5 Two Examples of Real Variables

of V$AVG is formed by summing the contents of Queues 1, 2, and 3, then dividing by 3, then adding one half to this intermediate result. As a consequence, the value of the Real Variable AVG is the integer portion of the rounded average content of Queues 1 through 3. The rounding occurs because 0.5 (i.e., 1/2) is added to the true average content before the fractional portion is discarded to produce the final result. For example, suppose that Q1, Q2, and Q3 are 5, 7, and 2, respectively. Then their sum is 14. This sum, divided by 3, is 4.66666+. After 1/2 is added to this intermediate result, the true value of the expression is 5.16666+. The integer portion, 5, is then the value of V$AVG. If AVG were defined as an Integer Variable, its value in this

example would be 4. First of all, the sum 14, divided by 3, would be 4. Secondly, the value of 1/2 would be 0. The final result, then, would be 4 + 0, or 4. In fact, in Integer Variables, the term 1/2 is the defining expression can never contribute anything to the final value of the Variable.

Now consider the second Figure 5.5 example. Assume that FN$XPDIS is the usual 24-point continuous Function with which to sample from the exponential distribution with expected value 1. When a random sample from this distribution is multiplied by 50, the result is a random sample from the exponential population with mean 50. This Variable suggests an alternative, then, for simulating a Poisson arrival process at a GENERATE Block, or an exponential server at an ADVANCE Block. For the Function DRAW defined in Figure 5.5, the Blocks "GENERATE 50,FN$XPDIS" and "ADVANCE V$DRAW" are fully equivalent. Similarly, the Blocks "ADVANCE 50, FN$XPDIS" and "ADVANCE V$DRAW" are equivalent.

It is instructive to consider what would happen if the Function DRAW in Figure 5.5 were defined as an Integer Variable. Only the integer portion of FN$XPDIS would be multiplied by 50. Examination of the standard 24-point Function XPDIS reveals that about 65% of the time, its integer portion is 0. This means that V$DRAW would have a value of 0 about 65% of the time. These values could hardly be viewed as representative samples from an exponential population with mean 50.

As suggested by the examples, Real Variables and Integer Variables are referenced the same way, i.e., both types are referred to as Vj, or V$sn. For this reason, they cannot have identical names in a given model. If an Integer Variable is named ZELDA, there cannot be a Real Variable in the same model which is also named ZELDA, and so on.

In the next section, the GPSS method for sampling from the normal distribution is discussed. A highly useful application of Real Variables is involved in such sampling.

5.3 Sampling from the Normal Distribution

In Section 3.13, the important Poisson and exponential random variables were introduced, their relationship to each other was indicated, and a method for simulating Poisson arrivals and exponential servers in GPSS models was developed. A similar development will now be carried out for another highly important, well-known random variable, the normal random variable.

A normal random variable is completely described by stipulating two of its properties: its mean, and its standard deviation.[5a] By definition, the standard normal random variable has a mean of 0, and a standard deviation of 1. A non-standard, or general normal random variable is one whose mean does not equal 0, and/or whose standard deviation does not equal 1. The normal random variables of interest usually are non-standard.

To sample from a non-standard normal population, it is convenient first to sample from a standard normal population, then do arithmetic on the resulting value, thereby converting it to the non-standard equivalent. The relationship between the standard sample, and the non-standard equivalent, is shown in Equation 5.1, where $SNORM_{sample}$ is

$$GNORM_{sample} = (GNORM_{stdev})(SNORM_{sample}) + GNORM_{avg}$$

Equation 5.1 Equation for Converting a Standard Normal Sample to a Non-standard Equivalent

[5a] In contrast, the Poisson and exponential random variables are completely described by specifying only a single property, the mean.

the value drawn from the standard normal population, and $GNORM_{avg}$ and $GNORM_{stdev}$ are the average (i.e., mean) and standard deviation of the general normal random variable of interest. If a method can be found for randomly determining a value of $SNORM_{sample}$, it is a simple matter to define the right-hand side of Equation 5.1 as a GPSS Arithmetic Variable. Such a Variable can then be evaluated each time another sample is required from the corresponding normal population.

As shown in any post-calculus statistics textbook, a random sample can be drawn from the standard normal population by solving Equation 5.2 for $SNORM_{sample}$, which appears in the equation as the upper limit of an integral. Note that RNj, on the left-

$$RNj = \int_{-\infty}^{GNORM_{sample}} \frac{e^{-x^2/2}}{\sqrt{2\pi}} dx$$

Equation 5.2 Equation for Computing the Value of a Random Sample from the Standard Normal Population

hand side of Equation 5.2, is a sample drawn at random from any one of the eight 0-1 uniform random number populations in GPSS.

Equation 5.2 cannot be manipulated analytically to solve for $SNORM_{sample}$ explicitly in terms of RNj. As a result, the values of $SNORM_{sample}$ determined from Equation 5.2 for various values of RNj have been tabulated.[5b] Figure 5.6 displays the resulting relationship between RNj and $SNORM_{sample}$ in graphical form. The Figure 5.6 curve can be approximated with a continuous GPSS Function constructed by fitting a series of straight-line segments to it. An approximation suitable for this purpose has been developed by IBM. The approximation consists of a series of 24 line segments. Figure 5.7 shows the definition of the continuous GPSS Function composed of the 25 pairs of points used for the approximation.[5c] The Function has been symbolically named SNORM. RN1 has been arbitrarily chosen as the Function's argument. Note that, whereas the values of $SNORM_{sample}$ satisfying Equation 5.2 can range from minus infinity (for RNj values arbitrarily close to 0) to plus infinity (for RNj values arbitrarily close to 1), the values taken on by FN$SNORM defined in Figure 5.7 can only vary from -5 to +5. In view of the other inaccuracies in simulation modeling, the small discrepancies which arise in using the Figure 5.7 Function to approximate the Equation 5.2 relationship are acceptable in most cases.

Suppose now that the inter-arrival time of orders at a warehouse is normally distributed, with a mean value of 7 hours, and a standard deviation of 2 hours. If a Transaction represents an order, and the implicit time unit in a model is 1 hour, Figure 5.8(a) shows a GENERATE Block which simulates the order-arrival process. Figure 5.8(b) shows the Function GNORM which is used as the A Operand at the GENERATE

(5b) There are several ways to develop such a tabulation. For example, a value can be assumed for RNj. Then, by using a numerical integration technique and following a trial-and-error method, the value of $SNORM_{sample}$ for which the right-hand side of Equation 5.2 equals the assumed value of RNj can be determined.

(5c) The values in Figure 5.7 appear in the out-of-print IBM publication General Purpose Systems Simulator II, Form Number H20-6346.

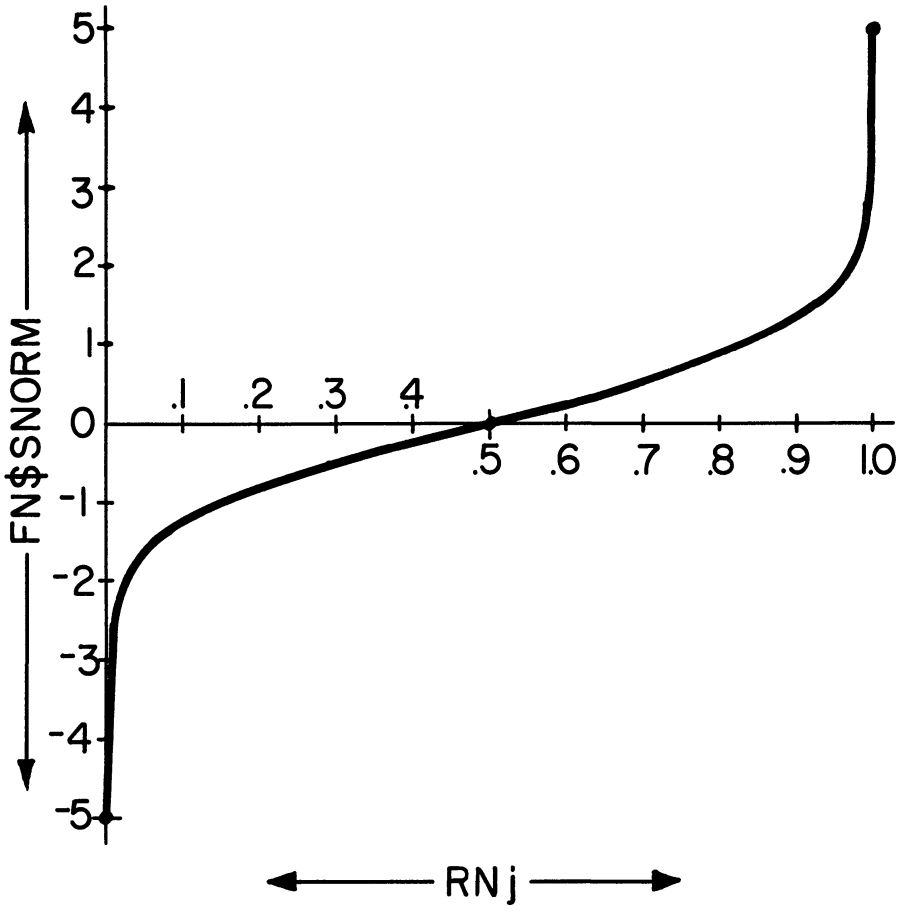
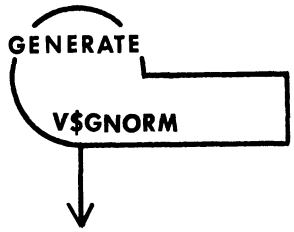


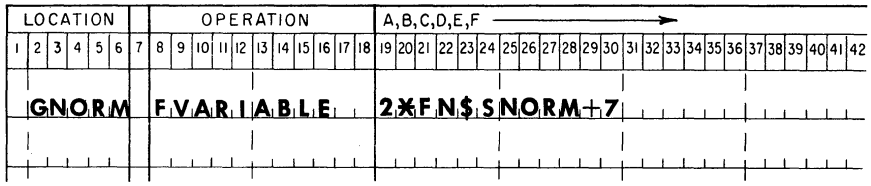
Figure 5.6 Graph of the Function Used to Sample from the Standard Normal Distribution

```
LOCATION          OPERATION         A,B,C,D,E,F ------>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
SNORM             FUNCTION          RN1,C25
0,-5/.00003,-4/.00135,-3/.00621,-2.5/.02275,-2
.06681,-1.5/.115507,-1.2/.158666,-1/.211886,-.8/.274425,-.6
.344578,-.4/.420074,-.2/.500000,0/.579926,.2/.655422,.4
.725747,.6/.788145,.8/.841344,1/.884493,1.2/.933193,1.5
.977249,2/.993790,2.5/.998650,3/.999968,4/1,5
```

Figure 5.7 Continuous 25-Point GPSS Function Used to Sample from the Standard Normal Distribution



(a) A GENERATE Block Which Simulates Normally-Distributed Inter-arrival Times



(b) Definition of the Function GNORM Used at the GENERATE Block

Figure 5.8 An Example for Simulating Normally-Distributed Inter-arrival Times in GPSS

Block. It is assumed that Figure 5.7 defines the Function SNORM. Consistent with Equation 5.1, the value of the Function GNORM is determined by drawing a sample from the standard normal population defined by the Function SNORM, multiplying this value by 2 (standard deviation of the non-standard population), and adding 7 (mean of the non-standard population) to the product. Each time a Transaction moves out of the Figure 5.8(a) GENERATE Block, the Processor randomly determines the value of its successor's inter-arrival time, then, in a manner faithful to Equations 5.1 and 5.2.

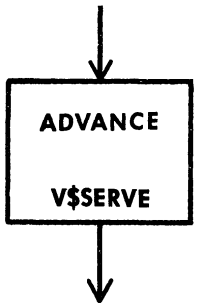
As another example, assume that the service time at a certain point in a model is normally distributed, and that the mean and standard deviation in effect depend on the type of service being provided. Table 5.4 shows the applicable means and standard

Type of Service	Properties of Normally Distributed Service Times, Minutes	
	Mean	Standard Deviation
1	30	5
2	20	3
3	36	6

Table 5.4 Properties of Normally-Distributed Service Times

deviations for the three different types of service that can be demanded. Suppose that a Transaction has its service-type recorded in Parameter 4. Figure 5.9(a) shows an ADVANCE Block at which the service time is simulated. Figure 5.9(b) shows definitions of the Functions and the Arithmetic Variable which support the operation of the Figure 5.9(a) Block. The relationship in Equation 5.2 is expressed through the Variable symbolically names SERVE in Figure 5.9(b). That Variable, in turn, calls on the Functions MEAN and STDEV to supply the correct mean and standard deviation each time a Transaction moves into the ADVANCE Block. The Functions use the P4 value of the active Transaction as their argument. List Functions have been employed, because the

sequence of feasible Function values consists of consecutive integers beginning with 1.



(A) An ADVANCE Block Which Simulates Normally-Distributed Service Times

LOCATION	OPERATION	A,B,C,D,E,F →
MEAN	FUNCTION	P4,L3
1,30/2,20/3,36		
STDEV	FUNCTION	P4,L3
1,5/2,3/3,6		
SERVE	FVARIABLE	FN$STDEV*FN$SNORM+FN$AVG

(b) Definition of Functions and the Arithmetic Variable Used at the ADVANCE Block

Figure 5.9 An Example for Simulating Normally-Distributed Service Times in GPSS

In conclusion, three features of sampling from normal populations in GPSS should be noted.

(1) The mean and standard deviation characterizing the normal population of interest are specified either directly or indirectly in a Variable definition. More than one such Variable may have to be defined, depending on the number of normal populations from which samples must be drawn in a model, and the context in which the corresponding samples are used.

(2) The word FVARIABLE must be used in the Operation Field of the Variable Definition Card when defining a Variable with which to sample from a normal distribution. Use of the word VARIABLE would mean that only the integer portion of the values drawn from the standard normal population would be carried in the subsequent computation. This, in turn, would result in unrealistic distortions in the computed non-standard values.

(3) The Function SNORM defined in Figure 5.7 can return values as small as -5. If the mean of a given normal distribution is less than 5 standard deviations above zero, Equation 5.1 shows that samples drawn from the population can consequently be negative. This is consistent with the fact that, in theory, normally-distributed values can range from minus infinity to plus infinity. In practice, however, it may be that the quantities which are postulated as being normally distributed have meaning only when they are non-negative. For example, negative inter-arrival times and negative service times are meaningless. In such contexts, then, the analyst must either (a) insure that the mean is at least 5 standard deviations above zero, or (b) test sampled values for negativity and arbitrarily set these values to zero when they occur, or discard them and repeat the sampling process. The TEST Block, which will not be introduced until Section 5.8, could be used to implement the possibilities in (b).

5.4 Exercises

5.4.1 Assuming the current value of W$RUTE6 is 7, what will be the value of VAR1, VAR2, VAR3, and VAR4 as defined in Table T5.4.1?

Location	Operation	Operands
VAR1	VARIABLE	W$RUTE6/5
VAR2	VARIABLE	W$RUTE6@5
VAR3	VARIABLE	25/W$RUTE6
VAR4	VARIABLE	W$RUTE6/25

Table T5.4.1

5.4.2 How do GPSS Integer and Real Variables differ from each other? In what ways are they similar to each other?

5.4.3 Show the definition of Variables which will take on these values.

(a) The sum of the capacities of the Storages TUGS and SLIPS.

(b) The rounded average content of the Queues 5, 7, and LONG.

(c) A sample drawn at random from the exponential population with a mean value of 75.

(d) A sample drawn at random from the normal population with a mean of 10, and a standard deviation of 2.

(e) An integer drawn at random from the population of integers uniformly distributed between 0 and 15, inclusive.

(f) An integer drawn at random from the population of integers uniformly distributed between 1 and 15, inclusive.

(g) An integer drawn at random from the population of integers uniformly distributed between 7 and 18, inclusive.

(h) An integer drawn at random from the population of integers uniformly distributed between 1 and P2, inclusive. (Which Transaction will supply the value of P2?)

5.4.4 Define a Variable which returns a value as a function of PR according to the pattern shown in Table T5.4.4.

PR	Value of Variable
1	1
2 or more	PR-1

Table T5.4.4

(Which Transaction will supply the value of PR?)

5.4.5 Depending on whether its Parameter 7 value is 1, 2, 3, or 4, the holding time of a Transaction at an ADVANCE Block is to be determined by sampling from normal distributions with means and standard deviations of (20,3), (35,4), (30,4), and (25,2), respectively. Show an ADVANCE Block at which the holding time can be simulated. Also show the definitions of Variables and/or Functions needed to support operation of the ADVANCE Block.

5.4.6 (a) An analyst defines the Real Variable ALPHA with the expression (P2+P5)*0.4*R$TUG. This expression will result in an error message. Why?

(b) Show an acceptable alternative expression which produces the effect the analyst desired in (a).

(c) In an ADVANCE Block which uses a spread modifier, show how to make the spread 75% of the mean holding time. Assume that the mean is available as the value of Parameter 1 of Transactions which enter the ADVANCE Block.

5.4.7 The Real Variable BETA is defined with the expression 9*FN$SNORM+2, where the Function SNORM is defined in Figure 5.7. Why will an error eventually result if the Block "ADVANCE V$BETA" is included in a model? Refer to Appendix C and determine exactly which error message will result.

5.4.8 (a) The j-th Transaction arriving at Point A in a model is to have j assigned as the value of its 5th Parameter, for j = 1, 2, 3, ... Show how to accomplish this. (Caution: The "Block entry count" Standard Numerical Attribute is updated only as the last step in execution of a Block subroutine. For example, the first Transaction to enter the Block "HERE ASSIGN 3,N$HERE" has zero assigned as the value of Parameter 3 because, at the time of its evaluation, a value of zero is still recorded for N$HERE. The second Transaction to enter the same Block has one assigned as the value of Parameter 3, and the third Transaction to enter the Block has two assigned as the value of Parameter 3, and so on.

(b) The jth, j+10th, j+20th, etc. Transaction arriving at Point A in a model is to have j assigned as the value of its 5th Parameter, for j = 1,2,3,..., 8,9, and 10. Show how to accomplish this.

5.4.9 Show how a single GENERATE Block can be used to bring Transactions into a model according to the specifications in Problem P2.34.3(a). (Hint: the sequence of inter-arrival times is 1,3,6,4,11,3,6,4,11,3,6,4,11,... Construct a Function which returns the values 3,6,4,11 cyclically.)

5.5 Savevalues: The INITIAL Card and the SAVEVALUE Block

Section 5.1 listed the "inability to compute" and the "inability to construct tests" as GPSS modeling limitations to be overcome in this chapter. The computational facility within the language has now been described. Before discussing tests, however, it is appropriate to identify several other limiting features of the models presented so far, and indicate how those limitations can be handled.

A list of the modeling restrictions implicit in the presentation until now is shown below.

(1) The only way to incorporate external data into models has been by specifying it directly, as Block Operands, or by providing it as ordered pairs of points in Function definition.

(2) A restriction not previously mentioned is that constants used as Block Operands cannot take up more than six columns on a punchcard. The largest possible value of a directly-specified Block Operand, then, is 999999.

(3) Transactions cannot directly "talk to each other", or "talk about each other". A Transaction which has just entered an ASSIGN Block, for instance, cannot have the value of some other Transaction's P5 copied into one of its Parameters. Or, at the SELECT Block in MIN mode, a Transaction cannot ask, "which one of the Transactions waiting at the Block in Location BLOK 9 has the smallest P1 value?"

(4) There is no direct way to have the values of Variables printed out. Their values at the end of a simulation run are not computed and outputted by the Processor; furthermore, the values of selected Variables cannot be printed out during a run by use of the PRINT Block.

Any problems these restrictions might pose can be resolved by use of a series of "permanent" memory locations whose initial values can be assigned before a simulation begins, and to and from which values can be transferred from anywhere in a model, as a simulation proceeds. These memory locations are termed Savevalues. They are Standard Numerical Attributes in GPSS. Unlike a Transaction's Parameter set, Priority Level, and Mark Time, which are lost when a Transaction exits a model, the Savevalues endure throughout a simulation. And, unlike such other Standard Numerical Attributes as FRj, QCj, Sj, etc., their values are not automatically updated by the Processor. In contrast, their values are changed only under direct control of the analyst.

Pertinent properties of Savevalues will now be discussed. Then the method of pre-setting their values before a simulation starts, and modifying their values as a simulation proceeds, will be described. Finally, the effect of RESET and CLEAR cards on Savevalues will be indicated.

5.5.1 Properties of Savevalues

The values of Savevalues are signed integers.[5d] The maximum magnitude of a Savevalue depends whether it is of the halfword or fullword type. Identical to the situation with Parameters, the maximum magnitude of halfword and fullword Savevalues is 32,767 [i.e., $2^{15}-1$], and 2,147,483,647 [i.e., $2^{31}-1$], respectively.

The names of Savevalues are XHj (or XH$sn) and Xj (or X$sn), for halfword and fullword Savevalues, respectively. As usual, the "j form" is used if a Savevalue is being referred to numerically; and the "sn form" is used if it is being referenced by symbolic name.

The quantity of Savevalues normally available depends on the amount of computer memory available. With 64K bytes of memory, for example, there are normally 50 halfword and 100 fullword Savevalues available to the analyst. In this case, the halfword

---

[5d] In GPSS V, real-valued Savevalues are also available.

Savevalues would be numbered from 1 to 50, and the fullword Savevalues would be numbered from 1 to 100.

As Standard Numerical Attributes, Savevalues can be used to indirectly provide data as Block Operands, and as Function and Table arguments. For example, the Table Definition Card "SMITH TABLE XH3,-20,5,25" might be used in a model. Each time a Transaction moved into a TABULATE Block referencing the Table SMITH, the current value of halfword Savevalue 3 would be entered in the Table. Similarly, each time a Transaction moved into the Block "ADVANCE X3,XH$SPRED", the mean holding time would be determined by sampling from the uniform distribution X3±XH$SPRED. Fullword Savevalue 3 would be used as the mean of the distribution, and the halfword Savevalue SPRED would be used as the Spread Modifier.

At the end of a simulation, all non-zero Savevalues are automatically printed out. Selected Savevalues can also be outputted during a run by use of the PRINT Block. This is accomplished by providing X, or XH, as the Block's Field C mnemonic. For example, when a Transaction moves into the Block "PRINT 3,8,XH", halfword Savevalues 3 through 8 will be printed out.

5.5.2 The INITIAL Card

Normally, the GPSS Processor sets Savevalues to zero before a simulation begins. At the analyst's option, selected Savevalues can be initialized with non-zero values by use of the INITIAL Card. The card takes the form shown in Figure 5.10. As indicated,

Location	Operation	Operands
Not Used	INITIAL	$name_1,value_1/ \ldots /name_i,value_i/ \ldots /name_n,value_n$

Figure 5.10 Savevalue Initialization Card Format

the Location Field is not used. The word INITIAL is entered in the Operation Field. The "basic unit" of information in the Operands Field is of the form "$name_i,value_i$", where $name_i$ is the name of the i-th Savevalue being initialized, and $value_i$ is the initial value. Examples of "basic units" are "XH3,-450", "X$JOE,400000", and "XH49,6". Note that a "name" consists of the family name X or XH, followed by the number or symbolic name of a specific family member. The maximum magnitude of a "value" is 32,767 and 2,147,483,647 for halfword and fullword Savevalues, respectively.

Slashes (/) are used to separate the "basic units" on an INITIAL Card. No blanks are allowed within or between consecutive basic units. No information can be entered beyond card column 71. If there is not room enough on one card to initialize all the Savevalues of interest, more than one INITIAL Card must be used.

Figure 5.11 provides several examples of the Savevalue INITIAL Card. In the first

```
LOCATION  OPERATION  A,B,C,D,E,F ------>
          INITIAL    X$TIMER,1000000                    FIRST EXAMPLE
          INITIAL    X3,25/XH7,-10/XH4,452/X98,1        SECOND EXAMPLE
          INITIAL    X$BETA,2/X2,3/X$ALPHA,1/X4,7       THIRD EXAMPLE
          INITIAL    XH2,3/X7,5/X8,5/X9,5/X1,-66        FOURTH EXAMPLE
          INITIAL    XH2,3/X7-X9,5/X1,-66               FIFTH EXAMPLE
```

Figure 5.11 Examples of the Savevalue INITIAL Card

example, the fullword Savevalue TIMER is initialized at one million. If it were of interest to have a "timer Transaction" appear at time 1,000,000 and turn off a model, the Block "GENERATE X$TIMER" could be used for this purpose. Note that the Block "GENERATE 1000000" is not acceptable, because more than six card columns are required for the A Operand. If the Processor encountered this latter GENERATE card during the input phase, an error message TOO MANY DIGITS IN NUMERIC CONSTANT would be printed out and the simulation would not be performed.

If one or more Savevalues are referenced as A and/or B Operands in GENERATE Blocks, and their values are to be non-zero, the INITIAL Cards must appear in the card deck before the GENERATE Block images appear. The values will otherwise still be zero when the Processor encounters the GENERATE cards, and zeros will then be used in prescheduling the arrival time of the first Transaction at those GENERATE Blocks. It is recommended that all INITIAL Cards, as well as Function definitions, Storage capacity cards, and Table cards, be placed ahead of Block images in the model.

In the second Figure 5.11 example, fullword Savevalue 3 is initialized at 25; halfword Savevalue 7 at -10; halfword Savevalue 4 at 452; and fullword Savevalue 98 at 1. This example shows that there are no restrictions on the order in which the Savevalues appear on the INITIAL Card. It is not necessary to group the halfword or fullword Savevalues together; furthermore, it is not necessary that the numbers of the Savevalues involved be in ascending order.

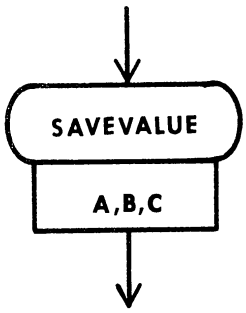
The third example involves a mixture of fullword Savevalues named symbolically and numerically. This raises the question, how is a correspondence established between symbolic Savevalue names, and their numeric equivalent? As might be expected, the same two-step process followed in this regard with respect to Facilities, Storages, Queues, Functions, and Tables is used. As Savevalues named numerically are encountered, their numbers are removed from the "still available" list. In their order of first appearance, the symbolic names are then made equivalent to increasingly larger values on the "still available" list. Assuming that no other Savevalues are referenced in the model in which the third example INITIAL Card appears, this means that BETA and ALPHA would be equivalent to fullword Savevalues 1 and 4, respectively.

The fourth example shows that fullword Savevalues 7, 8, and 9 are each to be 5 initially. When Savevalues of the same type are to have identical initial values, and the Savevalue numbers form a sequence of consecutive integers, the "basic unit" can be expressed as "$name_i$-$name_k$,value", where $name_i$ and $name_k$ are the names of the smallest and largest numbered Savevalues, respectively, and "value" is the common initial value. An alternative form for the fourth Figure 5.11 example, then, is shown in the fifth example.

5.5.3 The SAVEVALUE Block

The value of one Savevalue is modified when a Transaction moves into a SAVEVALUE Block. This Block, and its A, B, and C Operands, are shown in Figure 5.12. When a Transaction enters a SAVEVALUE Block, the B Operand data becomes the value of the Savevalue whose number, or symbolic name, is provided by the A Operand. Whenever a Transaction enters the SAVEVALUE Block shown in Figure 5.13, for example, the value of the Variable ALPHA is first computed. The result is then assigned to the fullword Savevalue whose number is in P5. The old value of the Savevalue is destroyed in the process.

A second example is shown in Figure 5.14. When a Transaction enters the SAVEVALUE Block, the standard deviation in the Table TIME3 is copied into the halfword Savevalue MARIE.



Operand	Significance	Default Value or Result
A	Number or symbolic name of the Savevalue to be modified	Error
B	Data to be used in the modification process	Error
C	Specifies whether the Savevalue involved is a halfword or fullword type; the characters H and F designate halfword and fullword types, respectively	H

Figure 5.12 The SAVEVALUE Block and Its A, B, and C Operands

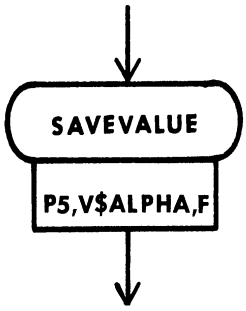


Figure 5.13 A First Example for the SAVEVALUE Block

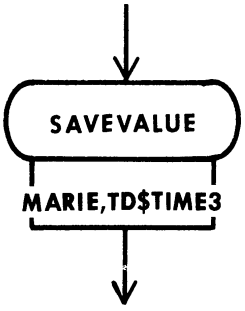


Figure 5.14 A Second Example for the SAVEVALUE Block

Like the ASSIGN Block, the SAVEVALUE Block can be used in increment mode and decrement mode, as well as in replacement mode. In increment mode, the previous value of the Savevalue is incremented by the B Operand data. In decrement mode, it is decremented by the B Operand data. Increment and decrement mode are specified by placing a plus (+) or minus (-) sign, respectively, ahead of the comma which separates the A and B Operands. For example, when a Transaction enters the Block "SAVEVALUE 5+,X2,F", the value of X5 will be increased by the amount X2. Or, when a Transaction enters the Block "SAVEVALUE DAVID-,FN$HOLD", the value of XH$DAVID will be decreased by the amount FN$HOLD.

5.5.4 The Effect of RESET and CLEAR Cards on Savevalues

The values in Savevalue locations are not changed by the RESET Card. Use of the CLEAR Card, however, normally causes all the halfword and fullword Savevalues to be set to zero. This often proves to be inconvenient, because it may mean that an entire series of INITIAL Cards must be repeated before the simulation can be started again. For this reason, there is an optional form of the CLEAR Card available with which the user can specify selected Savevalues which are to be left intact. To use this so-called Selective CLEAR Card, all the user must do is provide in the Operands Field the names of Savevalues not to be zeroed-out when the clearing operation occurs. In contrast to the flexibilities allowed with the INITIAL Card, however, the following restrictions apply to the Selective CLEAR Card.

(1) Halfword and fullword Savevalues cannot be inter-mixed on the card.

(2) Within each Savevalue type (i.e., halfword or fullword), the numbers of the Savevalues to be left intact must be in ascending order.

Two examples of the Selective CLEAR Card are shown in Figure 5.15. The first

```
LOCATION  OPERATION  A,B,C,D,E,F ------>
1 2 3 4 5 6  7 8 9 10 11 12 13 14 15 16 17 18  19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66
          CLEAR      X3,X12,X14,XH1-XH5,XH9                    FIRST EXAMPLE
          CLEAR      XH1-XH5,XH9,X3,X12,X14                    SECOND EXAMPLE
```

Figure 5.15 Examples for the Selective CLEAR Card

example specifies that fullword Savevalues 3, 12, and 14 and halfword Savevalues 1 through 5, and 9, are not to be set to zero when the CLEAR Card has its effect. The second example is identical to the first, except that the halfword Savevalues involved are specified first, and then the fullword Savevalues are specified.

5.6 Case Study 5A: A Third Tour Through Case Study 2D

(1) Statement of the Problem

Case Study 2D involved modeling a widget-manufacturing operation to determine how many assemblers should be hired to maximize profits. In the solution presented, hand computations based on model output were performed to determine the profit realized for each "number of assemblers" alternative investigated. Modify the Case Study 2D solution to incorporate these features.

(1) Arrange to have the average daily profit realized for each alternative be included in the model output.

(2) Make the number of simulated days in effect for each simulation be a "variable" in the sense that it is specified through a Savevalue.

(3) Investigate alternatives of 4, 5, and 6 assemblers in a single run. For each configuration, simulate for the equivalent of a 40-hour work week. Assume there are no discontinuities within a working day, or in moving between consecutive 8-hour work days.

For convenience, the description of the problem in Case Study 2D is repeated below.

The manufacture of a certain line of widgets involves a relatively lengthy assembly process, followed by a short firing time in an oven. Since ovens are expensive to maintain, several assemblers share a single oven, which holds only one widget at a time. An assembler cannot begin assembling a new widget until he has removed the old one from the oven.

This is the pattern followed, then, by each assembler:

(1) Assemble next widget.

(2) Wait, first-come, first-served, to use the oven.

(3) Use the oven.

(4) Return to (1).

Time and cost studies have been conducted to provide the information in Tables 5A.1 and 5A.2, respectively.

Operation	Time Required, Minutes
Assemble	30±5
Fire	8±2

Table 5A.1 Operation Times for Case Study 5A

Item	Cost Information
Assembler's Salary	$3.75 per hour
Oven Cost	$80 per 8-hour work day (independent of utilization)
Raw Material	$2 per widget
Value of Finished Widgets	$7 per widget

Table 5A.2 Financial Data for Case Study 5A

(2) Approach Taken in Building the Model

In Case Study 2D, a Transaction was used to simulate an assembler. The proper number of assemblers was introduced into the model through use of the GENERATE Block Limit Count (D Operand). This value, in turn, was directly specified. In the approach taken here, the Limit Count is indirectly specified through the fullword Savevalue GUYS. This makes it possible to re-configure the model simply by re-defining the value of GUYS between consecutive runs. The Savevalue GUYS can then also be used for the fixed-cost computations when determining the average daily profit. [Note that it would also be possible to build the model requested here, even while continuing to directly specify the Limit Count. See Problem 5.7.2(c).]

The fullword Savevalue named TIMER is used as the A Operand for the GENERATE Block through which the timer Transaction enters the model. The default value of zero is in effect for the B Operand. The simulated time at which the timer enters the model, then, equals the value of X$TIMER. As explained below, the value of X$TIMER is also used in a computational role in the model, to aid in determining the average daily profit.

The average daily profit is $5 times the average number of widgets produced each day, minus the fixed daily costs. If the symbolic Location Name MADE is given to the Block "RELEASE OVEN", then N$MADE is the number of widgets produced during a given simulation. If it is assumed that the implicit time unit is 1 minute, and that X$TIMER is an integral multiple of 480, then X$TIMER/480 is the number of days simulated. (Problem 5.7.2(e) involves relaxing the assumption that X$TIMER is an integral multiple of 480.) Now suppose that the expression X$TIMER/480 is used to define a Variable named DAYS. Then 5*N$MADE/V$DAYS is the average daily profit before deduction of oven cost and salaries. Daily oven cost, from the statement of the problem, is $80. At $3.75 per hour, each assembler earns $30 per 8-hour day. The total daily salary, then, is 30*X$GUYS. Average daily profit is consequently the value of the expression 5*N$MADE/V$DAYS-80-30*X$GUYS. Assume this expression is used to define the Variable named PROFT (for profit). Then, when the timer Transaction enters the model, it can move through a SAVEVALUE Block, causing the average daily profit to be computed and

copied into a Savevalue. When the model shuts off, this profit value will be included in the listing of Savevalues shown as part of the standard model output.

(3) Table of Definitions

Time Unit: 1 Minute

GPSS Entity	Interpretation
Transactions	
Model Segment 1	Assemblers
Model Segment 2	The timer
Facilities	
OVEN	The oven
Savevalues	
GUYS	The number of assemblers
TIMER	Duration of the simulation for each alternative investigated, minutes
Variables	
DAYS	Duration of the simulation for each alternative investigated, 8-hour days
PROFT	Average daily profit for the alternative being investigated, dollars

Table 5A.3 Table of Definitions for Case Study 5A

(5) Extended Program Listing

```
BLOCK                                                                    CARD
NUMBER  *LOC    OPERATION  A,B,C,D,E,F,G            COMMENTS             NUMBER
                SIMULATE                                                    1
        *                                                                   2
        *       INITIALIZE SAVEVALUES                                       3
        *                                                                   4
                INITIAL    X$GUYS,4/X$TIMER,2400                            5
        *                                                                   6
        *       DEFINE VARIABLES                                            7
         DAYS   VARIABLE   X$TIMER/480                                      8
         PROFT  VARIABLE   5*N$MADE/V$DAYS-80-30*X$GUYS                     9
        *                                                                  10
        *       MODEL SEGMENT 1                                            11
        *                                                                  12
1               GENERATE   ,,,X$GUYS       PROVIDE ASSEMBLERS              13
2        BACK   ADVANCE    30,5            ASSEMBLE NEXT WIDGET            14
3               SEIZE      OVEN            CAPTURE THE OVEN                15
4               ADVANCE    8,2             USE THE OVEN                    16
5        MADE   RELEASE    OVEN            FREE THE OVEN                   17
6               TRANSFER   ,BACK           GO DO THE NEXT ASSEMBLY         18
        *                                                                  19
        *       MODEL SEGMENT 2                                            20
        *                                                                  21
7               GENERATE   X$TIMER         TIMER ARRIVES                   22
8               SAVEVALUE  INDEX,V$PROFT   SET X$INDEX = AVERAGE DAILY PROFIT 23
9               TERMINATE  1               SHUT OFF THE RUN                24
        *                                                                  25
        *       CONTROL CARDS                                              26
        *                                                                  27
                START      1               START THE 1ST RUN (4 ASSEMBLERS) 28
                INITIAL    X$GUYS,5        RE-CONFIGURE FOR 2ND RUN        29
                CLEAR      X$GUYS,X$TIMER  SELECTIVELY CLEAR FOR 2ND RUN   30
                START      1               START THE 2ND RUN (5 ASSEMBLERS) 31
                INITIAL    X$GUYS,6        RE-CONFIGURE FOR 3RD RUN        32
                CLEAR      X$GUYS,X$TIMER  SELECTIVELY CLEAR FOR 3RD RUN   33
                START      1               START THE 3RD RUN (6 ASSEMBLERS) 34
                END        RETURN CONTROL TO OPERATING SYSTEM              35
```

Figure 5A.2 Extended Program Listing for Case Study 5A

(4) Block Diagram

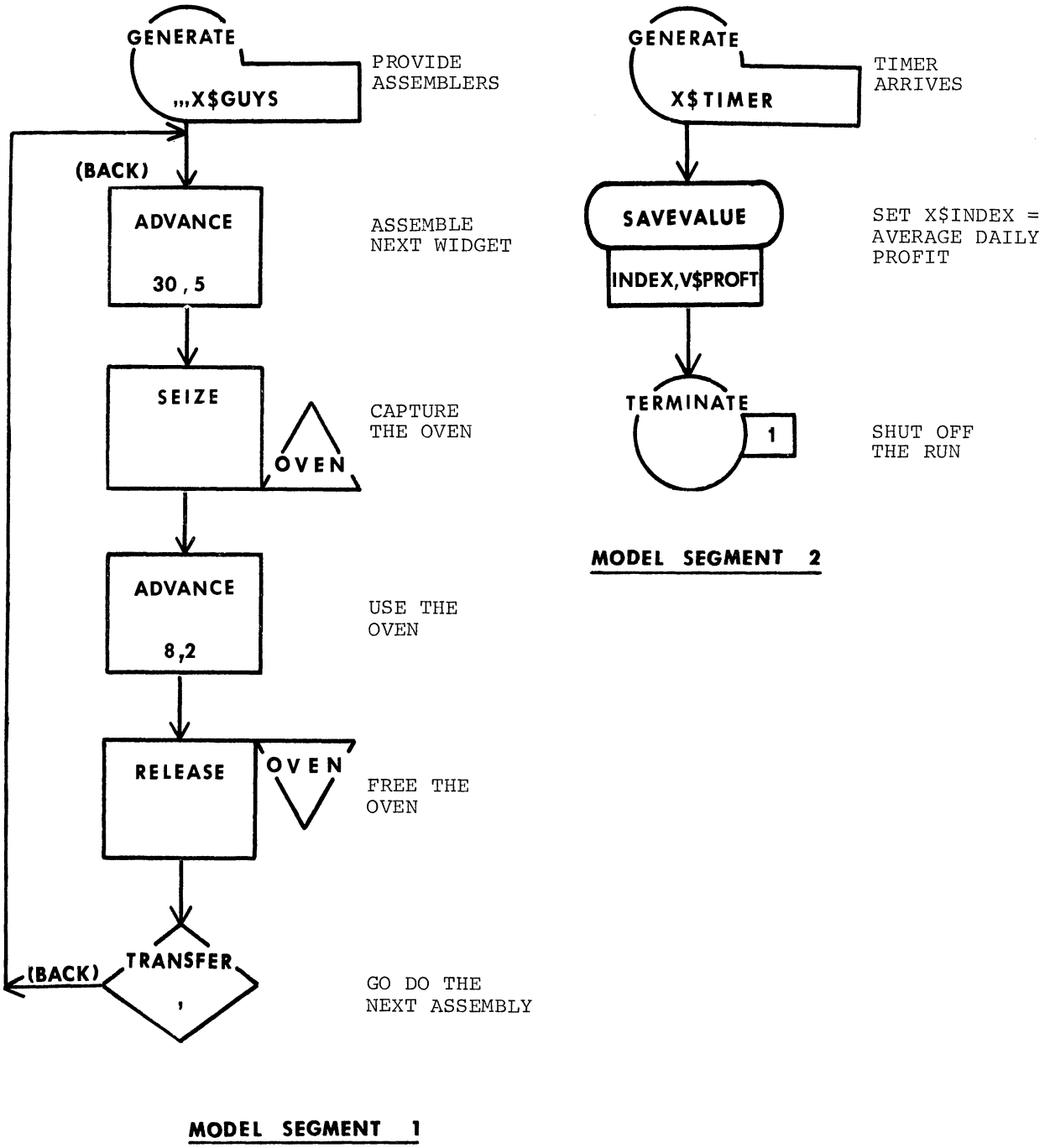


Figure 5A.1 Block Diagram for Case Study 5A

(6) Program Output

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
OVEN	.785	236	7.991	5	

CONTENTS OF FULLWORD SAVEVALUES (NON-ZERO)

SAVEVALUE NR,	VALUE	NR,	VALUE	NR,	VALUE	NR,	VALUE
GUYS	4	TIMER	2400	INDEX	35		

(a) 4-Assembler Configuration

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
OVEN	.942	281	8.053	6	

CONTENTS OF FULLWORD SAVEVALUES (NON-ZERO)

SAVEVALUE NR,	VALUE	NR,	VALUE	NR,	VALUE	NR,	VALUI
GUYS	5	TIMER	2400	INDEX	50		

(b) 5-Assembler Configuration

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRAN	SEIZING TRANS. NO.	PREEMPTING TRANS. NO.
OVEN	.989	296	8.020	1	

CONTENTS OF FULLWORD SAVEVALUES (NON-ZERO)

SAVEVALUE NR,	VALUE	NR,	VALUE	NR,	VALUE	NR,	VALUI
GUYS	6	TIMER	2400	INDEX	35		

(c) 6-Assembler Configuration

Figure 5A.3 Facility and Savevalue Information Outputted by the Figure 5A.2 Model

(7) Discussion

Model Implementation

In the extended program listing in Figure 5A.2, Card 2 is the INITIAL Card used to supply the fullword Savevalues GUYS and TIMER with values of 4, and 2400, respectively. This is consistent with first simulating with a 4-assembler configuration, and having the timer Transaction enter the model after 40 hours (i.e., 2400 minutes) of simulated time have elapsed.

Cards 4 and 5 in Figure 5A.2 define the Variables DAYS and PROFT, respectively, as discussed earlier. Note that the Processor must evaluate the Variable DAYS as one of the steps to be performed in evaluating the Variable PROFT.

When the timer Transaction comes into the model, it enters a SAVEVALUE Block at which the value of the Variable PROFT is computed and then copied to the Savevalue named INDEX (Card 20). The timer Transaction then moves into a TERMINATE Block (Card 21), shutting off the model.

After the first START Card (Card 22, Figure 5A.2), there is an INITIAL Card which re-defines the value of the Savevalue GUYS as 5. This accomplishes re-configuring the model for the 5-assembler alternative. The following card (Card 23) is then a Selective CLEAR, which sets all values back to zero (with the exception of the Savevalues GUYS and TIMER), returns Transactions to the latent pool, and then pre-schedules Transaction arrivals at the two GENERATE Blocks in the model. The START Card that appears next (Card 24) causes the simulation to begin again for the newly-configured model. A similar INITIAL-CLEAR-START sequence is then provided (Cards 25, 26, and 27) for the 6-assembler configuration.

Program Output[5e]

Facility-statistics and Savevalue values outputted by the Figure 5A.2 model are shown in Figure 5A.3. The Facility statistics for all three configurations exactly match the results produced in Section 2.33 when the CLEAR Card was first introduced, then used for this same problem to study alternatives of 4, 5, and 6 assemblers sequentially (see Figures 2.37 and 2.38).

Because the Savevalues GUYS, TIMER, and INDEX were encountered by the Processor in that order in the Figure 5A.2 model, their values appear in that order in Figure 5A.3. In the Savevalue information in Figure 5A.3(a), for example, under the columns labeled NR (short for "NUMBER"), appear GUYS, TIMER, and INDEX. At the right of each NR column, in a column labeled VALUE, is entered the value of the corresponding Savevalue. Hence, GUYS has a value of 4; TIMER has a value of 2400; and INDEX has a value of 35 in Figure 5A.3(a). This means that, for the case of 4 assemblers, and based on a simulation of 2400 minutes, the average daily profit was $35. Parts (b) and (c) in Figure 5A.3 indicate average daily profits of $50 and $35 for the cases of 5 and 6 assemblers, respectively.

[5e] Total CPU time for the simulation was 12.5 seconds.

5.7 Exercises

5.7.1 This exercise consits of short problems for Savevalues.

(a) Compare and contrast Savevalues with Parameters.

(b) What is the difference between X1 and XH1?

(c) Show an INITIAL Card which initializes fullword Savevalues 3, 5, and MACRO with values of -25, 75, and 40, respectively, and which initializes halfword Savevalues 1 through 5 with the value 100.

(d) Show a CLEAR Card which does not alter the values of halfword Savevalues 1 and 5, and fullword Savevalues 4, 5, and 6, but causes all other Savevalues to be set to zero.

(e) Why is the Block "ADVANCE 4500000" in error? How could you avoid this error, yet accomplish the same effect?

5.7.2 In a certain model, the Block "GENERATE X$MEAN,X$SPRED" is used. Two alternative configurations are being studied sequentially, with the understanding that the Savevalues MEAN and SPRED are to have different values for the second configuration than they do for the first. Explain the differences among the three control card sequences shown in (a), (b), and (c) below. Which sequence is correct, and why? (Suggestion: you might want to review the operation of the CLEAR Card as explained in Section 2.33.)

(a)
```
START       1
CLEAR
INITIAL     X$MEAN,60/X$SPRED,30
START       1
```

(b)
```
START       1
CLEAR       X$MEAN,X$SPRED
INITIAL     X$MEAN,60/X$SPRED,30
START       1
```

(c)
```
START       1
INITIAL     X$MEAN,60/X$SPRED,30
CLEAR       X$MEAN,X$SPRED
START       1
```

5.7.3 These exercises all pertain to Case Study 5A.

(a) Use the Facility statistics in Figure 5A.3 to compute (by hand) the average daily profit for the cases of 4, 5, and 6 assemblers, respectively. Do your results match those appearing in the Savevalue information in Figure 5A.3?

(b) Explain how the Variable DAYS can be eliminated in the Figure 5A.2 model.

(c) Explain how to build a model for Case Study 5A which uses only the Savevalue INDEX, and does not use any other Savevalues. Show all the details involved by presenting the complete model on a coding sheet.

(d) Change the Figure 5A.2 model so that each estimate of average daily profit is based on a simulation of 10 working days, not 5. Make a similar change for the model you built as a solution to (c) above. How many cards had to be changed in each case? Using "number of cards changed" as a measure, which model seems to be the more flexible?

(e) The Figure 5A.2 model assumes that the value of X$TIMER is an integral multiple of 480. Show how to modify the model to eliminate this assumption.

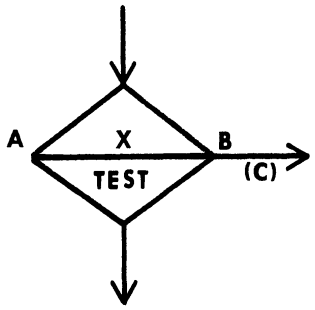
The modified model should work, for example, when the value of X$TIMER is 2450. In this case, the average daily profit computation would be based on a simulated time span of 5 8-hour work days, and the first 50 minutes of the 6th work day.

(f) In Case Study 5A, daily profit is a random variable. Show how to modify the Figure 5A.2 model so that the value of this random variable is entered in a GPSS Table on a day-by-day basis. The modified model should simulate with each configuration for 25 consecutive working days. Included in the output, then, will not only be the average value of the "daily profit" random variable, but an estimate of its distribution based on a sample of size 25. Clearly state all assumptions you make in modifying the model.

5.7.4 Case Study 2E involves searching for a system configuration which minimizes the average daily cost associated with a production process. The extended program listing for a model which investigates 9 alternative system configurations sequentially appears in Figure 2E.3. Show how to modify the Figure 2E.3 model so that the average daily cost estimate for each configuration is included as part of the standard model output.

5.8 Testing Numeric Relationships: The TEST Block

The relation between the values of two Standard Numerical Attributes can be examined by use of the TEST Block. This Block, its various Operands, and its Auxiliary Operator, are shown in Figure 5.16. The A and B Operands are the names of the two



Operand	Significance	Default Value or Result
A	Name of the first Standard Numerical Attribute	Error
B	Name of the second Standard Numerical Attribute	Error
X	The Auxiliary Operator X represents the relational operator to be used in the test; the forms X can assume are shown below	

Relational Operator	Question Implied in TEST Block Context
G	Is A greater than B?
GE	Is A greater than or equal to B?
E	Is A equal to B?
NE	Is A not equal to B?
LE	Is A less than or equal to B?
L	Is A less than B?

C	Optional Operand; Block Location to which the testing Transaction moves if the answer to the question implied by the relational operator is "no"	The test is conducted in refusal mode when no C Operand is provided

Figure 5.16 The TEST Block and Its Operands

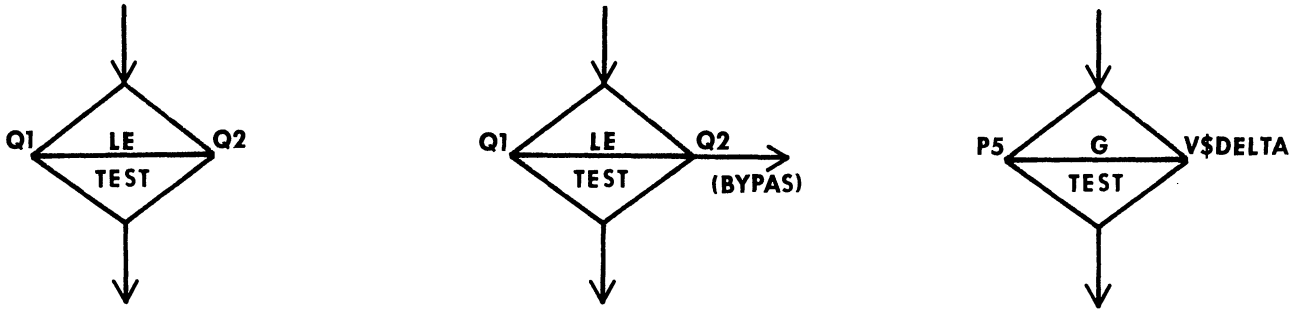
Standard Numerical Attributes involved. An Auxiliary Operator, X, indicates the way the two SNA's are to be compared against each other. As indicated in Figure 5.16, X is to be the single letter G, E, or L (for "greater", "equal", and "less", respectively), or the pair of letters GE, NE, or LE (for "greater or equal", "not equal", and "less or equal", respectively).

The C Operand is optional. When only the A and B Operands are supplied, the test is conducted in refusal mode. When a Transaction attempts to enter a TEST Block used in refusal mode, entry is denied unless the answer to the question implied by the relational operator is "yes". When entry is denied, the Transaction is held at the preceding Block, where it contributes to the Current Block Count. Each time the Current Events Chain is scanned, the Transaction again attempts to gain entry to the TEST Block. Eventually, the TEST Block is successfully entered because the value of one or both of the SNA's involved is presumably subject to change as the simulation proceeds. When a Transaction has gained entry to the refusal-mode TEST Block, it then attempts to move to the sequentia Block, and so on.

When the TEST Block's C Operand is used, the test is conducted in transfer mode. A Transaction which arrives at the TEST Block moves to the sequential Block if the answer to the implied question is "yes"; otherwise, it moves to the indicated non-sequential Block.(5f) In transfer mode, then, entry to the TEST Block is not denied. A Transaction arriving at the Block enters it immediately, and the path it is to subsequently follow is immediately determined by execution of the Block subroutine.

It is possible to have a non-zero "Current Block Count" at a TEST Block. This will happen for either test mode whenever the Transaction's next scheduled Block refuses to accept it. Of course, even if a Transaction does come to rest in a TEST Block, its "next Block attempted" has already been determined. There is no need to subsequently re-execute the TEST Block subroutine in behalf of the delayed Transaction.

Several TEST Block examples are shown in Figure 5.17. In Figure 5.17(a), the



(a) Refusal Mode (b) Transfer Mode (c) Refusal Mode

Figure 5.17 Examples of the TEST Block

testing Transaction is to be held at the preceding Block until it is true that the contents of Queue 1 are less than or equal to those of Queue 2. In Figure 5.17(b), the testing Transaction will also move to the sequential Block if the contents of Queue 1 are less than or equal to those of Queue 2. If this condition is not satisfied, the Transaction is to move to the non-sequential Block at the Location BYPAS. In the (c) example, a Transaction is to wait until the value of its P5 exceeds the value of the Variable DELTA. Although the use of arithmetic operators is not permitted at the TEST Block, their use is of course allowed in defining Variables. By use of Arithmetic Variables as the A and/or B Operands, then, the values of arithmetic expressions can be examined with the TEST Block.

When preparing the punchcard corresponding to a TEST Block, the Auxiliary Operator is entered in the Operation Field, in card column 13 (or 13 and 14). This follows the prescription given earlier for entering Auxiliary Operators in punchcards. That is, they appear in the Operation Field, after the Operation itself, and separated from the Operation by a blank column.

(5f) The TEST Block in transfer mode works just the opposite of conditional transfer statements in most programming languages. For example, consider the FORTRAN IV statement "IF (A .LT. B) GO TO 25". If it is true that A is less than B, control transfers to non-sequential statement 25; otherwise, it goes to the sequential program statement. At the TEST Block in transfer mode, the Transaction transfers to the non-sequential Block when the examined condition is false; otherwise, it goes to the sequential Block.

5.9 Case Study 5B: A Problem in Inventory Control

(1) Statement of the Problem

In a retail store, the daily demand for a given item is normally distributed wit a mean and standard deviation of 10 and 2 units, respectively. Whenever the retailer's stock-on-hand drops to or beneath a pre-determined point, called the reorder point, he places a stock-replenishment order with his supplier. The replenishment amount, called the reorder quantity, is always 100 units. The replenishment order arrives at the reta store anywhere from 6 to 10 days after placement of the order. This lapsed time between placing the replenishment order, and having it arrive at the retail store, is termed lead time. The lead-time distribution is shown in Table 5B.1. Demand that

Lead Time, Days	Relative Frequency
6	.05
7	.25
8	.30
9	.22
10	.18

Table 5B.1 Lead-Time Distribution

arises when the reatil is out-of-stock is "lost"; that is, customers whose demand canno be satisfied immediately go elsewhere to transact their business.

The retailer wants to know how his experience with the item will vary, depending on where he sets the reorder point. From the records summarized in Table 5B.1, he knows that lead time is about 8 days on average. Because average demand for the item is 10 units per day, he reasons that the reorder point must be no smaller than 80; otherwise, he would not have enough of the item on hand to meet the lead-time demand, i.e., the demand expected to occur during the lead-time period. He reasons further that by setting the reorder point at a higher level, such as 90 or 100, the possibility that he will experience lost sales while waiting for a replenishment order to arrive decreases. On the other hand, higher reorder points mean that he is carrying more stock on average, which increases the amount of his capital tied up in stock.

Build a GPSS model for the retailer's situation. Design the model to measure the distribution of two random variables: "lost daily sales"; and "number of units carried in inventory". Run the model to estimate these two distributions when the reorder quantity is 100, and the reorder point is 80, or 90, or 100. For each alternative configuration, shut off the model after a simulation of 1,000 days.

For simplicity, assume that the retailer checks his inventory level, and then does or does not place a replenishment order, only at the end of each day's business. Also assume that replenishment orders always arrive after the close of a day's business; this means that none of the replenishment amount can be used to meed demand occuring during the day on which the replenishment order arrives. Also ignore any "weekend problems". Such problems arise because, in practice, the retailer may not be open for business on Saturday and/or Sunday; nevertheless, on Saturdays and Sundays, a replenishment order "in transit" continues to proceed toward its destination. Ignoring the weekend problem is equivalent to assuming that the retailer does business 7 days a week.

(2) Approach Taken in Building the Model

The model is composed of two separate segments. The Demand Segment simulates the daily demand for the item, arranges for as much of that demand to be met as possible, and enters the values of "lost sales" and "stock-on-hand" into GPSS Tables at the end of each day. The Inventory Control Segment maintains a watch over the stock-on-hand situation, and causes a replenishment order to be placed when stock-on-hand is at or below the reorder point and there is no replenishment order already en route from the supplier. These two separate segments "communicate" with each other through a single Savevalue, STOCK, whose value is the current stock-on-hand.

In the Demand Segment, rather than letting demand occur "unit by unit" as the day proceeds, the day's total demand is determined by a single Transaction which enters the model each day and samples from the demand distribution. This total demand, carried in Transaction Parameter 1, is then tested in transfer mode against the Savevalue STOCK to determine if all the day's demand can be met. If it can, STOCK is updated, entries are made in the two Tables, and the Transaction leaves the model. When the day's demand exceeds STOCK, the Transaction takes the non-sequential exit from the TEST Block and proceeds to compute lost sales and set STOCK equal to 0. It then makes entries in the two Tables, and leaves the model.

In the Inventory Control Segment, a single office-worker Transaction is held ahead of a refusal-mode TEST Block, waiting for STOCK to drop to or beneath the reorder point. The reorder point itself is held in the Savevalue ROP (for reorder point). When the TEST Block permits passage, indicating that a replenishment order should be placed, the Transaction moves through it to an ADVANCE Block, where lead time is simulated. Moving from the ADVANCE Block after the lead time has elapsed, the office-worker updates STOCK by adding the reorder quantity to it. The reorder quantity value is held in the Savevalue ROQ (for reorder quantity). The office worker then transfers back to re-establish the watch over the stock-on-hand situation.

The Transaction in the Inventory Control Segment has a lower Priority Level than Transactions moving through the Demand Segment. This priority distinction is made to honor the problem condition that replenishment orders always arrive after the close of a day's business. Suppose, for example, that an order is scheduled to arrive on the 51st day of a simulation. The office-worker Transaction "bringing in the order" was put on the Future Events Chain no later than the end of the 45th simulated day (minimum lead time is 6 days). With certainty, the Demand Segment Transaction was not put on the Future Events Chain until the beginning of the 50th day (pre-scheduled to move into the model on the 51st day).

The sequence of placement on the Future Chain means that, after transfer to the Current Chain, the office worker would be ahead of the Demand Segment Transaction if both were to have the same Priority Level. This, in turn, would mean that the replenishment order would arrive before the day's demand was determined, which would be a violation of the stated problem conditions. Making the Priority Level distinction reverse this situation.

(3) Table of Definitions

Time Unit: 1 Day

GPSS Entity	Interpretation
Transactions	
Model Segment 1	A Store Worker P1: Total demand on the day in question P2: Number of lost sales on the day in question
Model Segment 2	An Office Worker
Functions	
LTIME	Function describing the lead-time distribution
SNORM	Function describing the standard normal distribution
Savevalues	
ROP	Reorder point
ROQ	Reorder quantity
STOCK	Stock-on-hand
Tables	
LOSES	Table used to estimate the distribution of lost daily sales
STOCK	Table used to estimate the distribution of stock-on-hand
Variables	
DMND	Variable whose value is the total demand on the day in question
LOST	Variable whose value is the number of lost sales on the day in question

Table 5B.2 Table of Definitions for Case Study 5B

(5) Extended Program Listing

```
BLOCK                                                                                    CARD
NUMBER   *LOC    OPERATION  A,B,C,D,E,F,G              COMMENTS                         NUMBER
         SIMULATE                                                                          1
         *                                                                                 2
         *       SET NON-STANDARD RANDOM NUMBER SEQUENCES                                  3
         *                                                                                 4
         RMULT   11,33                                                                     5
         *                                                                                 6
         *       DEFINE FUNCTIONS                                                          7
         *                                                                                 8
          LTIME FUNCTION   RN2,D5      REPLENISHMENT ORDER LEAD-TIME DIST'N                9
         .05,6/.3,7/.6,8/.82,9/1,10                                                       10
          SNORM FUNCTION   RN1,C25     STANDARD NORMAL DISTRIBUTION FUNCTION               11
         0,-5/.00003,-4/.00135,-3/.00621,-2.5/.02275,-2                                   12
         .06681,-1.5/.11507,-1.2/.15866,-1/.21186,-.8/.27425,-.6                          13
         .34458,-.4/.42074,-.2/.5,0/.57926,.2/.65542,.4                                   14
         .72575,.6/.78814,.8/.84134,1/.88493,1.2/.93319,1.5                               15
         .97725,2/.99379,2.5/.99865,3/.99997,4/1,5                                        16
         *                                                                                17
         *       INITIALIZE SAVEVALUES                                                    18
         *                                                                                19
         INITIAL  X$ROP,80/X$ROQ,100/X$STOCK,100                                          20
         *                                                                                21
         *       DEFINE TABLES                                                            22
         *                                                                                23
          LOSES TABLE      P2,0,1,17       TABLE FOR LOST DAILY SALES                     24
          STOCK TABLE      X$STOCK,0,10,12 TABLE FOR DAILY STOCK LEVEL                    25
         *                                                                                26
         *       DEFINE VARIABLES                                                         27
         *                                                                                28
          DMND  FVARIABLE  2*FN$SNORM+10   DAILY DEMAND DISTRIBUTION                      29
          LOST  VARIABLE   P1-X$STOCK      DAY'S LOST SALES                               30
         *                                                                                31
         *       MODEL SEGMENT 1                                                          32
         *                                                                                33
1                GENERATE   1,,,,1            CLERK COMES EACH DAY                       34
2                ASSIGN     1,V$DMND          SET P1 = DAY'S DEMAND                      35
3                TEST GE    X$STOCK,P1,TRUBL  CAN DAY'S DEMAND BE MET?                   36
4                SAVEVALUE  STOCK-,P1         YES; DEPLETE STOCK BY DAY'S DEMAND         37
5          TAB   TABULATE   STOCK             RECORD END-OF-DAY STOCK LEVEL              38
6                TABULATE   LOSES             RECORD DAY'S LOST DEMAND (IF ANY)          39
7                TERMINATE  1                 CLERK GOES HOME                            40
8          TRUBL ASSIGN     2,V$LOST          NO; SET P2 = DAY'S LOST DEMAND             41
9                SAVEVALUE  STOCK,0           SOLD OUT; SET STOCK = 0                    42
10               TRANSFER   ,TAB              GO RECORD DAY'S EXPERIENCE                 43
         *                                                                                44
         *       MODEL SEGMENT 2                                                          45
         *                                                                                46
11               GENERATE   ,,,1              SEED THE INVENTORY CONTROL SEGMENT         47
12         WATCH TEST LE     X$STOCK,X$ROP     IS IT TIME TO PLACE A                      48
         *                                      REPLENISHMENT ORDER?                      49
13               ADVANCE    FN$LTIME          THE ORDER IS ON ITS WAY                    50
14               SAVEVALUE  STOCK+,X$ROQ      ORDER ARRIVES; ADD ROQ TO STOCK            51
15               TRANSFER   ,WATCH            GO BACK TO WATCH                           52
         *                                                                                53
         *       CONTROL CARDS                                                            54
         *                                                                                55
                  START      1000                 START 1ST RUN (ROP = 80)               56
                  RMULT      11,33                RESTORE RANDOM SEQUENCES               57
                  CLEAR      X$ROQ                SELECTIVELY CLEAR FOR 2ND RUN          58
                  INITIAL    X$ROP,90/X$STOCK,100 RE-CONFIGURE; SET INITIAL STOCK        59
                  START      1000                 START 2ND RUN (ROP = 90)               60
                  RMULT      11,33                RESTORE RANDOM SEQUENCES               61
                  CLEAR      X$ROQ                SELECTIVELY CLEAR FOR 3RD RUN          62
                  INITIAL    X$ROP,100/X$STOCK,100 RE-CONFIGURE; SET INITIAL STOCK       63
                  START      1000                 START 3RD RUN (ROP = 100)              64
                  END        RETURN CONTROL TO OPERATING SYSTEM                          65
```

Figure 5B.2 Extended Program Listing for Case Study 5B

(4) Block Diagram

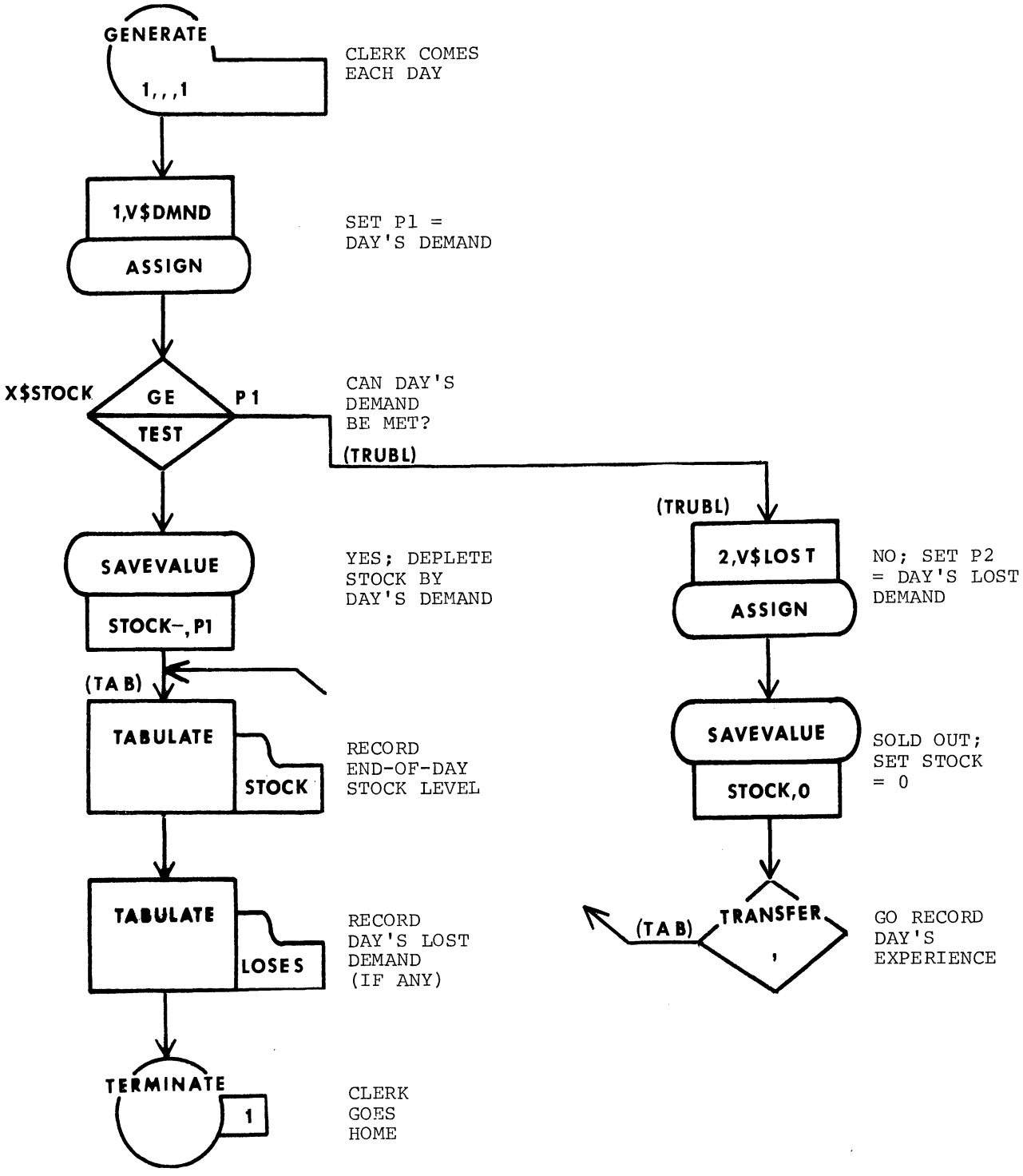
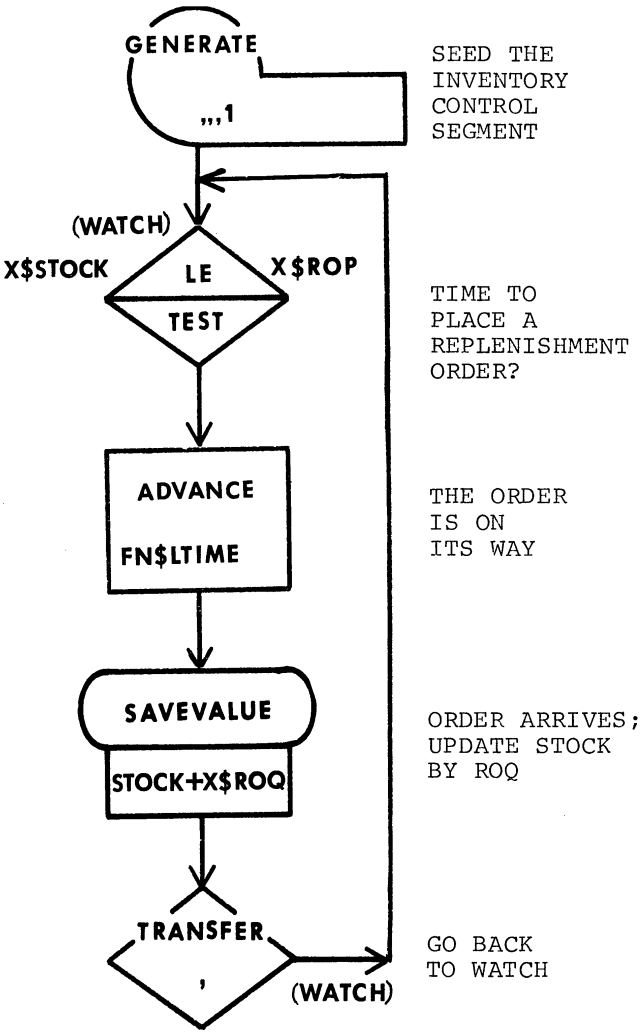


Figure 5B.1 Block Diagram for Case Study 5B

(Continued on Next Page)



MODEL SEGMENT 2

TABLE LOSES

ENTRIES IN TABLE	MEAN ARGUMENT	STANDARD DEVIATION	SUM OF ARGUMENTS	
1000	.516	2.015	517.000	NON-WEIGHTED

UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIATION FROM MEAN
0	923	92.29	92.2	7.7	-.000	-.256
1	10	.99	93.2	6.7	1.934	.239
2	2	.19	93.4	6.5	3.868	.735
3	2	.19	93.6	6.3	5.802	1.231
4	5	.49	94.1	5.8	7.736	1.727
5	5	.49	94.6	5.3	9.671	2.224
6	12	1.19	95.8	4.1	11.605	2.720
7	9	.89	96.7	3.2	13.539	3.216
8	8	.79	97.5	2.4	15.473	3.712
9	6	.59	98.1	1.8	17.408	4.208
10	8	.79	98.9	1.0	19.342	4.704
11	3	.29	99.2	.7	21.276	5.200
12	6	.59	99.8	.1	23.210	5.696
13	0	.00	99.8	.1	25.145	6.193
14	1	.09	100.0	.0	27.079	6.689

REMAINING FREQUENCIES ARE ALL ZERO

(a) Tabulation of Daily Lost Sales

TABLE STOCK

ENTRIES IN TABLE	MEAN ARGUMENT	STANDARD DEVIATION	SUM OF ARGUMENTS	
1000	47.815	30.875	47816.000	NON-WEIGHTED

UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIATION FROM MEAN
0	81	8.09	8.0	91.8	-.000	-1.548
10	75	7.49	15.5	84.3	.209	-1.224
20	94	9.39	24.9	75.0	.418	-.900
30	90	8.99	33.9	66.0	.627	-.577
40	90	8.99	42.9	57.0	.836	-.253
50	93	9.29	52.2	47.7	1.045	.070
60	95	9.49	61.7	38.2	1.254	.394
70	94	9.39	71.1	28.8	1.463	.718
80	101	10.09	81.2	18.7	1.673	1.042
90	95	9.49	90.7	9.2	1.882	1.366
100	69	6.89	97.6	2.3	2.091	1.690
OVERFLOW	23	2.29	100.0	.0		
AVERAGE VALUE OF OVERFLOW		106.86				

(b) Tabulation of Stock-on-Hand

Figure 5B.3 Selected Program Output for Case Study 5B (Configuration: Reorder Point=80)

(7) Discussion

Model Logic

In the Figure 5B.1 Block Diagram, note how the model uses one TEST Block in transfer mode, and another TEST Block in refusal mode. Also note how the two model segments "communicate" with each other through the Savevalue STOCK.

The model is designed for duplication of experimental conditions. The two sources of randomness are "daily demand" and "lead time". Each of the corresponding distributions is represented by a Function (Cards 9 through 16, Figure 5B.2), and each Function has its own random-number souces as an argument. With the RMULT Card (Cards 4, 57, and 61 in Figure 5B.2), these random number sources are set or re-set to given starting points before the simulation begins for each different reorder point.

Program Output[5g]

Figure 5B.3 shows the "lost daily sales" and "stock-on-hand" Tables produced for the simulation corresponding to a reorder point of 80. The number of entries in each Table is 1000, one for each day of the simulation. On average, 0.516 sales were lost each day (MEAN ARGUMENT in the Table LOSES). During the course of the 1000 days simulated, this amounts to 565 lost sales. On about 92% of the days, there were no lost sales (PER CENT OF TOTAL for an UPPER LIMIT of 0 in the Table LOSES). On one occasion, there were 14 lost sales in a single day.

The Figure 5B.3 Table STOCK indicates that stock-on-hand was approximately uniformly distributed between 0 and 100 units. Average stock-on-hand was 47.8 units. About 8% of the time, stock-on-hand was zero at the end of the day's business, exclusive of the possibility that a replenishment order might be due in at the end of that day (PERCENT OF TOTAL for an UPPER LIMIT of 0 in the Table STOCK). On 23 of the simulated days, stock-on-hand at the end of the day exceeded 100 units.

Table 5B.3 shows a summary of the Table output for all three reorder-point alternatives investigated. The trends shown in the table are consistent with expecta-

	Average Daily Lost Sales	Percent of Days Stockout Occurred	Average Stock-on-Hand
80	0.516	7.7	47.8
90	0.173	3.2	55.0
100	0.005	0.4	65.0

Table 5B.3 Summary of Selected Program Output for Case Study 5B

tions. Average daily lost sales decreases as the reorder point is raised. At the same time, the average stock-on-hand increases with an increasing reorder point.

There are many more features of "the inventory control problem" which could be discussed. Many of these features are included in the set of exercises which follows.

[5g] Total CPU time for the simulation was 14.3 seconds.

5.10 Exercises

5.10.1 These exercises are designed for practice in use of the TEST Block.

(a) An analyst wants a Transaction arriving at Point A in a model to move to the sequential Block if the content of Queue 1 is less than that of Queue 2; otherwise, the Transaction is to move to the Block at RUTE7. Show the details of the TEST Block the analyst should place at Point A.

(b) Show how to hold a Transaction at Point A in a model until the remaining capacity of the Storage ALPHA exceeds the remaining capacity of the Storage BETA.

(c) When a customer arrives at a certain barber shop, he decides to wait for service only if 5 or less customers are already waiting for service. Otherwise, he leaves, and does not return later. Show a portion of a Block Diagram to simulate this situation. (Recall that, when a Storage is used to simulate waiting space, the TRANSFER Block can be used in BOTH mode to simulate this same situation. See Figure 2.57.)

(d) A Transaction is to be held at Point A in a model until the utilization of the Facility CRANE exceeds 50%. Show how this can be accomplished.

(e) Why is the Block "TEST LE Q1+Q2,Q3,BYPAS" in error? Suggest a method for eliminating the error while accomplishing the apparent purpose of the TEST Block.

(f) When a Transaction reaches Point A in a model, it is to move to the Block in Location LOC1 if Queues ONE, TWO, and THREE are all empty; otherwise, it is to move to LOC2. Show how this can be accomplished.

(g) Transactions arriving at Point A in a model choose their "next Block" at random. In the long run, 35% of them move next to the Block in Location BLOK1; the other 65% move next to the Block in Location BLOK2. Show a TEST Block which can be used at Point A to simulate this random selection of a next Block. (Note: the same effect can be accomplished with the TRANSFER Block in statistical transfer mode, as described in Chapter 2. It is more efficient to use the TRANSFER Block than the TEST Block for this purpose.

(h) Show how a Savevalue can be used to simulate a Facility. Discuss the advantages and disadvantages of such a simulation.

(i) Show how a Savevalue can be used to simulate a Storage. Discuss the advantages and disadvantages of such a simulation.

5.10.2 These exercises involve using Savevalues to simulate Queues.

(a) Discuss the possibility of using a Savevalue to simulate a Queue. Include in your discussion such things as (1) amount of computer memory saved by such a simulation, (2) probable direction of change in the CPU time requirement, and (3) change in the number of statistics available which describe the waiting process. Can you think of one or more situations in which simulating a Queue with a Savevalue would provide an adequate representation of the Queue?

(b) Show how to use a Savevalue to simulate the Queue in your solution to 5.10.1(c), thereby making it possible to eliminate the QUEUE/DEPART Block pair.

(c) In Case Study 4C (the bank queuing problem), the only reason for having individual Queues ahead of each teller in the multiple-line system was to provide the "Qj" statistic for use as the simulation proceeded. Show how to simulate the individual Queues with Savevalues in that problem, thereby eliminating the individual Queues. Do this first of all by leaving both SELECT Blocks in the solution as shown in Case Study 4C. Then do it by an alternative approach, eliminating the first of the two SELECT Blocks in the process. (See Problem 4.16.5.)

5.10.3 These exercises refer to Case Study 5B.

(a) What are the numbers of the Transactions used in the Figure 5B.2 simulation?

(b) Show how Transactions in both model segments can have Priority Levels of 0, and still meet the problem condition that "replenishment orders come in after a day's business has been concluded". Except for eliminating the GENERATE Block E Operand in the Demand segment, do not change any other Blocks in the model.

(c) The Figure 5B.2 model does not count replenishment orders received at the end of a day as contributing to that day's entry in the stock-on-hand Table. Show what changes to make in the model to reverse this situation.

(d) A backorder is said to be placed when a customer whose demand cannot be met immediately agrees to wait until the next replenishment order arrives to have his demand satisfied. Assume that each customer's demand is for exactly 1 unit of the item, and that the probability is 0.35 that a customer will be willing to backorder. Show what changes to make in the Figure 5B.2 model to incorporate these assumptions.

(e) Assume the probability a backorder will be placed depends on the number of units already on backorder, as shown in Table T5.10.3.

Number of Units Already Backordered	Backorder Probability
0 - 2	0.98
3 - 5	0.80
6 - 8	0.50
9 - 10	0.20
More than 10	0.0

Table T5.10.3

Show what changes to make in the Figure 5B.2 model to incorporate these backorder probabilities.

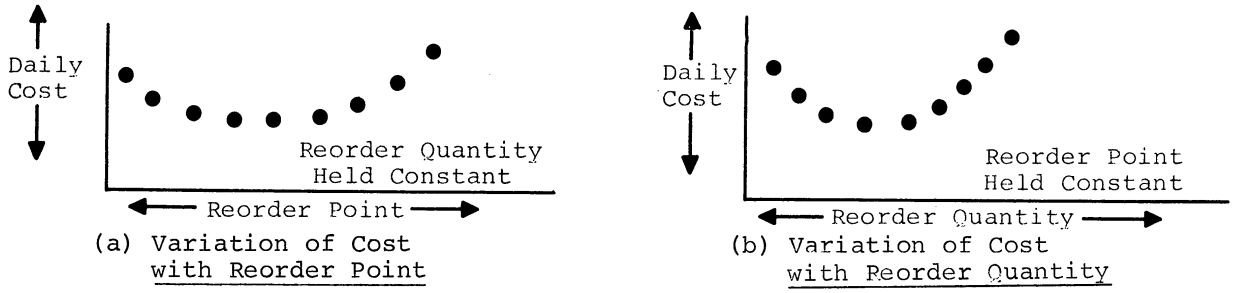
(f) In Case Study 5B, a replenishment order is placed whenever stock-on-hand is less than or equal to the reorder point, and no replenishment order is already en route. Show how to modify the Figure 5B.2 model so that a replenishment order is placed whenever the sum of stock-on-hand and replenishment orders already en route (if any) is less than or equal to the reorder point.

5.10.4 (a) There are usually three cost components in an inventory control problem: the marginal cost of placing a replenishment order; the cost of a lost sale due to stockout; and the cost of carrying inventory. In Case Study 5B, assume that the marginal cost of ordering is $10; the cost of a lost sale (because of the profit thereby lost) is $1; and the cost of carrying 1 unit of inventory for 1 year is 25% of the item's $10 cost. By hand, compute the "average daily cost" for each of the three different reorder-point policies used in Case Study 5B. Which reorder point corresponds to the lowest average daily cost? (From Block Counts not shown in the Case Study 5B program output, there were 90, 93, and 95 replenishment orders placed during the simulations for reorder points of 80, 90, and 100, respectively.)

(b) Using your computations from (a), plot the average daily cost against the reorder point. Does the cost appear to pass through a minimum? If not, use the model to experiment further until the minimum-cost reorder point corresponding to the fixed reorder quantity of 100 has been found.

(c) Show how to modify the Case Study 5B model to automate the computations requested in (a).

5.10.5 In the inventory control problem, average daily cost usually passes through a minimum as the reorder quantity is varied while the reorder point remains fixed. This is suggested in Figure P5.10.5(a), where average daily cost is high for low reorder points (because stockout costs contribute inordinately to total costs), and is high again for high reorder points (because carrying costs contribute excessively to total costs). By the same token, average



(a) Variation of Cost with Reorder Point

(b) Variation of Cost with Reorder Quantity

Figure P5.10.5

daily cost usually passes through a minimum as the reorder quantity is varied while the reorder point remains fixed, as suggested in Figure P5.10.5(b). Low reorder quantities produce high average costs, because of the frequently-incurred marginal cost of ordering; and high reorder quantities result in high average costs, because of the large carrying cost component thereby incurred. In a three-dimensional setting, where average daily cost is plotted against reorder point on one axis, and reorder quantity on another, the cost can easily be visualized as passing through a minimum. [The discussion in Case Study 2E should now be reviewed. In that problem, cost also depended on the choice of two independent decision variables, and was visualized as passing through a minimum. The "univariate searching strategy" was explained as a technique with which to methodically search for the low-cost point.]

In Case Study 5B, let both the reorder point and the reorder quantity be variable. Using the cost information given in Problem 5.10.4, experiment with the model to find the reorder point and reorder quantity combination for which the average daily cost is a minimum.

5.10.6 In an inventory control setting, an important random variable is "lead-time demand". Lead-time demand is the total demand that occurs while lead time elapses. It is dependent on both the daily-demand distribution, and the lead-time distribution. Build a GPSS model to estimate the distribution of lead-time demand for the data given in Case Study 5B. Base the distribution estimate on a sample consisting of 1,000 lead-time experiences. Use the output to estimate where the retailer should set his reorder point if the probability of stockout is to be (a) 0.05, (b) 0.10, and (c) 0.15.

5.10.7 This problem involves a scheduling-period, target-level approach to inventory control, in contrast with the reorder-point, reorder-quantity approach used in Case Study 5B.

A retailer knows from past records that the daily demand for a certain item varies randomly according to the pattern in Table T5.10.7(a). The lead time required for replenishment orders to be filled is distributed as

Quantity Demanded	Relative Frequency
0	0.30
1	0.35
2	0.13
3	0.11
4	0.07
5	0.04

Table T.10.7(a)

shown in Table T5.10.7(b). When the retailer is out of stock, the

Lead Time, Days	Relative Frequency
4	0.30
7	0.60
8	0.10

Table T5.10.7(b)

probability is 0.65 that, instead of backordering, a customer will go elsewhere to have his order filled.

The retailer controls his supply of the item by placing replenishment orders at the opening of business either every Monday, or every second Monday. (The number of weeks between replenishment orders is termed the scheduling period.) As the lead-time data show, his replenishment order then arrives at the opening of business on a Friday, Monday, or Tuesday. He is not open for business on Saturday or Sunday.

The retailer's reorder quantity is variable. It is computed by subtracting the number of units of stock-on-hand from a pre-determined, invariant value called the target level.

The cost of placing a replenishment order is $40. Carrying cost is $1 per item per day (including weekends). The cost of a lost sale is $100.

Build a GPSS model to simulate this situation. Then use the model to determine whether the scheduling period should be 1 or 2 weeks, and what the target level should be if the objective is to minimize average weekly cost. Cost computations should be included as part of the model.

5.10.8 A machine which uses a particular type of part must be shut off from time to time because of part failure. In contrast to the situation described in Case Study 4E, failed parts cannot be repaired; they are simply discarded. Hence, the supply of spares is gradually depleted and must be replenished periodically by placing orders with the parts supplier.

The average part lifetime is 240 hours. The lifetime is exponentially distributed. Lead time for replenishment orders is 30±10 days. As long as parts are available, the machine is kept running, 24 hours a day, 7 days a week. The time required to remove a failed part, and install a spare, is negligibly small.

A reorder-point, reorder-quantity approach is used to control the inventory of spare parts. Replenishment orders are placed whenever the sum of parts-on-hand and replenishment-parts-en-route is at or below the reorder point.

Build a GPSS model to estimate the fractional machine utilization for various choices of reorder point and reorder quantity. What combination minimizes total costs if the marginal cost of ordering is $20, the cost of carrying a spare in inventory is $5 per day, and the cost of having the machine out of use is $1 per hour?

5.10.9 Two important random variables associated with a waiting line are "residence time in the line", and "length of the line". In the Queue statistics provided by GPSS, the average value of these two random variables is automatically provided. (The average residence time is given both for all Queue entries, and for non-zero Queue entries. The average line length is provided as "average content".)

Furthermore, when the GPSS Qtable is used, an estimate of the distribution of the residence time random variable is automatically provided in the simulation output. There is no such "automatic" means, however, for estimating the distribution of the "line length" random variable. The analyst must supply his own logic to estimate this distribution.

Suppose it is of interest in a particular model to gather information showing what fraction of the time the Queue LONG is of length 0; 1; 2; 3; 4; 5; and of length 6 or greater. Show a self-contained model segment which causes this information to be collected in a GPSS Table.

5.11 Entities to Simulate Control Elements: Logic Switches

Consider the concept of "entities whose purpose is to signal that a given condition exists in a system". Entities that provide such signals can usually be thought of as elements which control the sequence of events in a system. Here are some examples of system control elements.

(a) the lock on a door
(b) a traffic light
(c) "Go to Next Window" sign at a bank teller's window
(d) "Lot Full" sign at a parking lot
(e) "Closed" sign in a gas station window

In GPSS, it would be an easy matter to use Savevalues to simulate such control elements in models. For example, the Savevalue LOCK could be used to indicate whether a barber shop is open or not. X$LOCK values of 1 and 0 could be understood to mean that the shop is or is not open, respectively. As simulated by a Transaction, a customer arriving at the door of the barber shop could attempt to move through a TEST Block which tests to determine whether X$LOCK equals 1. Only if the shop is open would the customer be permitted to enter; otherwise, he would be refused entry.

Rather than forcing the analyst to use Savevalues in this fashion to simulate control elements, GPSS makes available a type of entity designed explicitly for this purpose. "Logic Switches" are the entities used to simulate control elements. Logic Switches are "two-position" switches. Rather than being called "on" and "off", the two different switch positions are termed "Set" and "Reset", respectively. At any given time, then, a given Logic Switch in a model is either Set, or Reset. Of course, the setting of selected Logic Switches can be reversed as a simulation proceeds, to reflect changing model conditions. And, Logic Switch settings can be tested and in this sense used to influence the movement of Transactions in a model.

There are at least two advantages to using Logic Switches instead of Savevalues to simulate system control elements. First of all, it is presumably more natural for the analyst to think about control elements in an "on-off" (i.e., Set-Reset) sense than in terms of the numeric encoding scheme which would be required with Savevalues. And second, the execution time required to modify and test Logic Switch settings is less than that required to perform the same actions with respect to Savevalues. Of course, Logic Switches are limited by being restricted to only two possible settings. On occasion, it might be necessary to use Savevalues to simulate more complicated control elements. A traffic light, for example, usually is in one of three states, red, green, or amber. These three states could be represented more easily with a Savevalue than with a Logic Switch.

The normal quantities of Logic Switches in a model with 64, 128, and 256K bytes of computer memory are 200, 400, and 1000, respectively. Like Facilities, Storages, Queues, Functions, Tables, and Savevalues, Logic Switches can be named numerically or symbolically. The usual rules hold when choosing names.

5.12 Controlling Logic Switch Settings: The INITIAL Card and the LOGIC Block

In this section, the method for pre-setting Logic Switch positions before a simulation begins, and reversing the positions as a simulation proceeds, will be described. Then, in section 5.13, the Block used to test the status of Logic Switches will be introduced.

5.12.1 The INITIAL Card

Like Savevalues, Logic Switches automatically exist in a model, whether they are used or not. Normally, the Processor puts all Logic Switches in the Reset position before a simulation begins. At the analyst's option, selected Logic Switches can initially be put into a Set position by use of the INITIAL Card. The card takes the form shown in Figure 5.18. As indicated, the Location Field is not used. The word INITIAL is entered

Location	Operation	Operands
Not Used	INITIAL	$LSname_1/LSname_2/ \ldots /LSname_i/ \ldots /LSname_n$

Figure 5.18 Logic Switch Initialization Card Format

in the Operation Field. Each unit of information in the Operands Field consists of LS, followed by the numeric or symbolic name of a Logic Switch whose initial position is to be Set. When symbolic names are used, a dollar sign ($) must be interposed between LS and the symbolic name itself. Examples of possible entries in the Operands Field are LS5, LS21, LS$LOCK, and LS$SYGNL.

Slashes (/) are used to separate consecutive units of information on an INITIAL Card. The first blank column encountered by the Processor in the Operands Field is interpreted as an "end of card" character. No information (except for comments) can be entered beyond card column 71. More than one INITIAL Card can be used, if necessary.

Figure 5.19 provides several examples of the Logic Switch INITIAL Card. As

```
LOCATION      OPERATION     A,B,C,D,E,F ------>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
              INITIAL       LS4/LS2/LS231/LS50                                  FIRST EXAMPLE
              INITIAL       LS5/LS$MARK/LS$FLAG/LS21                            SECOND EXAMPLE
              INITIAL       LS5/LS8/LS9/LS10/LS11/LS2                           THIRD EXAMPLE
              INITIAL       LS5/LS8-LS11/LS2                                    FOURTH EXAMPLE
```

Figure 5.19 Examples of the Logic Switch INITIAL Card

indicated in the first example, it is not necessary that the numbers of the Logic Switche involved be in ascending order on the INITIAL Card. The second example shows a mixture of Logic Switches named numerically and symbolically. The GPSS Processor establishes a correspondence between symbolic names and their numeric equivalent by the two-step procedure already described in Chapter 5 in the context of other language entities. The third example reveals that Logic Switches 8, 9, 10, and 11, among others, are initially to be in the Set position. When Logic Switches whose numbers form an uninterrupted sequence of integers are all to be Set initially, the corresponding entries on the INITIAL Card can be compacted to the form "LSi-LSk", where i and k are the smallest and largest Logic Switch numbers, respectively, in the range of switches to be Set. The fourth example in Figure 5.19 repeats the third example, making use of this compact form.

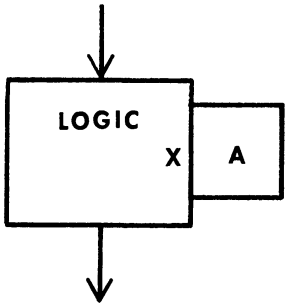
Logic Switch settings are not affected by the RESET Card. The CLEAR Card causes all Logic Switches in a model to be put into the Reset position. Unlike the situation with Savevalues, there is no Selective CIEAR Card whereby certain Logic Switches can be left intact when the clearing operation takes place.

At the end of a simulation, the names of all Set Logic Switches are automatically printed out. The positions of selected Logic Switches can also be outputted during a run by use of the PRINT Block. This is accomplished by providing LG as the Block's Field C mnemonic. For example, when a Transaction moves into the Block "PRINT 5,10,LG",

the positions of Logic Switches 5 through 10 will be printed out.

5.12.2 The LOGIC Block

The status of a Logic Switch can be changed by having a Transaction enter the LOGIC Block. This Block, and its A Operand and Auxiliary Operator, are shown in Figure 5.20. When a Transaction enters the Block, the referenced Logic Switch is put into a



Operand	Significance
A	The name (numeric or symbolic) of a Logic Switch
X	The Auxiliary Operator X indicates what is to be done to the indicated Logic Switch; the forms X can assume are shown below.

X	Effect Produced
R	Reset the Logic Switch
S	Set the Logic Switch
I	Invert the Logic Switch

Figure 5.20 The LOGIC Block, Its A Operand, and Its Auxiliary Operator

Reset or Set position, or Inverted, depending on whether the Auxiliary Operator is R, S, or I, respectively. Table 5.5 shows what the switch position will be after the LOGIC Block subroutine has been executed, depending both on its prior position, and the

Switch Position Before Transaction Enters LOGIC Block	Auxiliary Operator Used	Switch Position After Block Subroutine is Executed
Reset	R	Reset
Set	R	Reset
Reset	S	Set
Set	S	Set
Reset	I	Set
Set	I	Reset

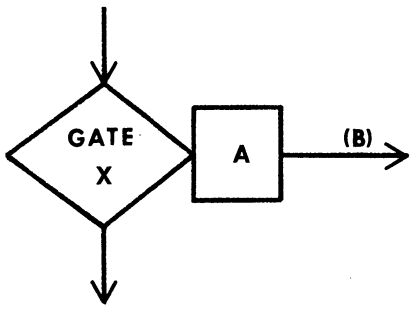
Table 5.5 Various Possible Effects of the LOGIC Block

Auxiliary Operator used. The setting of a Logic Switch is changed with certainty only if the Auxiliary Operator is I. For Auxiliary Operators R and S, the referenced Logic Switch may already be in the desired position at the time a Transaction enters the LOGIC Block.

When a Transaction which has moved through a LOGIC Block finally comes to rest, the Processor re-starts its scan of the Current Events Chain. This is done because the potentially-changed position of the Logic Switch may make it possible for one or more previously-blocked Transactions in a model to resume their movement.

5.13 Testing the Setting of Logic Switches: The GATE Block

When the setting of a Logic Switch is tested, no numeric properties of the switch are involved. In fact, a switch has no numeric properties. This suggests that some Block other than the TEST Block must be introduced to control the flow of Transactions as a function of the Set or Reset status of Logic Switches. The Block used for this purpose is the GATE Block, which is shown in Figure 5.21 with its two Operands and Auxiliary Operator.



Operand	Significance	Default Value or Result
A	Name of a Logic Switch	Error
X	Auxiliary Operator, termed a Logical Mnemonic, indicating the switch setting which is required for the test to be true; the two Logical Mnemonics for Logic Switches are shown below.	Error

Logical Mnemonic	Meaning
LS	Test for Set Condition
LR	Test for Reset Condition

B	Optional Operand; Block Location to which the testing Transaction moves if the Logic Switch is not in the condition required for the test to be true	Test is conducted in refusal mode when no B Operand is provided

Figure 5.21 The GATE Block As Used to Test the Status of Logic Switch

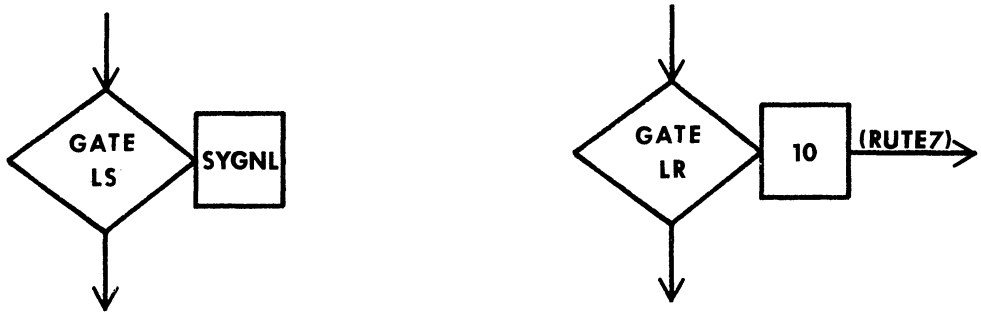
The GATE Block's A Operand states the name of the Logic Switch to be tested. If the name is numeric, it can be supplied directly, or indirectly. The Auxiliary Operatc is a Logical Mnemonic which indicates the Logic Switch setting required for the test tc be true. If the Logical Mnemonic LS is used, the TEST is true if the switch is Set; otherwise, the test is false. When LR is the Logical Mnemonic, the test is true if the switch is Reset, and is false otherwise.

The B Operand is optional. When no B Operand is used, the test is said to be conducted in refusal mode. If a refusal-mode test is not true, "the GATE is closed". When a Transaction finds the GATE closed, it is held at the Block preceding the GATE, contributing to the Current Count there. From a chain viewpoint, the blocked Transaction remains on the Current Events Chain, and an internal flag on the Transaction is turned "on" to indicate that the cause of the blockage is the status of a certain Logic Switch. Because the Transaction's flag is "on", the Transaction is said to be scan-inactive. This means the Processor will not try to move the Transaction in subsequent scans of the Current Events Chain. This has important benefits in terms of execution time economy.

Later in the simulation, when a LOGIC Block referencing the Logic Switch is executed (thereby possibly changing the switch setting), the Processor turns the flag of the blocked Transaction (or Transactions) "off". This makes the Transaction (or Transactions) scan-active again. In the Current Events Chain re-scan which then occurs because the LOGIC Block was executed, the Processor will try to move the previously-blocked Transaction (or Transactions) through the GATE. This flag-setting aspect of the Processor's internal logic, only briefly mentioned here, will be considered in more detail later.

When the GATE Block's B Operand is used, testing at the GATE is conducted in transfer mode. A Transaction which arrives at the GATE moves to the sequential Block if it is true that the Logic Switch has the indicated setting; otherwise, it moves to the non-sequential Location indicated via the B Operand.(5h) In transfer mode, then, entry to the GATE is not denied. A Transaction arriving at the Block enters it immediately, and the path it then will follow is determined by execution of the Block subroutine. If the first Block in that path refuses entry, the Transaction is left on the Current Events Chain, contributing to the Current Count at the GATE Block in question. Eventually, the blocking condition at the Transaction's next Block will be removed, and forward motion of the Transaction in the model will be resumed.

Examples of the GATE Block are given in Figure 5.22. In (a), the GATE is closed



(a) Refusal Mode (b) Transfer Mode

Figure 5.22 Examples of the GATE Block Used to Test Logic Switch Status

whenever the Logic Switch SYGNL is Reset, and is open otherwise. When the GATE is closed, Transactions must wait at the Block preceding the GATE. In (b), Transactions entering the GATE move to the sequential Block next if Logic Switch 10 is Reset; otherwise, their next-Block-attempted is the one located at RUTE9 in the model.

5.14 Use of the GATE Block with Facilities and Storages

Whether a Logic Switch is Set or Reset involves a logical condition, not a numeric one. Several logical conditions associated with Facilities and Storages can also be identified. In particular, whether a Facility is in use or not, and whether a Storage is full or not, or empty or not, pertains to the logical condition of these entities. To cover these conditions, a series of Logical Mnemonics pertaining to Facilities and Storages is supplied in GPSS. With these mnemonics, the GATE Block can be used to make Transaction movement depend on the logical status of Facilities and Storages.

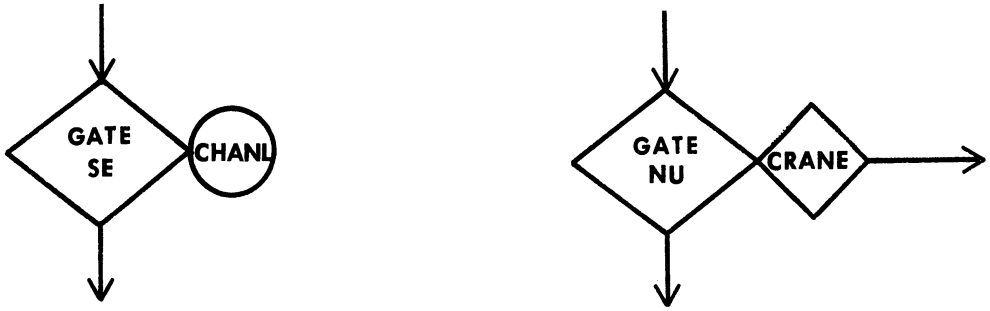
(5h) Note that the GATE and TEST Blocks in transfer mode are similar with respect to their "true" and "false" exits. The Transaction moves from the GATE to the non-sequential Block if it is false that the Logic Switch has the indicated status.

Figure 5.23 shows the GATE Block Logical Mnemonics supplied for Facilities and

Logical Mnemonic	Meaning
U	Test for Facility in Use
NU	Test for Facility Not in Use
SF	Test for Storage Full
SNF	Test for Storage Not Full
SE	Test for Storage Empty
SNE	Test for Storage Not Empty

Figure 5.23 GATE Block Logical Mnemonics for Facilities and Storages

Storages. Figure 5.24 shows several of these Logical Mnemonics used in a GATE Block context. (Notice the slight variation in the shape of the Block, depending on its use for Logic Switches as shown in Figure 5.22, or for Facilities or Storages as shown in Figure 5.24.) In Figure 5.24(a), Transactions are held at the Block preceding the GATE



(a) Refusal Mode (b) Transfer Mode

Figure 5.24 Examples of the GATE Block Used to Test Storage and Facility Status

until the Storage CHANL is empty. In Figure 5.24(b), Transactions move through the GATE to the sequential Block if the Facility CRANE is not in use; otherwise, they move to the Block in Location PATH2.

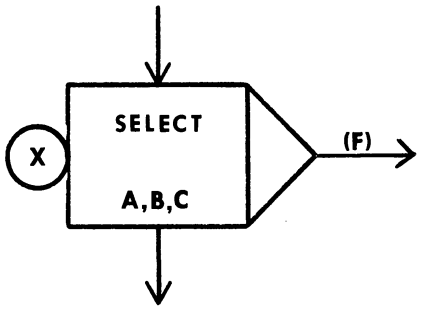
It must be admitted that, for Facilities and Storages, the logical conditions described by the Figure 5.23 mnemonics can be evaluated by testing available numeric information. At the Block "TEST E S$CHANL,0", for example, a Transaction can test to determine if the Storage CHANL is empty. The effect accomplished by this TEST Block is identical to that of the GATE in Figure 5.24(a). Similarly, recalling that the value of the Standard Numerical Attribute F$CRANE is 0 if the Facility CRANE is not in use, the Block "TEST E F$CRANE,0,PATH2" accomplishes the same effect as the GATE in Figure 5.24(b). Why bother, then, to make Facility and Storage Logical Mnemonics available for use with the GATE Block? The reason is that, from the point of view of computer time, GATE Blocks are more efficient than TEST Blocks. When the analyst has a choice, then, between the GATE and TEST Blocks, use of the GATE Block is much to be preferred.(5i)

(5i) The specific reason why GATE Blocks are more efficient than TEST Blocks will be explained in detail in Chapter 6. A quantitative measure of the relative efficiency of GATE Block use will also be provided there.

5.15 SELECT Block Use in Logical Mode

As described in Chapter 4, the SELECT Block can be used in first-such mode to scan a specified set of entity members in search for the first one, if any, which satisfies a stated numeric condition. It is natural to extend the use of this Block so the condition being examined can pertain to the logical status of entities. This makes it possible, for example, to search a group of Logic Switches to find the first one Set, or to search a group of Storages to find the first one not full, etc. The Logical Mnemonics introduced for use with the GATE Block can be used with the SELECT Block to provide these capabilities.

The SELECT Block and its specifications for use in logical mode are shown in Figure 5.25. The A, B, C, and F Operands serve the same purpose as when the SELECT



Operand	Significance	Default Value or Result
A	Number of the Parameter into which the number of the "first-such entity is to be put	Error
B and C	The smallest and largest numbers, respectively, in the range of entities to be examined.	Error
X	The Logical Mnemonic indicating the condition which must be satisfied; the available Logical Mnemonics are shown below.	Error

Logical Mnemonic	Condition to be Satisfied
LS	Logic Switch Set
LR	Logic Switch Reset
U	Facility in Use
NU	Facility Not in Use
SF	Storage Full
SNF	Storage Not Full
SE	Storage Empty
SNE	Storage Not Empty

F	Optional Operand; Block Location to which the selecting Transaction moves if no entity satisfies the required condition	Selecting Transaction moves to sequential Block unconditionally

Figure 5.25 The SELECT Block and Its Operands for Use in Logical Mode

Block is used in first-such mode. As a Logical Mnemonic, the Auxiliary Operator uniquely identifies both the entity type involved, and the condition which is to be satisfied. There is no information, then, that need be supplied by the D and E Operands, and they are therefore not used.

Figure 5.26 shows two examples of SELECT Block use in logical mode. In Figure 5.26(a), Logic Switches 7 through 11 are scanned to find the first one Set. The

(a) No Alternate Exit Specified (b) Alternate Exit Specified

Figure 5.26 Examples of SELECT Block Use in Logical Mode

number of the pertinent Logic Switch is to be placed in Parameter 15 of the selecting Transaction, which then proceeds to the sequential Block. If none of the Logic Switches between 7 and 11 are set, the selecting Transaction simply proceeds to the sequential Block with its Parameter 15 value unaltered.

In Figure 5.26(b), Storages 5 through 8 are examined to find the first one not full. If a not-filled Storage is found, its number is copied into Parameter 1 of the selecting Transaction, which then proceeds to the sequential Block. If all Storages in the range from 5 through 8 are full, Parameter 1 is left unchanged, and the selecting Transaction takes the non-sequential exit to the Block in the Location FULUP.

With respect to specifying the range of the entity involved, the SELECT Block in logical mode is subject to the same restriction as the SELECT Block in first-such mode, and in MIN and MAX mode. That is, the B and C Operands must be specified numerically, either directly or indirectly. Symbolic entity names cannot be used.

5.16 Case Study 5C: A Gas Station Problem

(1) Statement of the Problem

The inter-arrival time distribution of cars approaching a gas station with the intention of possibly stopping for service is shown in Table 5C.1. Service time for

Inter-arrival Time, Seconds	Cumulative Frequency
Less than 0	0.0
100	.25
200	.48
300	.69
400	.81
500	.90
600	1.0

Table 5C.1

cars follows the distribution shown in Table 5C.2. A car stops for service only if the number of cars already waiting for service is less than or equal to the number of cars currently being served. (That is, a car stops only if the driver concludes that not more than "one car per attendant" is already waiting to be served.) Cars which do not stop go to another gas station, and therefore represent "lost business".

Service Time, Seconds	Cumulative Frequency
Less than 100	0.0
200	.06
300	.21
400	.48
500	.77
600	.93
700	1.0

Table 5C.2

The gas station is open from 7 a.m. until 7 p.m. The "lights are turned out" at 7 p.m., meaning that cars arriving later than that are not accepted for service. Any cars already waiting in line at 7 p.m. are served, however, before the attendants leave for the night.

It is estimated that the profit per car served averages $1, excluding attendants' salaries and other fixed costs. Attendants earn $2.50 per hour and are only paid for working a 12-hour day, even if they stay beyond 7 p.m. to finish service on previously-waiting cars. The other fixed costs amount to $75 per day.

The station's owner wants to determine how many attendants he should hire to maximize his daily profit. Build a GPSS model simulating the operation of the station, then use the model to provide the answer. Simulate with each "attendants-hired" configuration for 10 different days. Use the GPSS Table entity to tabulate the resulting distribution of the daily profit. Design the model to control experimental conditions, so that the various alternatives are investigated under identical sets of circumstances.

(2) Approach Taken in Building the Model

The basic structure of the model is straightforward. There are only three features of the problem which require discussion. The first concerns conditions under which the gas station is "shut down" at the end of each day; the second involves using the Table entity to estimate the daily profit distribution; and the third deals with the computation of each day's profit.

Shutting down the station requires coordination between the major model segment, and the timer segment. After 12 hours of operation, no additional cars are to be allowed to enter the major model segment. This is accomplished by using a refusal-mode GATE after the GENERATE Block through which cars arrive. When the Logic Switch LOCK is "Reset", the GATE is open, meaning that cars can enter the model. The Logic Switch is initially "Reset", by default. Then, when a timer-Transaction arrives after 12 hours, it immediately moves through a "LOGIC S GATE" Block, thus closing the GATE in the major model segment. This prevents additional cars from entering the station.

Use of the Table entity is potentially subtle, because the CLEAR Card cannot be used from day-to-day in building up the requested 10-day sample. The reason is that, as part of the clearing operation, the tabled information would be zeroed-out. Hence, to place 10 consecutive "daily profit" entries in the Table, a "START 10" Card is used to begin each 10-day simulation. A timer-Transaction appears at the end of each day, shuts the GATE as described above, and waits for previously-waiting cars to be served. It then computes and tabulates that day's profit, and leaves the model through a "TERMINATE 1" Block. This means that 10 days of simulated time must elapse before

a run shuts off. The desired daily profit tabulation then appears in the model output produced at the end of the run.

Just before each timer-Transaction leaves the model, it moves through a "LOGIC R LOCK" Block. It does this to re-open the GATE in the major model segment for the next day's business.

Computation of a day's profit requires knowing how many cars were serviced that day. This statistic is available through a Block Count in the major model segment. However, it is not enough to use the Total Count at, say, the QUEUE Block for this purpose. Because no clearing operation takes place from day-to-day, the Total Count includes all cars serviced since the beginning of the simulation. To compute the number of cars serviced on day "j", the QUEUE Block's Total Count at the end of day "j - 1" must be subtracted from the Total Count at the end of day "j". This means that, at the end of each day, the Total Count realized "so far" must be saved, for use at the end of the next day. The Total Count is saved in the Savevalue named SOFAR. The timer-Transaction updates the entry in this Savevalue at the end of each day, before leaving the model. The difference between the value of SOFAR and the Total Count at the end of the next day is then computed with the Variable used to determine daily profit.

(3) Table of Definitions

Time Unit: 1 Second

GPSS Entity	Interpretation
Transactions	
Model Segment 1	A car P1: The car's latent service time requirement
Model Segment 2	A timer
Functions	
IAT	A Function describing the inter-arrival time distribution
STIME	A Function describing the service time distribution
Logic Switches	
LOCK	A Logic Switch simulating the "open"-"not open" condition of the gas station
Savevalues	
SOFAR	A Savevalue whose updated value at the end of day "j" is the sum of cars served on days 1, 2, 3, ..., j
Storages	
GUYS	A Storage whose total capacity equals the number of attendants on duty
Tables	
PROFT	A Table used to estimate the distribution of the random variable "daily profit"
Variables	
NET	A Variable whose value is the day's profit after costs

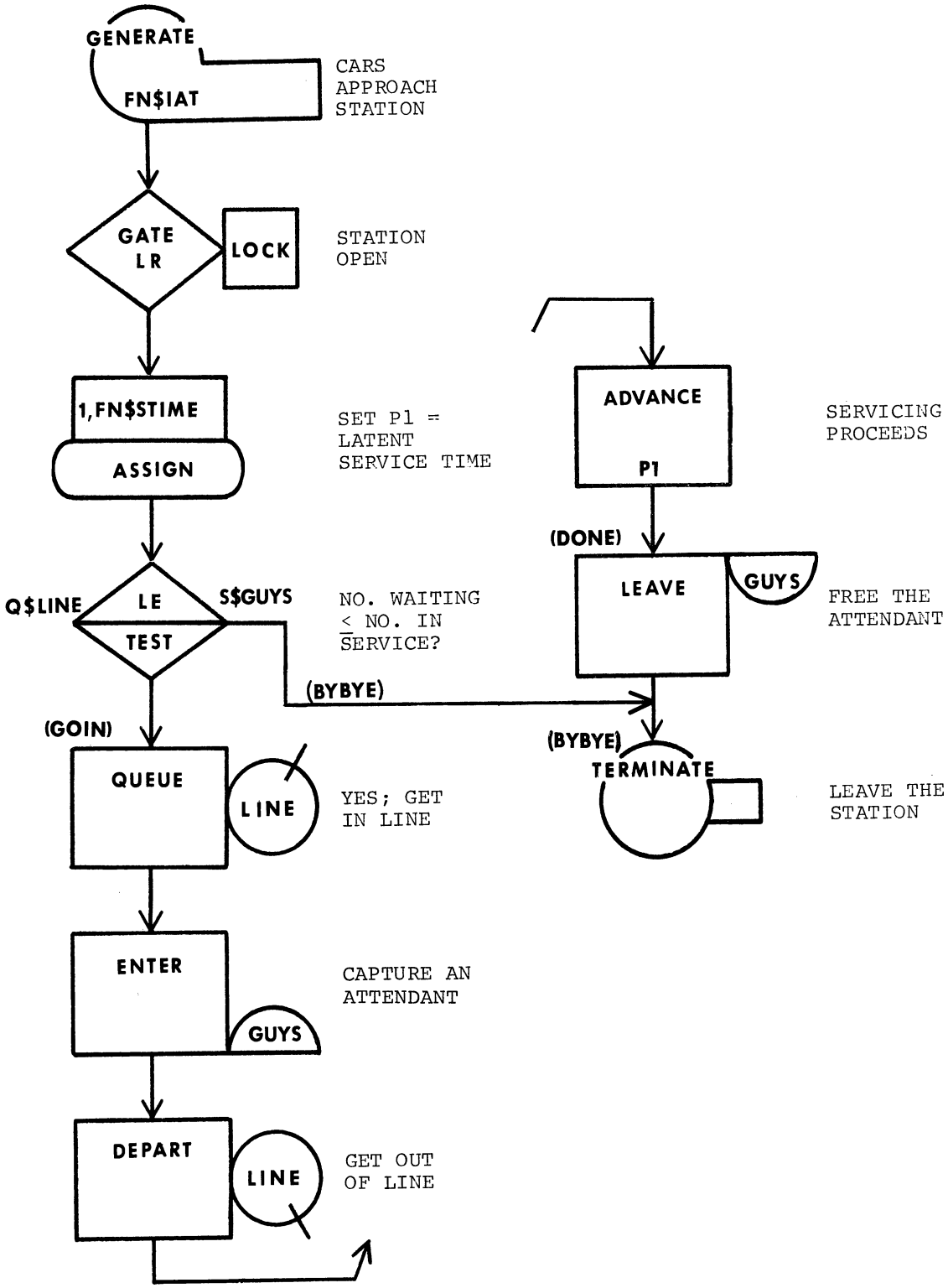
Table 5C.3 Table of Definitions for Case Study 5C

(5) Extended Program Listing

```
BLOCK                                                                    CARD
NUMBER  *LOC    OPERATION  A,B,C,D,E,F,G            COMMENTS             NUMBER
        *       SIMULATE                                                    1
        *                                                                   2
        *       DEFINE FUNCTIONS                                            3
        *                                                                   4
         IAT    FUNCTION   RN1,C7      INTER-ARRIVAL TIME DISTRIBUTION      5
        0,0/.25,100/.48,200/.69,300/.81,400/.9,500/1,600                    6
         STIME  FUNCTION   RN1,C7      SERVICE TIME DISTRIBUTION            7
        0,100/.06,200/.21,300/.48,400/.77,500/.93,600/1,700                 8
        *                                                                   9
        *       DEFINE STORAGE CAPACITIES                                  10
        *                                                                  11
                STORAGE    S$GUYS,1                                        12
        *                                                                  13
        *                                                                  14
        *       DEFINE TABLES                                              15
        *                                                                  16
         PROFT  TABLE      V$NET,50,25,7              TABLE FOR DAILY PROFIT 17
        *                                                                  18
        *       DEFINE VARIABLES                                           19
        *                                                                  20
         NET    VARIABLE   N$GOIN-X$SOFAR-75-30*R$GUYS  DAY'S PROFIT AFTER COSTS 21
        *       MODEL SEGMENT 1                                            22
        *                                                                  23
1               GENERATE   FN$IAT             CARS APPROACH STATION        24
2               GATE LR    LOCK               STATION OPEN?                25
3               ASSIGN     1,FN$STIME         SET P1 = LATENT SERVICE TIME 26
4               TEST LE    Q$LINE,S$GUYS,BYBYE NO. WAITING <= NO. IN SERVICE? 27
5        GOIN   QUEUE      LINE               YES;  GET IN LINE            28
6               ENTER      GUYS               CAPTURE AN ATTENDANT         29
7               DEPART     LINE               GET OUT OF LINE              30
8               ADVANCE    P1                 SERVICING PROCEEDS           31
9        DONE   LEAVE      GUYS               FREE THE ATTENDANT           32
10       BYBYE  TERMINATE                      LEAVE THE STATION            33
        *                                                                  34
        *       MODEL SEGMENT 2                                            35
        *                                                                  36
11              GENERATE   72000             CLOSEUP TIME; OWNER ARRIVES  37
12              LOGIC S    LOCK               SET "NOT OPEN" SIGNAL        38
13              TEST E     N$GOIN,N$DONE      WAIT 'TIL LAST CARS ARE SERVICED 39
14              TABULATE   PROFT              RECORD DAY'S NET PROFIT      40
15              SAVEVALUE  SOFAR,N$GOIN       RECORD CARS SERVICED SO FAR  41
        *                                     THIS RUN                     42
16              LOGIC R    LOCK               SET "OPEN" SIGNAL FOR TOMORROW 43
17              TERMINATE  1                  GO HOME                      44
        *                                                                  45
        *       CONTROL CARDS                                              46
        *                                                                  47
                START      10            START THE 1ST RUN (1 ATTENDANT)   48
                RMULT      1             RESTORE RANDOM SEQUENCE           49
                CLEAR                     CLEAR FOR 2ND RUN                 50
                STORAGE    S$GUYS,2      RE-CONFIGURE FOR 2ND RUN          51
                START      10            START THE 2ND RUN (2 ATTENDANTS)  52
                RMULT      1             RESTORE RANDOM SEQUENCE           53
                CLEAR                     CLEAR FOR 3RD RUN                 54
                STORAGE    S$GUYS,3      RE-CONFIGURE FOR 3RD RUN          55
                START      10            START THE 3RD RUN (3 ATTENDANTS)  56
                END        RETURN CONTROL TO OPERATING SYSTEM              57
```

Figure 5C.2 Extended Program Listing for Case Study 5C

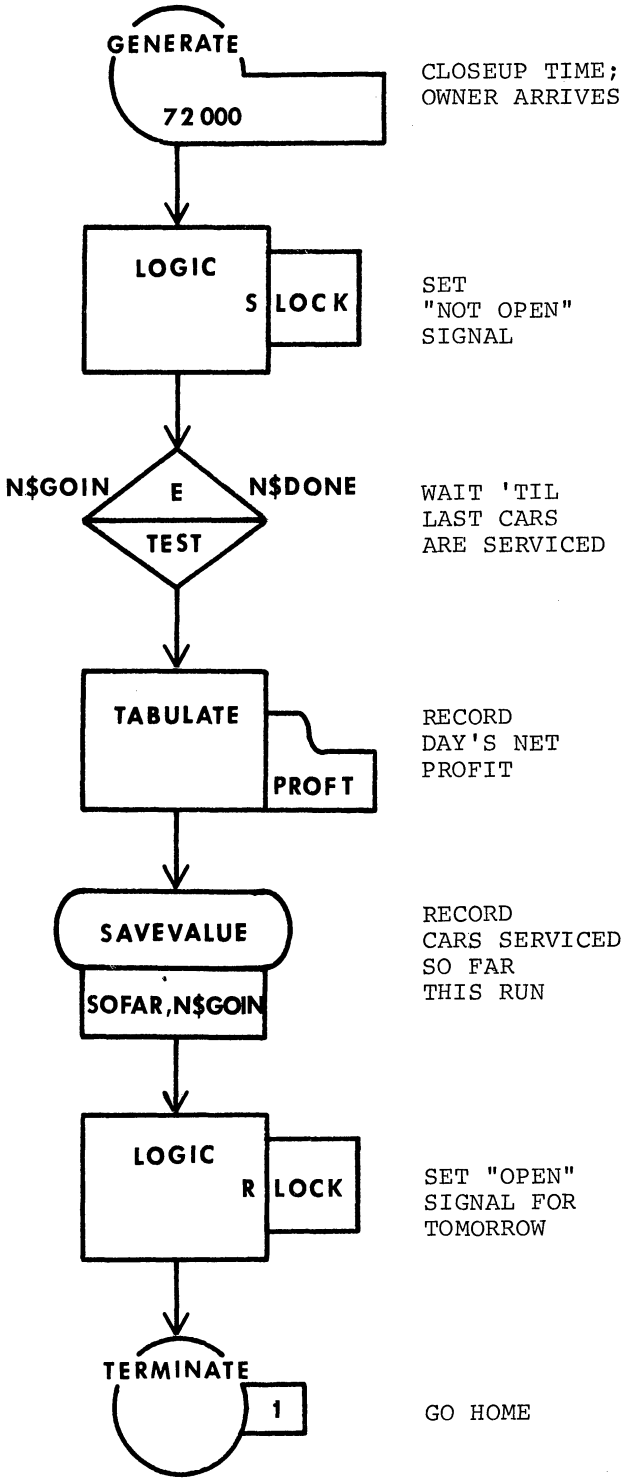
(4) Block Diagram



MODEL SEGMENT 1

Figure 5C.1 (Block Diagram for Case Study 5C

(Continued on Next Page)



MODEL SEGMENT 2

TABLE PROFT

ENTRIES IN TABLE	MEAN ARGUMENT	STANDARD DEVIATION	SUM OF ARGUMENTS	
10	72.099	3.539	721.000	NON-WEIGHTED

UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIATION FROM MEAN
50	0	.00	.0	100.0	.693	-6.244
75	8	79.99	79.9	20.0	1.040	.819
100	2	19.99	100.0	.0	1.386	7.883

REMAINING FREQUENCIES ARE ALL ZERO

(a) 1-Attendant Configuration

TABLE PROFT

ENTRIES IN TABLE	MEAN ARGUMENT	STANDARD DEVIATION	SUM OF ARGUMENTS	
10	159.500	10.308	1595.000	NON-WEIGHTED

UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIATION FROM MEAN
50	0	.00	.0	100.0	.313	-10.622
75	0	.00	.0	100.0	.470	-8.197
100	0	.00	.0	100.0	.626	-5.771
125	0	.00	.0	100.0	.783	-3.346
150	3	29.99	29.9	70.0	.940	-.921
175	7	69.99	100.0	.0	1.097	1.503

REMAINING FREQUENCIES ARE ALL ZERO

(b) 2-Attendant Configuration

TABLE PROFT

ENTRIES IN TABLE	MEAN ARGUMENT	STANDARD DEVIATION	SUM OF ARGUMENTS	
10	143.099	12.703	1431.000	NON-WEIGHTED

UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIATION FROM MEAN
50	0	.00	.0	100.0	.349	-7.328
75	0	.00	.0	100.0	.524	-5.360
100	0	.00	.0	100.0	.698	-3.392
125	0	.00	.0	100.0	.873	-1.424
150	7	69.99	69.9	30.0	1.048	.543
175	3	29.99	100.0	.0	1.222	2.511

REMAINING FREQUENCIES ARE ALL ZERO

(c) 3-Attendant Configuration

Figure 5C.3 Daily-Profit Tables for Case Study 5C

(7) Discussion

In Figure 5C.1, note how the timer-Transaction waits for "previously-waiting cars to be served". The timer does its waiting at the refusal-mode TEST Block in Model Segment 2. The condition which must be satisfied for the test to be true is that N$GOIN equals "N$DONE". GOIN is the Location occupied by the QUEUE Block in Model Segment 1; DONE is the location of the LEAVE Block in that model segment. The two Total Block Counts involved are equal to each other only when the last "previously-waiting" car has been served and has left the model.

In the Variable NET (Card 21, Figure 5C.2), the value of "N$GOIN-X$SOFAR" is the number of cars served on the day in question. Or, at an average profit of $1 per car, it is a given day's revenue. This value is reduced by $75 for non-salary overhead. The salary expense is computed as 30*R$GUYS. The "30" is the $30 per day earned by each attendant (12 hours times $2.50 per hour). R$GUYS is the number of attendants working at the station. Strictly speaking, R$GUYS is the "remaining capacity" of the Storage GUYS; but, when the value of the Variable is computed, the Storage is empty, meaning R$GUYS equals the Storage's total capacity.

RN1 is the random number source for both the inter-arrival time Function, and the service-time Function. The generator's default starting point is used. The starting point is then restored with RMULT Cards (Cards 48 and 52 in Figure 5C.2) before each next configuration is investigated.

When a car approaches the station, its latent service time requirement is computed and saved in Transaction Parameter 1 (Block 3, Figure 5C.2), even before the car "tests" to determine whether to remain for service or not. This guarantees that, from configuration to configuration, identical sequences of "potential business" approach the station. When a car fails to stay to be served, the station not only loses the "business", but also fails to experience the demand-on-resources represented by that "business". If a given arrival did not represent the same latent service time demand for each configuration, the investigations would not be conducted under "identical circumstances". (For another situation in which it is important to "use up a random number" by pre-determining a latent service time requirement, see Problem 3.21.4.)

Program Output[5j]

Figure 5C.3 shows the "profit Tables" for the alternatives of hiring 1, 2, or 3 attendants to work at the gas station. The daily profit corresponding to these alternatives is $72, $159, and $143, respectively. Hence, the 2-attendant case maximizes the daily profit. The standard deviation for the 2 and 3-attendant cases is $10 and $12, respectively, which is large relative to the difference between the average daily profits for these two cases ($16). This means that a misleading conclusion could be drawn if (a) experimental conditions were not duplicated, and/or (b) the sample size was so small that the 3-attendant profit turned out, by chance, to exceed that for the 2-attendant case more often than not.

[5j] Total CPU time for the simulation was 48 seconds.

5.17 Exercises

5.17.1 A certain traffic light is either red or green. When the light switches to red, it remains red for 3 minutes. When it switches to green, it remains green for two minutes. Using the Logic Switch LIGHT to simulate the traffic light, and interpreting Reset as "green", show a self-contained model segment which properly switches the light back and forth. Assume the implicit time unit is 1 second, and that at the start of the simulation, the traffic light has just turned green.

5.17.2 (a) Referring to Appendix B, show a REALLOCATE Card which reallocates the number of Logic Switches in a model downward to 100.

(b) Assuming that the model is to be run at the 64-K level, extend the information in the REALLOCATE Card in (a) so that the memory freed by the downward Logic Switch reallocation is added to COMMON.

(c) Show an INITIAL Card which causes the Logic Switches ALPHA, BETA, and 7 to be Set at the start of a simulation.

(d) Critically discuss this statement: "The CLEAR Card has no effect on Logic Switch settings."

(e) Assuming that a Transaction has caused a LOGIC Block to be executed, what does the GPSS Processor do next after the Transaction has come to rest?

(f) What does it mean to say that a Transaction is scan-inactive? When a Transaction is denied entry to the Block "GATE LR 7" and has been made scan-inactive, under what conditions will it again be made scan-active?

5.17.3 Critically discuss each of the following statements.

(a) "The Current Count at a refusal-mode GATE is always zero."

(b) "If a Transaction comes to rest in a refusal-mode GATE Block, the next Block that Transaction will attempt to enter remains to be determined."

(c) "The Current Count at a transfer-mode GATE Block is always zero."

(d) "If a Transaction comes to rest in a transfer-mode GATE Block, the next Block that Transaction will attempt to enter remains to be determined."

5.17.4 Show refusal-mode GATE Blocks which will be closed unless the following conditions are true.

(a) The Logic Switch MARK is Set.

(b) Facility 9 is not in use.

(c) The Storage TUGS is empty.

5.17.5 Show transfer-mode GATE Blocks which will cause a Transaction to move to the non-sequential Location BYPAS unless the following conditions are true.

(a) The Logic Switch JOE is Reset.

(b) The Facility CRANE is in use.

(c) Storage 21 is not full.

5.17.6 (a) Give a list of all conditions under which an analyst can choose between the GATE and TEST Blocks to accomplish a desired effect. Given a choice, which of the two Block types should an analyst use, and why?

(b) Show GATE Blocks which are equivalent to each of these TEST Blocks.

(i) TEST NE R$SLIPS,0

(ii) TEST E F$CPU,1,BYPAS

(iii) TEST GE R10,1

5.17.7 (a) Show a SELECT Block which scans Facilities 1 through 10 to find the first one in use. Place the number of the first qualifying Facility in Parameter 12 of the selecting Transaction. Whether or not such a Facility is found, the selecting Transaction is to proceed to the sequential Block.

(b) Present another solution for (a), this time using "E" instead of "U" as the SELECT Block's Auxiliary Operator.

(c) List the conditions under which an analyst can use either a Logical Mnemonic or a Relational Operator with the SELECT Block to accomplish a desired effect. Given a choice, would a Logical Mnemonic or a Relational Operator seem more "natural" to use?

(d) Show the punchcard image of a SELECT Block which scans Logic Switches 5 through 8 to find the first one which is Set. Place the number of the first qualifying Logic Switch in Parameter 1 of the selecting Transaction. If none of the switches qualify, route the selecting Transaction to the non-sequential Block at SHUKS.

5.17.8 These questions pertain to Case Study 5C.

(a) Assume that the next car is scheduled to arrive at the gas station at precisely the same time the timer Transaction enters the model. Will this next car find the GATE closed, or not? Give a complete explanation of the reasoning behind your answer.

(b) Show what changes to make in the Figure 5C.2 model to reverse your conclusion for (a). That is, show how to let the hypothesized "last-minute arrival" get through the GATE if he cannot currently do so; or, show how to prevent him from getting through the GATE if he can currently do so.

(c) The timer Transaction waits at a refusal-mode TEST Block until all cars have left the gas station, and it then proceeds to compute and record the profit, and TERMINATE. Show how to replace the refusal-mode TEST Block with a GATE having the same effect. Which of the two Blocks is the better to use, and why?

(d) Critically discuss this statement: "After simulating for the 1-attendant station, the next day's first car will probably arrive at simulated time 1 in the Figure 5C.2 model."

(e) Critically discuss this statement: "The Figure 5C.2 model would be valid even if the two CLEAR Cards were eliminated."

5.17.9 In the spirit of Case Study 5C, show what changes to make in Case Study 4C (comparison of queuing systems in a bank) to incorporate these features: (a) After each 5-hour day, the bank is closed to new customers but remains open to finish servicing all "previously-waiting" customers, and (b) each day's average Queue residence time is placed in a GPSS Table, so that the "average" and "standard deviation" of the "daily average Queue residence time" random variable is automatically provided in the model output.

5.18 Boolean Variables

In GPSS, a Boolean Variable is a user-defined Standard Numerical Attribute. The name of a Boolean Variable is BVj, or BV$sn, where j is the number of the Variable if it has been named numerically, and sn is its symbolic name if it has been named symbolically. The value of a Boolean Variable is either 1 or 0, depending on whether the Boolean expression supplied by the user to define the Variable is true or false, respectively. A Boolean expression, in turn, consists of (a) references to the logical conditions of entities, or (b) logical-valued references to the numeric properties of entities, or (c) logical-valued combinations of the elements in (a) and/or (b). In succeeding parts of this section, the discussion will first focus on the construction of Boolean expressions by using elements of the type in (a), and in (b), and by combining these elements as suggested in (c). Then it will turn to the Boolean Variable Definition Card; examples of Boolean Variables and the rules followed in evaluating them; and, finally, the use of Boolean Variables in GPSS models.

5.18.1 Logical Operators

Logical Operators are used to reference the logical status of the so-called equipment-oriented entities in GPSS, namely Facilities, Storages, and Logic Switches. The available Logical Operators are listed in Table 5.6, with an indication of the

Logical Operators	Entity Condition Referenced
LS	Logic Switch Set
LR	Logic Switch Reset
FU (or F)	Facility in Use
FNU	Facility Not in Use
SF	Storage Full
SNF	Storage Not Full
SE	Storage Empty
SNE	Storage Not Empty

Table 5.6 The Logical Operators in GPSS

entity condition to which they make reference.

As an element in a Boolean expression, a Logical Operator must have appended to it either the number or symbolic name of a specific member of the entity type to which it refers. As usual, if a symbolic name is used, a dollar sign ($) must be interposed between it and the Logical Operator. Examples of Boolean expression elements constructed with Logical Operators are SNE$JANE, FNU21, and LR3. If the Storage JANE is not empty, then SNE$JANE is true; otherwise, it is false. If Facility 21 is not in use, then FNU21 is true; otherwise, it is false. Similarly, if Logic Switch 3 is Reset, then LR3 is true; otherwise, it is false. Single elements such as these constitute simple Boolean expressions.

The strong similarity between the Logical Operators and Logical Mnemonics should be noted. Except for Facilities, they are composed of the same character sequences. Despite the similarity in their appearance, they are used in quite different contexts. Logical Mnemonics are only used as Block Auxiliary Operators. Logical Operators are only used to construct elements in Boolean expressions.

5.18.2 Relational Operators

Relational Operators express a condition which might possibly exist between two pieces of numeric-valued data. The Relational Operators are shown in Table 5.7, with an indication of the tentative relationship they express.

Relational Operators	Condition Expressed
'G'	Greater
'GE'	Greater or Equal
'E'	Equal
'NE'	Not Equal
'LE'	Less or Equal
'L'	Less

Table 5.7 The Relational Operators in GPSS

An element in a Boolean expression can be constructed by placing a Relational Operator between a pair of Standard Numerical Attributes. Examples of elements constructed this way are R$CHANL'GE'3, Q$LINE1'L'Q$LINE2, and P5'G'X$ROP. If the remaining capacity of the Storage CHANL is greater than or equal to 3, then R$CHANL'GE'3 is true; otherwise, it is false. Or, if the content of the Queue LINE1 is less than that of the Queue LINE2, Q$LINE1'L'Q$LINE2 is true; otherwise, it is false. Finally, if P5 is greater than fullword Savevalue ROP, then P5'G'X$ROP is true; otherwise, it is false. Single elements such as these also constitute simple Boolean expressions.

It is evident that the Relational Operators listed in Table 5.7 strongly resemble the Auxiliary Operators available for use with the TEST Block. The same alphabetic characters are used in both instances, but as Relational Operators the characters are enclosed within single quotes (').

5.18.3 Boolean Operators

Boolean data is characterized by the fact that its values, rather than being numeric, are "logical". There are only two possible logical values: true, and false. Logical values are sometimes called Boolean values, or truth values.

A Boolean Operator stands between a pair of logical values, and produces a logical result. Such an operator can be contrasted with a Relational Operator, which stands between a pair of numeric values, and produces a logical value as a consequence. The two Boolean Operators, and the logical values they produce, depending on the logical values between which they stand, are shown in Table 5.8.

Boolean Operator	Name	Result of Operation
*	And	The result is true when the values on both sides of the operator are true; otherwise, the result is false
+	Or	The result is true when either or both of the values between which the operator stands are true; otherwise, the result is false

Table 5.8 The Boolean Operators in GPSS

As shown by earlier examples, elements constructed with Logical Operators, and with Relational Operators, are either true or false in value, i.e., are logical-valued.

As a result, Boolean Operators can be placed between such elements. When one or more Boolean Operators are used in this fashion, a compound Boolean expression results. An example of such a compound Boolean expression would be R5'G'3*LR$SYGNL. If it is true that the remaining capacity of Storage 5 is greater than 3, and that the Logic Switch SYGNL is Reset, then the compound Boolean expression is true; otherwise, the expressior is false.

A second example of a compound Boolean expression is FNU$CPU*(SNF12+SNF$LINE). Note that two Boolean Operators are used in this case. As this example suggests, parer theses can be used to set off part of expressions within compound expressions. When this is done, the information within parentheses is evaluated first; then, the resultir value is used in the context of the overall expression. In this example, if Storage 1 is not full, or if the Storage LINE is not full, or if neither is full, then the paren· thesized expression is true. When the parenthesized expression is true and, in additic the Facility CPU is not in use, then the final value of the entire expression is true; otherwise, it is false.

5.18.4 The Boolean Variable Definition Card

The format of the card used to define a Boolean Variable, shown in Table 5.9, is

Punchcard Field	Information Supplied in the Field
Location	The name (numeric or symbolic) of the Boolean Variable
Operation	Literally, the word BVARIABLE
Operands	The Boolean expression which defines the Boolean Variable

Table 5.9 Boolean Variable Definition Card Specifications

identical to that of an Arithmetic Variable Card. The differences in card content are that the word BVARIABLE is entered in the Operation Field, and the expression in the Operands Field is Boolean, not arithmetic. The other features of Arithmetic Variable Cards apply. In particular, the first blank column encountered in the Operands Field terminates definition of the Boolean Variable; and expressions too long to be containec within card columns 19-71 must be broken up into a series of shorter expressions, and defined through two or more Boolean Variables.

5.18.5 Examples and Rules

Three examples of Boolean Variables are shown in Figure 5.27. Simple Boolean

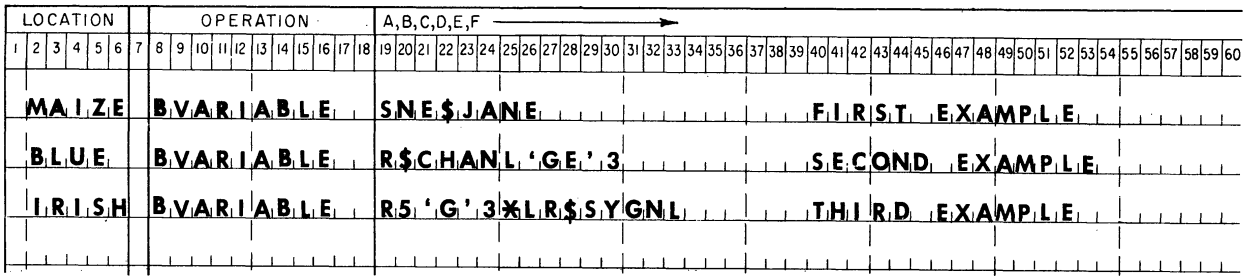


Figure 5.27 A First Set of Examples of Boolean Variables

expressions are shown in the first and second examples. The third example shows a compound Boolean expression. These three examples were among those just discussed in the sub-sections on Logical Operators, Relational Operators, and Boolean Operators,

respectively. Depending on whether the values of the Boolean expressions are true or false, the corresponding values of BV$MAIZE, BV$BLUE, and BV$IRISH will be 1 or 0, respectively.

For those without prior exposure to logical values and Boolean Operators, there may be some initial confusion about the way compound Boolean expressions are evaluated. As was true for Arithmetic Variables, there is a well defined sequence of steps followed when determining the value of a Boolean expression. These steps are especially easily described in GPSS, because arithmetic operators are not permitted in Boolean expressions. Hence, when the symbols + and * are seen in a Boolean expression, they are known to represent Boolean Operators, and not the arithmetic operations of addition and multiplication.

These are the rules for evaluation of Boolean expressions.

(1) Logical and Relational Operators are at the highest precedence level. Their effect takes place first when the Boolean expression in which they appear is evaluated.

(2) The Boolean Operator * (and) is at the next precedence level.

(3) The Boolean Operator + (or) is at the next (and last) precedence level.

(4) When two or more of the same Boolean Operators are used, their effect takes place in left-to-right sequence.

(5) When parentheses are used, everything within parentheses is evaluated first, using rules (1) through (4). The intermediate result produced is then used in the rest of the expression.

Figure 5.28 shows several additional examples of Boolean Variables. In the first

```
OCATION    OPERATION    A,B,C,D,E,F ------>
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
KAPPA      BVARIABLE    C1,'GE',500+SE$SPACE*LS$AMBER                                                    FIRST EXAMPLE
ALPHA      BVARIABLE    FNU1+FNU2+FNU3+FNU4                                                              SECOND EXAMPLE
THETA      BVARIABLE    (C1,'GE',500+SE$SPACE)*LS$AMBER                                                  THIRD EXAMPLE
```

Figure 5.28 A Second Set of Examples of Boolean Variables

example, the two Logical Operators and the Relational Operator first have their effect (Rule 1). Then the Boolean Operator * has its effect (Rule 2). Finally, the Boolean Operator + has its effect (Rule 3). For the Boolean expression to be true, either

(a) C1'GE'500 must be true, or

(b) both LS$AMBER and SE$SPACE must be true, or

(c) the conditions in both (a) and (b) must be satisfied.

In the second example, the Logical Operators first operate (Rule 1). Next, the Boolean Operator * has its effect (Rule 2). Then, the Boolean Operators + have their effect in left-to-right sequence (Rules 3 and 4). The Boolean expression is true if (a) FNU1 is true, or (b) FNU2 and FNU3 are both true, or (c) FNU4 is true, or (d) if any combination of the conditions in (a), (b), and (c) is satisfied.

The third example repeats the first, but with parentheses introduced around part of the expression. After the Logical and Relational Operators have their effect, the result of the parentheses is that the Boolean Operator + operates before the * does (Rule 5). For the Boolean expression to be true, either SE$SPACE or C1'GE'500 (or both)

must be true, and LS$AMBER must be true. The introduction of parentheses makes the conditions required for truth in the third example quite different from those in the first example.

5.18.6 Testing Boolean Variables

Because their values are either 1 or 0, the TEST Block is used to test Boolean Variables in GPSS models. Normally, one thinks of a required set of conditions as being in effect when a Boolean Variable is true, i.e., has a value of 1. As a consequence, Boolean Variables are usually tested to determine if their value equals 1. For example, a Transaction is held at the Block preceding "TEST E BV$ALLOK,1" until the Boolean Variable ALLOK is true. Or, a Transaction moving into the Block "TEST E BV$GREEN,1,RUT continues to the sequential Block if the Boolean Variable GREEN is true, and goes to the non-sequential Location RUTE3 otherwise. Of course, there is no reason why Boolean Variables cannot be tested to determine if they are false, i.e., equal 0. Transactions initiating such testing go to the sequential Block eventually (refusal mode) or immediately (transfer mode) if the variable is false, or go immediately to the non-sequentia Block (transfer mode) if the variable is true.

Use of simple Boolean expressions is to be discouraged. A Boolean expression is simple if it consists of exactly one Logical-Operator element, or of exactly one Relational-Operator element. Consider the following.

(1) In the former case, the GATE Block can be used to achieve the same effect with less execution time. For example, "GATE NU CPU" is logically identical to "TEST E BV$SIMPL,1", where "SIMPL BVARIABLE FNU$CPU" defines the Boolean Variable SIMPL. As indicated previously, substantially less execution time is required with the GATE than with the TEST Block.

(2) In the latter case, the TEST Block can be used to produce the desired logical effect with less execution time. For example, "TEST L Q1,Q2" is logically identical to "TEST E BV$WHY,1", where "WHY BVARIABLE Q1'L'Q2" defines the Boolean Variable WHY. The first TEST Block requires only one comparison (Q1 vs. Q2), whereas the second requires two comparisons (Q1 vs. Q2, then BV$WHY vs. 1).

5.19 Case Study 5D: Oil Tanker Accommodation at a Port

(1) Statement of the Problem

A port in Africa is used to load tankers with crude oil for overwater shipment. The port has facilities for loading as many as three tankers simultaneously. The tankers, which arrive at the port every 11±7 hours, are of three different types. The relative frequency of the various types, and their loading-time requirements, are shown in Table 5D.1.

Type	Relative Frequency	Loading Time, Hours
1	.25	18±2
2	.55	24±3
3	.20	36±4

Table 5D.1 Tanker Specifications for Case Study 5D

There is one tug at the port. Tankers of all types require the services of this tug to move into a berth, and later to move out of a berth. Furthermore, the area experiences frequent storms, and no berthing or de-berthing of a tanker can take place when a storm is in progress. When storms occur, they last 4±2 hours. Storms can begin

at any hour of the day, except that they do not extend into the following day. When the tug is available and no storm is in progress, any berthing or de-berthing activity takes about 1 hour.

Operating experience shows that the three berths at the port are occupied about 80% of the time. On average, due to delays because of storms, unavailability of a berth, or unavailability of the tug, tanker residence time at the port exceeds the mean tanker loading time by about 5 hours. This is true for each type of tanker.

A shipper is considering bidding on a contract to transport oil from the port to the United Kingdom. He has determined that 5 tankers of a particular type would have to be committed to this task to meet contract specifications. These tankers would require 21±3 hours to load oil at the port. After loading and de-berthing, they would travel to the United Kingdom, offload the oil, return to the port for re-loading, etc. Their round-trip travel time, including offloading, is estimated to be 240±24 hours.

Before the port authorities can commit themselves to accommodating the proposed 5 tankers, the effect of the additional port traffic on the in-port residence time of the current port users must be determined. Build a GPSS model to simulate the operation of the port under the proposed new commitment. Design the model to measure in-port residence time of the proposed additional tankers, as well as the three types of tankers which already use the port. Use a 1-year simulation to estimate the distributions of these residence-time random variables.

(2) Approach Taken in Building the Model

The model is composed of a major segment, and two supporting segments. In the major segment, movement of the various tankers through the port facilities is simulated. In one of the supporting segments, the 5 additional tankers which have been proposed are brought into the model at the beginning of the simulation, and are then routed into the major segment. In the other supporting segment, which is self-contained, the daily storm conditions are simulated. Whether it is currently storming or not is communicated to the major segment through a Logic Switch. These various segments will now be commented on in more detail.

A Transaction simulating "storm potential" enters the storm segment every 24 hours. It then moves into a TRANSFER Block, where a random number determines whether there will be a storm "today" or not. If there is no storm, the Transaction immediately leaves the model. If there is to be a storm, the Transaction is first delayed 9±9 hours in an ADVANCE Block. At the end of this holding period, it puts the Logic Switch STORM into "Set" position, indicating a storm is now in progress. The Transaction is then held at another ADVANCE Block 4±2 hours, while it storms. When the storm is over, the Logic Switch STORM is put back into "Reset" position, and the Transaction leaves the model.

In the proposed-tanker initialization segment, the 5 tankers enter the model at the start of the simulation and are tagged by setting their P1 value to 4. (The other three tanker types will have P1 values of 1, 2, and 3.) They then enter an ADVANCE Block, where the first is held 48 hours, the second 96 hours, the third 144 hours, and so on. These staggered holding times are designed to "space out" the 5 tankers before routing them to the major model segment. The 48-hour "time between tankers" was determined from the 240-hour round-trip time in effect for each of the 5 tankers. After this spacing out, each tanker is assumed to have just arrived at the port (for the first time). The time-of-arrival is marked in Parameter 3, and the tanker is transferred to the major model segment.

Tanker types 1, 2, and 3 arrive at the major model segment through a GENERATE Block, are tagged in Parameter 1 with their type number, and then proceed to compete for use of the port facilities. When they are finished, they leave the model. In contrast the 5 proposed tankers never leave the model. Like the other tankers, they compete in the major segment for use of the port facilities. When they leave the port, however, they are "sieved out" of the leaving-ship stream, and are routed to an ADVANCE Block where their round-trip time is simulated. After their round trip, their time-of-arrival back at port is marked in Parameter 3, and they proceed to compete again for use of the port facilities.

The conditions required to berth and de-berth are tested in the major model segment with Boolean Variables. Three conditions must be simultaneously satisfied for berthing to occur (berth available; tug available; no storm in progress). Two conditions must be satisfied for de-berthing to take place (tug available; no storm in progress). The simultaneous occurrence of these conditions is easily tested for with Boolean Variables.

Loading time for tankers at the port is determined through each Transaction's P1 value. P1 is used as the argument for a Function MEAN, which returns the proper mean loading time. P1 is also used as the argument for a Function SPRED, which returns the loading-time spread modifier. It is not possible, though, to simulate loading time at the Block "ADVANCE FN$MEAN,FN$SPRED". If this were done, the Processor would interpret the B Operand as a Function modifier, and the returned value would be used as a multiplier of the mean. To produce the desired spread modification, each tanker-Transaction first copies the proper spread to Parameter 2. It then enters the Block "ADVANCE FN$MEAN,P2" to simulate loading time.

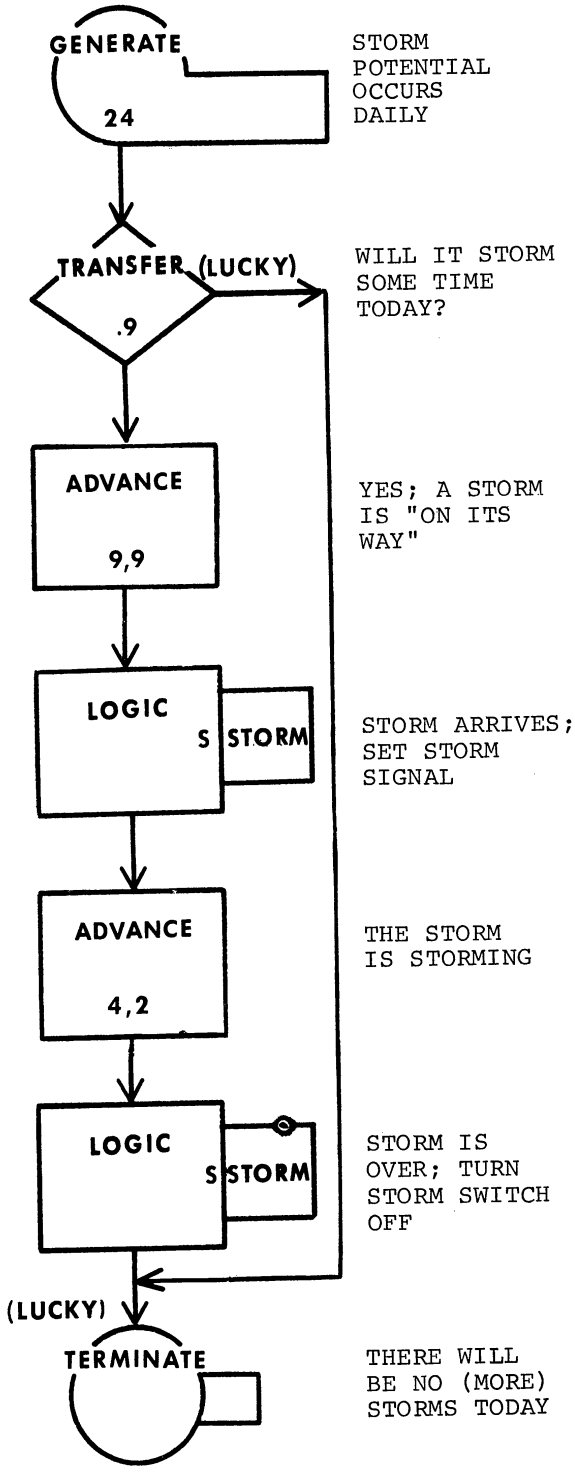
(3) Table of Definitions

Time Unit: 1 Hour

GPSS Entity	Interpretation
Transactions	
Model Segment 1	A carrier of a potential storm
Model Segment 2	One of the proposed tankers
Model Segment 3	One of the tankers using the port P1: A tanker-type code Value — Significance 1 — Type 1 tanker 2 — Type 2 tanker 3 — Type 3 tanker 4 — Proposed additional tanker P2: Unloading time spread, hours P3: Time of arrival at port (used only for Transactions simulating the proposed additional tankers)
Model Segment 4	A timer
Facilities	
TUG	The tug
Functions	
MEAN	Function describing mean loading time, depending on the type of tanker
SPRED	Function describing the loading time spread, depending on the type of tanker
TYPE	Function describing the distribution of tankers of type 1, 2, and 3
Logic Switches	
STORM	A Logic Switch which, when Set, indicates that a storm is in progress
Storages	
BERTH	A Storage simulating the loading facilities at the port
Tables	
1, 2, 3, and 4	Tables used to estimate the in-port residence time distributions for tankers of type 1, 2, and 3, and the proposed additional tankers, respectively
Variables (Arithmetic)	
SPACE	A Variable taking on the values 0, 48, 96, 144, and 192, in that chronological sequence; used to "space out" the first arrivals of the proposed additional tankers
Variables (Boolean)	
GOIN	A Variable which is true only when the berthing conditions are satisfied
GOOUT	A Variable which is true only when the de-berthing conditions are satisfied

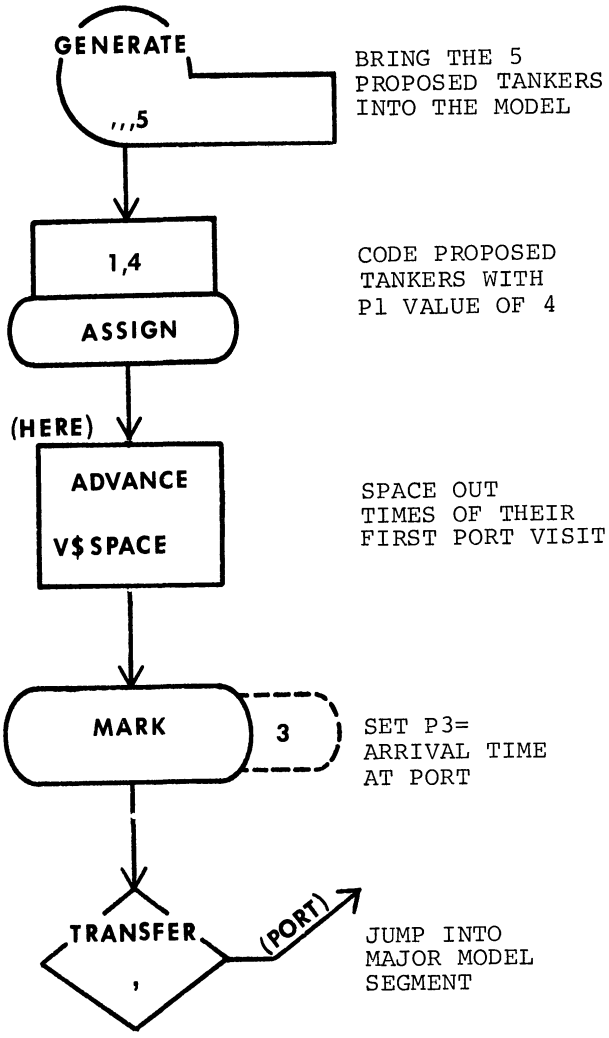
Table 5D.2 Table of Definitions for Case Study 5D

(4) Block Diagrams



MODEL SEGMENT 1

(Storm Generating Segment)

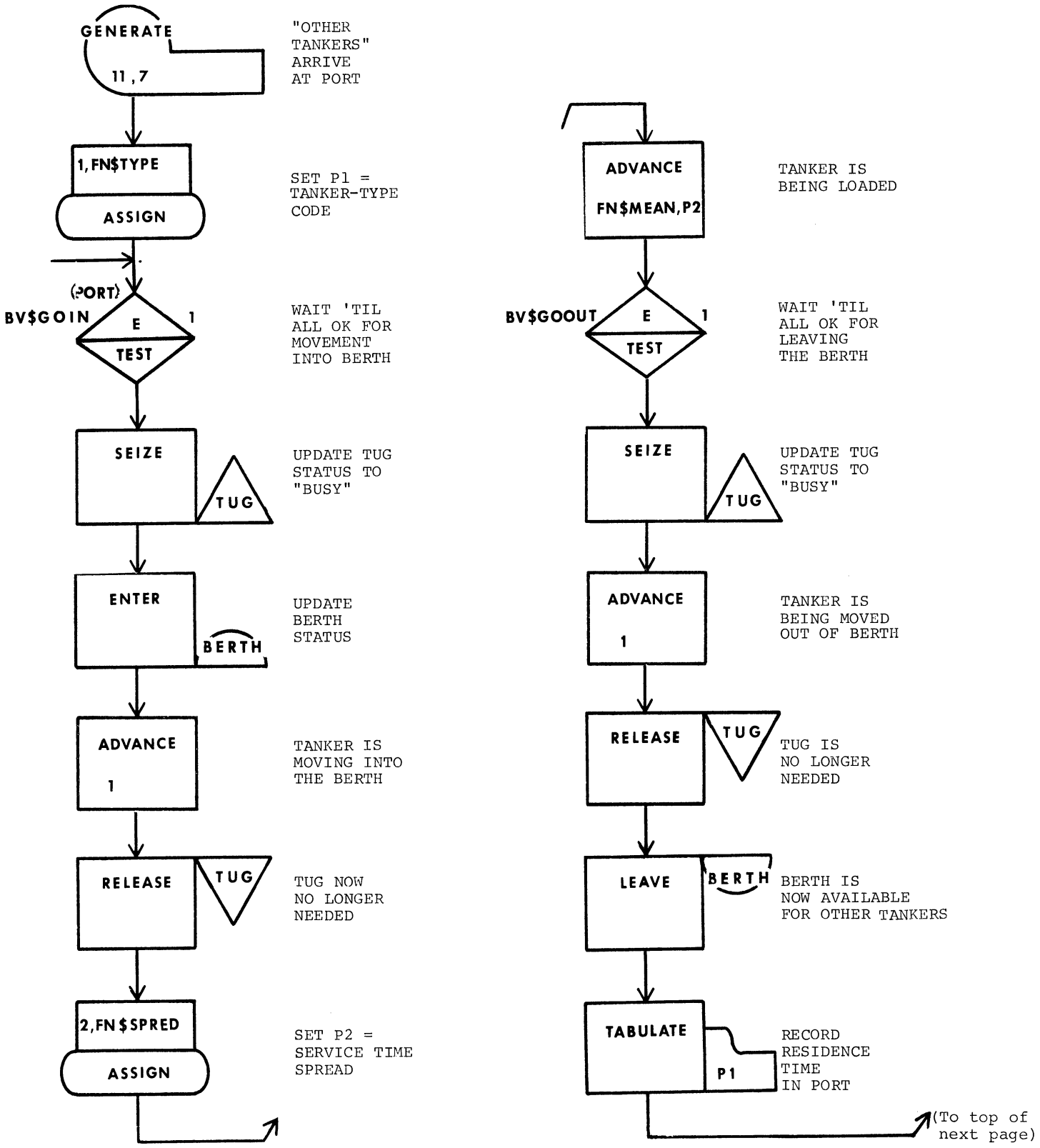


MODEL SEGMENT 2

(Initialization Segment for Proposed Tankers)

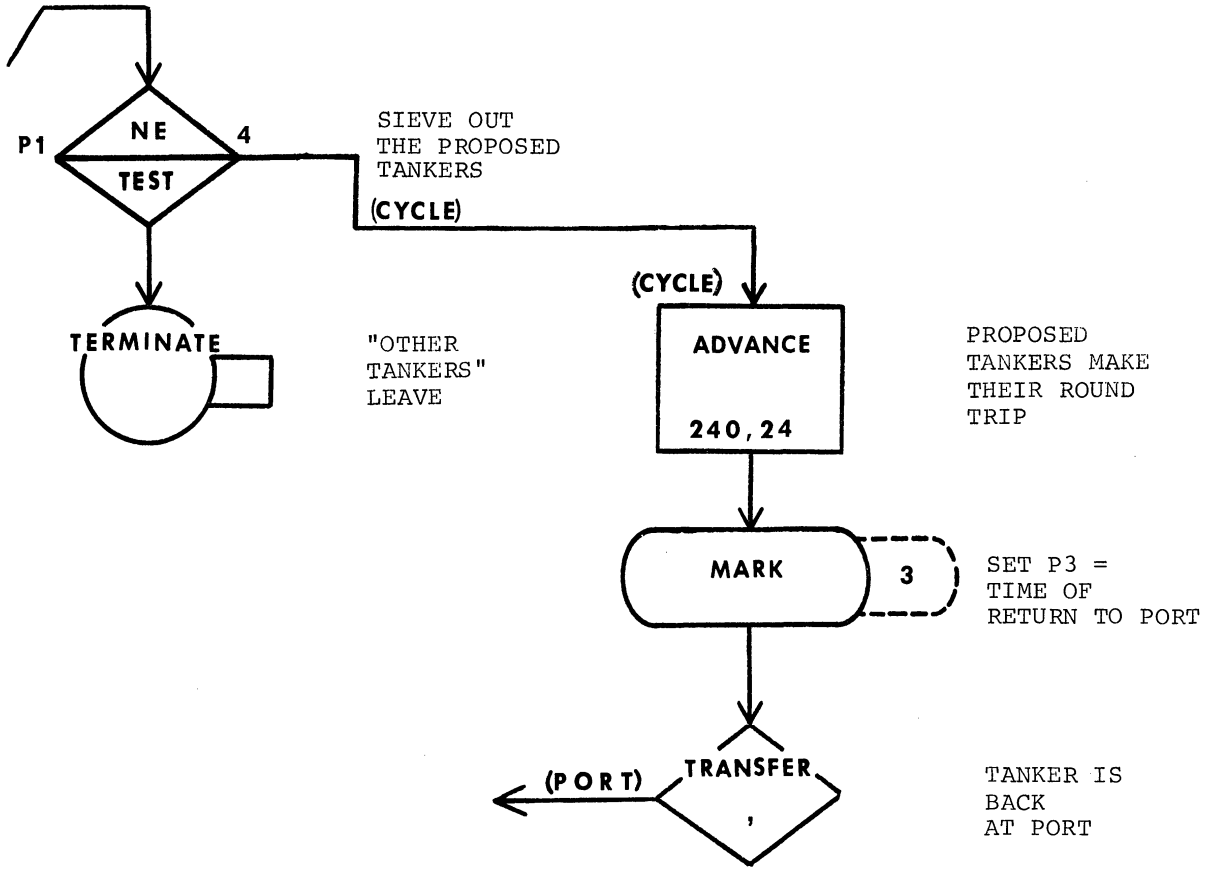
Figure 5D.1 Block Diagram for Case Study 5D

(Continued on Next Page)

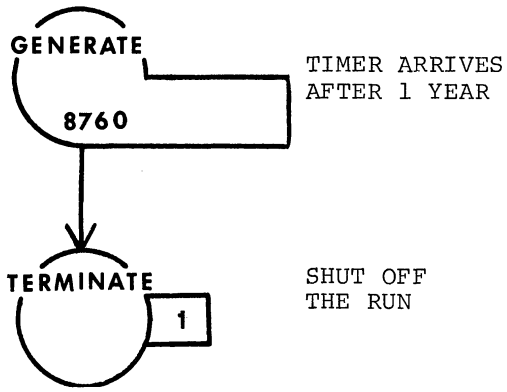


MODEL SEGMENT 3

(Port Activities; Cycling of Proposed Tankers)



MODEL SEGMENT 3 (Concluded)



MODEL SEGMENT 4

(Timer Segment)

(5) Extended Program Listing

```
BLOCK                                                                    CARD
NUMBER  *LOC   OPERATION  A,B,C,D,E,F,G            COMMENTS              NUMBER
               SIMULATE                                                     1
        *                                                                   2
        *      DEFINE FUNCTIONS                                             3
        *                                                                   4
         MEAN  FUNCTION   P1,L4       MEAN SERVICE TIME FOR TANKER TYPE "P1"  5
        1,18/2,24/3,36/4,21                                                 6
         SPRED FUNCTION   P1,L4       SERVICE TIME SPREAD FOR TANKER TYPE "P1" 7
        1,2/2,3/3,4/4,3                                                     8
         TYPE  FUNCTION   RN1,D3      DISTRIBUTION OF "OTHER TANKER" TYPES    9
        .25,1/.8,2/1,3                                                      10
        *                                                                   11
        *      DEFINE STORAGE CAPACITIES                                    12
        *                                                                   13
               STORAGE    S$BERTH,3                                         14
        *                                                                   15
        *      DEFINE TABLES                                                16
        *                                                                   17
         1     TABLE      M1,20,10,9     PORT RESIDENCE TIME, "TANKER TYPE 1"  18
         2     TABLE      M1,20,10,9     PORT RESIDENCE TIME, "TANKER TYPE 2"  19
         3     TABLE      M1,40,10,9     PORT RESIDENCE TIME, "TANKER TYPE 3"  20
         4     TABLE      MP3,20,10,9    PORT RESIDENCE TIME, PROPOSED TANKERS 21
        *                                                                   22
        *      DEFINE BOOLEAN VARIABLES                                     23
        *                                                                   24
         GOIN  BVARIABLE  SNF$BERTH*FNU$TUG*LR$STORM  BERTH-ENTERING CONDITIONS 25
         GOOUT BVARIABLE  FNU$TUG*LR$STORM            BERTH-LEAVING CONDITIONS  26
        *                                                                   27
        *      DEFINE ARITHMETIC VARIABLES                                  28
        *                                                                   29
         SPACE VARIABLE   48*N$HERE                                         30
        *                                                                   31
        *      MODEL SEGMENT 1 (STORM GENERATING SEGMENT)                   32
        *                                                                   33
1              GENERATE   24          STORM POTENTIAL OCCURS DAILY          34
2              TRANSFER   .9,,LUCKY   WILL IT STORM SOMETIME TODAY?         35
3              ADVANCE    9,9         A STORM IS "ON ITS WAY"               36
4              LOGIC S    STORM       STORM ARRIVES;  SET STORM SWITCH      37
5              ADVANCE    4,2         THE STORM IS STORMING                 38
6              LOGIC R    STORM       STORM IS OVER;  TURN STORM SWITCH OFF 39
7       LUCKY  TERMINATE              THERE WILL BE NO (MORE) STORMS TODAY  40
        *                                                                   41
        *      MODEL SEGMENT 2 (INITIALIZATION SEGMENT FOR PROPOSED TANKERS) 42
        *                                                                   43
8              GENERATE   ,,,5        BRING THE PROPOSED TANKERS INTO THE MODEL 44
9              ASSIGN     1,4         CODE PROPOSED TANKERS WITH P1 VALUE OF 4 45
10      HERE   ADVANCE    V$SPACE     SPACE OUT TIMES OF THEIR FIRST PORT VISIT 46
11             MARK       3           SET P3 = ARRIVAL TIME AT PORT         47
12             TRANSFER   ,PORT       JUMP INTO MAJOR MODEL SEGMENT         48
```

Figure 5D.2 Extended Program Listing for Case Study 5D

(Continued on Next Page)

```
                *                                                                    49
                *       MODEL SEGMENT 3 (PORT ACTIVITIES;  CYCLING OF PROPOSED TANKERS)  50
                *                                                                    51
13              GENERATE    11,7          "OTHER TANKERS" ARRIVE AT PORT             52
14              ASSIGN      1,FN$TYPE     SET P1 = TANKER-TYPE CODE                  53
15       PORT   TEST E      BV$GOIN,1     WAIT 'TIL ALL OK FOR MOVEMENT INTO BERTH   54
16              SEIZE       TUG           UPDATE TUG STATUS TO "BUSY"                55
17              ENTER       BERTH         UPDATE BERTH STATUS                        56
18              ADVANCE     1             TANKER IS BEING MOVED INTO THE BERTH       57
19              RELEASE     TUG           TUG NOW NO LONGER NEEDED                   58
20              ASSIGN      2,FN$SPRED    SET P2 = SERVICE TIME SPREAD               59
21              ADVANCE     FN$MEAN,P2    TANKER IS BEING LOADED                     60
22              TEST E      BV$GOOUT,1    WAIT 'TIL ALL OK FOR LEAVING THE BERTH     61
23              SEIZE       TUG           UPDATE TUG STATUS TO "BUSY"                62
24              ADVANCE     1             TANKER IS BEING MOVED OUT OF BERTH         63
25              RELEASE     TUG           TUG IS NO LONGER NEEDED                    64
26              LEAVE       BERTH         BERTH IS NOW AVAILABLE FOR OTHER TANKERS   65
27              TABULATE    P1            RECORD RESIDENCE TIME IN PORT              66
28              TEST NE     P1,4,CYCLE    SIEVE OUT THE PROPOSED TANKERS             67
29              TERMINATE                  "OTHER TANKERS" LEAVE                      68
30       CYCLE  ADVANCE     240,24        PROPOSED TANKERS MAKE THEIR ROUND TRIP     69
31              MARK        3             SET P3 = TIME OF RETURN TO PORT            70
32              TRANSFER    ,PORT         TANKER IS BACK AT PORT                     71
                *                                                                    72
                *       MODEL SEGMENT 4 (TIMER SEGMENT)                              73
                *                                                                    74
33              GENERATE    8760          TIMER ARRIVES AFTER 1 YEAR                 75
34              TERMINATE   1             SHUT OFF THE RUN                           76
                *                                                                    77
                *       CONTROL CARDS                                                78
                *                                                                    79
                START       1             START THE RUN                              80
                END         RETURN CONTROL TO OPERATING SYSTEM                       81
```

Figure 5D.2 Extended Program Listing for Case Study 5D

(Continued from Preceding Page)

TABLE 1

ENTRIES IN TABLE	MEAN ARGUMENT	STANDARD DEVIATION	SUM OF ARGUMENTS	
183	49.513	26.250	9061.000	NON-WEIGHTED

UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIATION FROM MEAN
20	16	8.74	8.7	91.2	.403	-1.124
30	40	21.85	30.6	69.3	.605	-.743
40	32	17.48	48.0	51.9	.807	-.362
50	18	9.83	57.9	42.0	1.009	.018
60	27	14.75	72.6	27.3	1.211	.399
70	8	4.37	77.0	22.9	1.413	.780
80	12	6.55	83.6	16.3	1.615	1.161
90	9	4.91	88.5	11.4	1.817	1.542
OVERFLOW	21	11.47	100.0	.0		
AVERAGE VALUE OF OVERFLOW		101.23				

(a) Tanker-Type 1

TABLE 2

ENTRIES IN TABLE	MEAN ARGUMENT	STANDARD DEVIATION	SUM OF ARGUMENTS	
439	54.309	26.625	23842.000	NON-WEIGHTED

UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIATION FROM MEAN
20	0	.00	.0	100.0	.368	-1.288
30	102	23.23	23.2	76.7	.552	-.913
40	64	14.57	37.8	62.1	.736	-.537
50	76	17.31	55.1	44.8	.920	-.161
60	67	15.26	70.3	29.6	1.104	.213
70	24	5.46	75.8	24.1	1.288	.589
80	18	4.10	79.9	20.0	1.473	.964
90	18	4.10	84.0	15.9	1.657	1.340
OVERFLOW	70	15.94	100.0	.0		
AVERAGE VALUE OF OVERFLOW		103.79				

(b) Tanker-Type 2

Figure 5D.3 Distribution of In-Port Residence Times, Case Study 5D

(Continued on Next Page)

TABLE 3

ENTRIES IN TABLE	MEAN ARGUMENT	STANDARD DEVIATION	SUM OF ARGUMENTS	
170	69.423	27.937	11802.000	NON-WEIGHTED

UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIATION FROM MEAN
40	28	16.47	16.4	83.5	.576	-1.053
50	22	12.94	29.4	70.5	.720	-.695
60	33	19.41	48.8	51.1	.864	-.337
70	21	12.35	61.1	38.8	1.008	.020
80	12	7.05	68.2	31.7	1.152	.378
90	12	7.05	75.2	24.7	1.296	.736
100	9	5.29	80.5	19.4	1.440	1.094
110	13	7.64	88.2	11.7	1.584	1.452
OVERFLOW	20	11.76	100.0	.0		
AVERAGE VALUE OF OVERFLOW		121.04				

(c) Tanker-Type 3

TABLE 4

ENTRIES IN TABLE	MEAN ARGUMENT	STANDARD DEVIATION	SUM OF ARGUMENTS	
151	49.887	26.000	7533.000	NON-WEIGHTED

UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIATION FROM MEAN
20	4	2.64	2.6	97.3	.400	-1.149
30	36	23.84	26.4	73.5	.601	-.764
40	32	21.19	47.6	52.3	.801	-.380
50	21	13.90	61.5	38.4	1.002	.004
60	17	11.25	72.8	27.1	1.202	.388
70	10	6.62	79.4	20.5	1.403	.773
80	4	2.64	82.1	17.8	1.603	1.158
90	10	6.62	88.7	11.2	1.804	1.542
OVERFLOW	17	11.25	100.0	.0		
AVERAGE VALUE OF OVERFLOW		103.35				

(d) Proposed Tankers

Figure 5D.3 Distribution of In-Port Residence Times, Case Study 5D
(Continued from Preceding Page)

(7) Discussion

Model Logic

Notice how the "spacing out" of first arrivals of the proposed additional tankers is accomplished in Model Segment 2. Holding time at the spacing-out ADVANCE Block (Block 10 in Figure 5D.2) is computed through the Variable SPACE as 48*N$HERE. HERE, in turn, is the Location occupied by the ADVANCE Block itself. When the first tanker-Transaction enters the Block, N$HERE is zero, so the holding time is zero. (Remember that Block Entry Counts are updated as the last step in execution of a Block's sub-routine. This means that N$HERE still has a value of 0 when the product 48*N$HERE is formed.) When the second tanker-Transaction enters the Block, N$HERE is 1 (thanks to the first tanker-Transaction), so its holding time is 48. And so on.

The ASSIGN Block in Model Segment 3 (Block 20, Figure 5D.2) is redundant to some extent. After their first visit to the harbor, Parameter 2 of the proposed additional tanker-Transactions already carries the correct value for the loading-time spread. In Problem 5.20.5, you are asked to improve on this situation, thereby eliminating a number of Block executions.

Program Output[5k]

Figure 5D.3 shows the in-port residence time Tables for the various tankers. Average in-port residence time for tanker types 1, 2, and 3 is 49.5, 54.3, and 69.4 hours, respectively (MEAN ARGUMENT in Tables 1, 2, and 3). Compared with their mean berth, load, and de-berth time-totals of 20, 26, and 38 hours, respectively (assuming no delays of any kind), their delays per port visit amount to about 30 hours. The 30-hour figure can be compared with the 3-hour delays these tankers experience when they do not have to compete with the additional tankers (as indicated in the problem statement). From model output not shown, berth occupancy has jumped to about 96% with the additional tankers in use, as contrasted with about an 80% occupancy previously (as reported in the problem statement).

The proposed additional tankers spend about 2 days (49.9 hours, Table 4 in Figure 5D.3) in port on average. With no delays, their berth, load, and de-berth time would amount to 23 hours on average. Hence, they experience about a 27-hour delay per visit to the port. Problem 5.20.6 involves further investigation of these various tanker delays.

[5k] Total CPU time for the simulation was 21.6 seconds.

5.20 Exercises

5.20.1 Show definitions for Boolean Variables which are true under the following conditions. Also define any Arithmetic Variables you may use as Boolean Variable components.

(a) Queue 7 is not empty and Queue 6 is empty.

(b) The Storage TUG is full or the Queue SHIP has a content greater than 3.

(c) The sum of the contents of Queues 1 and 2 is less than the content of Queue 3.

(d) The Boolean Variable defined with the expression "Q1+Q2'L'Q3" has been proposed as a solution to (c). Why will the expression result in an error condition? What would you do to eliminate the error condition?

(e) The standard deviation in the Table LTD is less than 10% of the Table mean, the number of entries in the Table exceeds 250, and the relative clock is an integral multiple of 24.

(f) The Queue ONE is empty, and the Queue TWO is not empty or the Storage THREE has a remaining capacity less than two.

(g) The Logic Switch AMBER is Set, or the Storage CROSS is full, or the Savevalue RTIME is zero in value.

5.20.2 A Transaction is to move beyond a certain point in a model only if the Logic Switch BETA is Set, and the Storage MEN has a remaining capacity of 3 or more. An analyst proposes using the Block Diagram segment shown in Figure P5.20.2 to accomplish this effect. What is wrong with his approach?

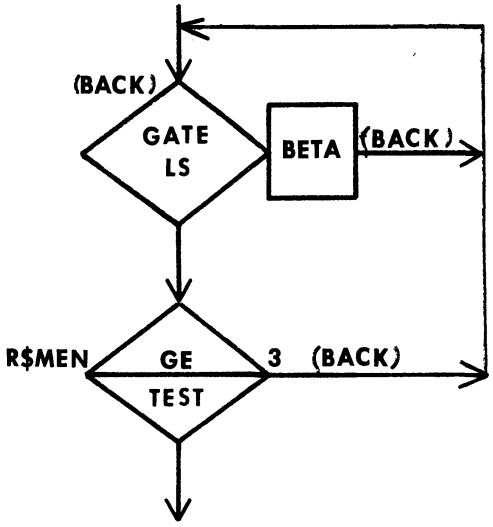


Figure P5.20.2

What will probably eventually happen if the corresponding model is run? What would you do to accomplish the same effect, and yet eliminate the flaw in the analyst's logic?

5.20.3 Show a Block Diagram segment delaying Transactions in Queue 7 until Facility 5 is not in use, Storage 3 is not full, and Logic Switch 2 is Reset.

5.20.4 A Transaction is to be sent to whichever one of the Facilities JOE and PETE is available. If both Facilities are either idle or busy, the Transaction is to go to the one whose utilization is the smaller of the

two. Show a Boolean Variable and the corresponding TEST Block at which it can be evaluated to accomplish this routing.

5.20.5 These questions are based on Case Study 5D.

(a) In Figure 5D.2, suppose the Location Name HERE were placed on Block 9, instead of Block 10. Show how to re-define the Variable SPACE so that it still takes on the values 0, 48, 96, etc., in that chronological sequence.

(b) Show how to modify the Figure 5D.2 model to eliminate the partial redundancy represented by the presence of the ASSIGN Block in Model Segment 2 (Block 20).

(c) Through the Storage BERTH, the Figure 5D.2 model measures the fraction of the time that the berths are occupied. The fact that a berth is occupied, however, does not necessarily mean that it is being used to load oil onto a tanker. Show how to modify the model so that it measures the fraction of the time that the berths are actually used for oil-loading purposes.

5.20.6 In Case Study 5D, assume that the increased in-port residence time for tankers of type 1, 2, and 3 resulting from use of the port by the 5 additional tankers is unacceptably high. The owners of the port want estimates of the various in-port residence time distributions for each of the following alternatives.

(a) The 5 additional tankers are given low priority at the port relative to the other type tankers. (In addition to the changed in-port residence times, to what extent does this alternative reduce the rate at which the 5 additional tankers can transport oil to the United Kingdom?)

(b) The additional tankers are given low priority at the port, and the number of proposed additional tankers is increased from 5 to 6. The 6th tanker is assumed to be identical to the other 5. (Besides changing in-port residence times, by how much does this alternative increase the rate at which the "6 additional tankers" can transport oil to the United Kingdom?)

(c) The various tankers have equal priority at the port, but the number of loading berths is increased from 3 to 4.

Make the necessary modifications to Case Study 5D to reflect these alternatives, then run the modified models to provide the requested information.

Appendix A

Sources of Information About GPSS from IBM[Aa]

Title of Publication	Form Number	Price[Ab]
360 GPSS Application Description	GH20-0186	
360 GPSS Introductory Users Manual	GH20-0304	
360 GPSS Operations Manual	GH20-0311	
360 GPSS Users Manual	GH20-0326	$2.30
360 GPSS DOS Operators Manual	GH20-0327	
360 OS GPSS Application Description	GH20-0691	
360 OS GPSS Operations Manual	SH20-0692	$2.10
360 OS GPSS Introductory Users Manual	SH20-0693	$2.90
360 OS GPSS Version 2 Users Manual	SH20-0694	$8.40
360 DOS GPSS Version 2 Operations Manual	SH20-0698	$2.10
GPSS V OS Application Description Manual	GH20-0825	
GPSS V DOS Application Description Manual	GH20-0826	
GPSS V OS/DOS Users Manual	SH20-0851	$17.50
GPSS V OS Operations Manual	SH20-0867	$3.80
GPSS V DOS Operations Manual	SH20-0868	$3.00
GPSS V OS Program Product Specifications	GH20-4035	

[Aa] Copies of these publications can be obtained through IBM offices. Single copies of the publications having Form Numbers beginning with GH are usually available on a complimentary basis.

[Ab] When no price is shown, the cost is under $1.

Appendix B[Ba]

Information Pertinent to Computer Memory Considerations in GPSS

Entity Type[Bb]	Normally Available Quantities for Various Computer Memory Levels 64K	128K	256K	Fixed Memory Requirement per Item, Bytes	Reallocation Mnemonic
Transactions	200	500	1200	16[a]	XAC
Blocks	120	500	1000	12[b]	BLO
Facilities	35	150	1000	28	FAC
Logic Switches	200	400	1000	6	LOG
Queues	70	150	300	32	QUE
Storages	35	150	300	40	STO
Savevalues (fullword)	100	400	1000	4	FSV
Savevalues (halfword)	50	200	500	4	HSV
Tables	15	30	100	48[d]	TAB
Variables (Arithmetic)	20	50	200	48[e]	VAR
Variables (Boolean)	5	10	25	32[f]	BVR
COMMON (bytes)[g]	5600	14400	25600		COM

[a] Add 20 bytes of COMMON for every active Transaction, plus additional bytes for Parameters (2 bytes per halfword Parameter, 4 bytes per fullword Parameter)

[b] Add 4 bytes of COMMON for each Block with more than one Operand used

[c] Add 4 bytes of COMMON for each point of an L type Function.
Add 8 bytes of COMMON for each pair of coordinates of a D type Function.
Add 12 bytes of COMMON for each pair of coordinates of a C type Function.

(d) Add 4 bytes of COMMON for each Table interval.

[e] Add 12 bytes of COMMON for each Standard Numerical Attribute in the defining expression.
Add 20 bytes of COMMON for each pair of parentheses used.
Add 2 bytes of COMMON for each arithmetic operator used. (This is an approximation.)

[f] Add 12 bytes of COMMON for each Logical Operator used.
Add 24 bytes of COMMON for each pair of parentheses used. (This is an approximation.)
Add 8 bytes of COMMON for the first Boolean Operator used.
Add 4 bytes of COMMON for each Boolean Operator used after the first.
Add 16 bytes of COMMON for each Relational Operator used.
Add 4 bytes of COMMON for each constant used.
Add 12 bytes of COMMON for each non-constant Standard Numerical Attribute used.
Add 4 additional bytes of COMMON

[g] COMMON is an otherwise-uncommitted pool of computer memory, portions of which can be used for such purposes as toring the pairs of points used to define Functions, providing the memory needed for Transaction Parameters, and so on.

[Ba] For a discussion of the significance and use of the information in Appendix B, and computational examples, see Section 4.13.

[Bb] Only those GPSS entities actually discussed in this advanced printing are included in Appendix B.

Appendix C

GPSS Error Messages

There are 5 categories of "Error Messages" that can arise in GPSS. They are shown in Table C.1. Table C.1 also shows the range of Error Message numbers corresponding to each category, and indicates the pages in this appendix where interpretations are given for the various error message numbers.

Category	Designation	Range of Error Numbers	Pages Showing Interpretations
1	Assembly Program Errors	1 - 81	C2-C3
2	Input Errors	201 - 302	C4-C5
3	Execution Errors	401 - 699	C6-C7
4	Output Errors	702 - 769	C8
5	Warning Messages	850 - 863	C8

Table C.1

Assembly Program errors can arise when the Processor takes its first pass over the model, but before the "input phase" (as explained on page II-54) begins. The corresponding types of errors usually involve simple violations of GPSS syntax or entity range limitations. For example, error 43, ILLEGAL SYMBOL (TOO LONG), would occur if a Block's Symbolic Location Name consisted of more than 5 characters. Similarly, error 32, ILLEGAL FACILITY NUMBER, would arise if the model was being run at the 64K memory level and the analyst used a value of 36 or greater as a Facility number, but had not reallocated the maximum quantity of Facilities available. (The maximum quantity of Facilities under the indicated conditions is 35. As described on page II-22, the largest permissible Facility number equals the maximum number of different Facilities allowable in a model.)

Input errors can occur during the "input phase" when the Processor, among other things, pre-schedules Transaction arrivals at GENERATE Blocks for the first time. A typical error condition that could come about in this category would be 216, MODIFIER EXCEEDS MEAN. This indicates that a GENERATE Block's Spread Modifier is larger that the mean inter-arrival time, which can cause a negative inter-arrival time to be computed.

Execution Errors can arise while the simulation is proceeding, but before the Termination Counter has been decremented to zero. A typical message in this category is number 406, ILLEGAL PRIORITY COMPUTED FOR XACT BEING CREATED AT A 'GENERATE' BLOCK. (XACT is an abbreviation for "Transaction".) This condition could occur if a GENERATE Block's E Operand were indirectly specified, and the value turned out to be less than 0, or larger than 127. (The only permissible Priority Levels are 0, 1, 2, 3, ... , 125, 126, and 127.)

Output Errors can come about after the simulation has shut off, while the Processor is outputting the statistics developed during the simulation. If such an error arises, it is very likely because the Processor is attempting to output statistics for an entity which has an illegal number, i.e., is outside of the permissible range.

Finally, Warning Messages are printed out when circumstances come about which are not "fatal" (i.e., do not cause the run to be aborted), but nevertheless should probably be called to the analyst's attention. For example, warning message 850 is ATTEMPT TO STORE INTEGER OF MAGNITUDE GREATER THAN 32767 IN A HALFWORD PARAMETER. The largest value that can be held in a halfword Parameter is 32,767. If an attempt is made to store a value larger than this in such a Parameter, the value actually stored is "MAINTAINED MODULO 2**15". Consequently, the stored value equals the remainder that would result if the too-large value were divided by 2**15 (i.e., divided by 32,768). This means, for example, that if the Block "ASSIGN 3,X$BIG" is executed, and the value of X$BIG is 43,562, the value stored in Parameter 3 will be 10,794, assuming the Transaction's Parameters are of the halfword type.

ASSEMBLY PROGRAM ERRORS

1 ILLEGAL SELECTION MODE SPECIFIED IN A-FIELD OF 'TRANSFER' BLOCK

2 ILLEGAL OPERATION FIELD

3 ENTITY NUMBER TO BE RESERVED BY AN 'EQU' CARD HAS BEEN RESERVED BY A PREVIOUS 'EQU' CARD

4 BLOCK SYMBOL HAS BEEN USED IN AN 'EQU' CARD

5 ILLEGAL TABLE ARGUMENT

6 FRACTIONAL SELECTION MODE IS MORE THAN A 3-DIGIT NUMBER

7 SYNTAX ERROR IN A 'MATRIX' OR 'EQU' CARD

8 ILLEGAL ENTITY INDICATOR

9 A-FIELD OF 'ASSIGN' BLOCK IS GREATER THAN 100

10 FIRST OPERAND IN A 'MATRIX' CARD IS NOT 'X' OR 'H'

11 UNDEFINED BLOCK SYMBOL

12 ILLEGAL JOBTAPE SPECIFIED

13 THE NUMBER OF ROWS AND/OR COLUMNS SPECIFIED IN A 'MATRIX' CARD IS NOT CONSTANT

14 FIELD-E OF 'MSAVEVALUE' BLOCK IS ILLEGAL

15 'TRANSFER' BLOCK WITH 'ALL' OR 'PICK' SELECTION MODE CONTAINS A-FIELD WHOSE VALUE IS LESS THAN B-FIELD

16 'TRANSFER' BLOCK WITH 'ALL' SELECTION MODE HAS A B-FIELD TO C-FIELD RANGE WHICH IS NOT EVENLY DIVISIBLE BY THE D-FIELD

17 CARD WHICH MUST HAVE ENTRY IN LOCATION FIELD DOES NOT

18 ILLEGAL HALFWORD MATRIX SAVEVALUE

19 ILLEGAL MNEMONIC SPECIFIED IN OPERATION FIELD OF 'GATE', 'LOGIC', 'TEST', 'COUNT', OR 'SELECT' BLOCK

20 ILLEGAL B-FIELD IN 'PRIORITY' BLOCK

21 SYMBOL IN ENTITY FUNCTION FOLLOWER CARD HAS BEEN USED IN AN 'EQU' CARD OR HAS BEEN USED AS A BLOCK SYMBOL OR HAS BEEN USED IN A PREVIOUS ENTITY FUNCTION

22 ILLEGAL BOOLEAN VARIABLE NUMBER

23 ILLEGAL REPORT TYPE SPECIFIED IN 'REPORT' CARD

24 CARD TYPE IS NOT PERMITTED WITHIN THE REPORT TYPE SPECIFIED

25 STORAGE DEFINED WITH CAPACITY GREATER THAN THE MAXIMUM PERMISSIBLE VALUE

26 TABLE MUST BE SPECIFIED NUMERICALLY

27 ILLEGAL SYMBOL

28 ILLEGAL FUNCTION TYPE

29 MODIFIER OF 'GENERATE' OR 'ADVANCE' BLOCK EXCEEDS MEAN

30 A-FIELD OMITTED WHERE IT MUST BE SPECIFIED

31 B-FIELD OMITTED WHERE IT MUST BE SPECIFIED

32 ILLEGAL FACILITY NUMBER

33 ILLEGAL STORAGE NUMBER

34 ILLEGAL QUEUE NUMBER

35 ILLEGAL LOGIC SWITCH NUMBER

36 ILLEGAL CHAIN NUMBER

37 ILLEGAL TABLE NUMBER

38 ILLEGAL VARIABLE NUMBER

39 ILLEGAL SAVEVALUE NUMBER

40 ILLEGAL FUNCTION NUMBER

ASSEMBLY PROGRAM ERRORS (CONTINUED)

41 ILLEGAL SYMBOL IN LOCATION FIELD OR NO SYMBOL WHERE ONE IS REQUIRED

42 ILLEGAL GROUP NUMBER

43 ILLEGAL SYMBOL (TOO LONG)

44 SYNTAX ERROR IN CARD

45 ILLEGAL SNA

46 C-FIELD OMITTED WHERE IT MUST BE SPECIFIED

47 ILLEGAL MATRIX SAVEVALUE NUMBER

48 D-FIELD OMITTED WHERE IT MUST BE SPECIFIED

49 MAXIMUM NUMBER OF MACRO DEFINITIONS EXCEEDED

50 UNDEFINED MACRO

51 ILLEGAL MACRO ARGUMENT - ARGUMENT MUST BE ALPHABETIC A-J

52 MACRO CARD EXPANDED PAST COLUMN 72

53 MORE THAN 2 MACROS NESTED WITHIN A MACRO

54 MORE THAN 10 ARGUMENTS SPECIFIED IN ABOVE MACRO CARD

55 C-FIELD OF 'SAVEVALUE' BLOCK IS ILLEGAL

56 ILLEGAL HALFWORD SAVEVALUE

57 NO LEGAL ENTITY NUMBER LEFT TO BE ASSIGNED TO ENTITY SYMBOL

58 OPERAND FIELD EXTENDS INTO COLUMN 72

59 THERE ARE MORE RIGHT PARENS THAN LEFT PARENS IN A 'VARIABLE' CARD

60 THERE ARE MORE LEFT PARENS THAN RIGHT PARENS IN A 'VARIABLE'

61 IMPOSSIBLE MODULO DIVISION SPECIFIED IN A 'VARIABLE' CARD

62 E-FIELD OMITTED OR ILLEGAL WHERE IT MUST BE SPECIFIED

63 'EQU' CARD OR ENTITY FUNCTION SPECIFIES THAT ILLEGAL ENTITY NUMBER BE RESERVED

64 GRAPH CARDS OUT OF ORDER

65 ILLEGAL A-FIELD IN 'STATEMENT' CARD

66 ILLEGAL ROW REQUEST IN 'STATEMENT' CARD

67 ILLEGAL B-FIELD IN 'STATEMENT' CARD

68 TOO MANY COLUMNS REQUESTED IN 'STATEMENT' CARD

69 DECREASING ROW NUMBERS REQUESTED IN 'STATEMENT' CARD

70 ILLEGAL STARTING COLUMN FOR STATEMENT

71 ILLEGAL SNA REQUESTED IN 'GRAPH' CARD

72 ILLEGAL ENTITY RANGE IN 'GRAPH' CARD

73 ILLEGAL REQUEST IN 'ORIGIN' CARD

74 ILLEGAL ENTITY REQUESTED IN 'TITLE' CARD

75 ILLEGAL FIELD IN 'X' CARD

76 ILLEGAL NUMERIC FIELD IN 'X' OR 'Y' CARD

77 ILLEGAL REQUEST IN 'Y' CARD

78 NUMBER OF POINTS IN FUNCTIONS DOES NOT AGREE WITH NUMBER SPECIFIED ON FUNCTION HEADER CARD

79 CONSTANT WITH MORE THAN 6 DIGITS SPECIFIED WHERE NOT PERMITTED

81 SIGNED CONSTANT IN FIELD WHERE NOT PERMITTED

201 NUMBER OF TRANSACTIONS EXCEEDED

202 REFERENCED TRANSACTION NOT INACTIVE

203 PRIORITY EXCEEDS 127

205 NUMBER OF PARAMETERS EXCEEDS 100

206 'GENERATE' BLOCK : F-FIELD MUST BE 'F', 'H', OR BLANK

207 'PREEMPT' BLOCK : B-FIELD MUST BE 'PR'

208 'PREEMPT' BLOCK : E-FIELD MUST BE 'RE'

209 'PREEMPT' BLOCK : C-FIELD NOT SPECIFIED WITH D- AND/OR C-FIELDS

210 ILLEGAL MNEMONIC IN OPERATION FIELD

211 ILLEGAL STORAGE NUMBER

212 D-FIELD NOT NECESSARY IF 'MAX' OR 'MIN' MODE SPECIFIED

213 ILLEGAL MNEMONIC IN C-FIELD OF 'PRINT' BLOCK

214 ILLEGAL FORMAT IN LOGIC SWITCH 'INITIAL' CARD

215 AMOUNT OF GPSS/360 COMMON CORE EXCEEDED
(See Section 4.13, page IV-48.)

216 MODIFIER EXCEEDS MEAN

217 ACTION TIME NOT GREATER THAN OR EQUAL TO ZERO

218 ILLEGAL FULLWORD MATRIX NUMBER

219 ILLEGAL HALFWORD MATRIX NUMBER

220 ILLEGAL FORMAT FOR 'TRANSFER-ALL'

221 ILLEGAL TABLE NUMBER

222 ILLEGAL FUNCTION NUMBER

223 FUNCTION 'X' VALUES NOT IN ASCENDING ORDER

224 'UNLINK' BLOCK : E-FIELD MUST BE BLANK IF 'BACK' SPECIFIED

225 ILLEGAL FULLWORD SAVEVALUE NUMBER

226 ILLEGAL HALFWORD SAVEVALUE NUMBER

227 ILLEGAL FORMAT IN SAVEVALUE 'INITIAL' CARD

228 MNEMONIC OTHER THAN 'X' OR 'H' USED IN SAVEVALUE 'INITIAL' CARD

229 FIRST INDEX HIGHER THAN SECOND IN MULTI-INITIALLIZATION

230 ILLEGAL LOGIC SWITCH NUMBER

231 'VARIABLE' DEFINITION CARD : COLUMN 18 NOT BLANK

232 'VARIABLE' DEFINITION CARD : ILLEGAL VARIABLE NUMBER

233 'VARIABLE' DEFINITION CARD : NUMBER STATED INCORRECTLY

234 'VARIABLE' DEFINITION CARD : IMPROPER NUMBER OF PARENS

235 'VARIABLE' DEFINITION CARD : TOO MANY SETS OF PARENS

236 'VARIABLE' DEFINITION CARD : IMPOSSIBLE MODULO DIVISION

237 'VARIABLE' DEFINITION CARD : ILLEGAL BOOLEAN VARIABLE NUMBER

238 'VARIABLE' DEFINITION CARD : MODULO DIVISION IN FVARIABLE

239 NO COMMA IN 'MATRIX' STATEMENT

240 ILLEGAL BOOLEAN OPERATOR

241 ILLEGAL STATEMENT OF OPERATION IN VARIABLE

242 ILLEGAL SNA IN 'VARIABLE' STATEMENT

243 ILLEGAL MATRIX ROW NUMBER

244 ILLEGAL MATRIX COLUMN NUMBER

245 ILLEGAL MATRIX SAVEVALUE MNEMONIC

246 ILLEGAL FORMAT IN MATRIX 'INITIAL' CARD

247 ILLEGAL FORMAT IN SAVEVALUE 'INITIAL' CARD

248 ILLEGAL HALFWORD SAVEVALUE

249 TOO MANY NUMERIC DIGITS IN A CONSTANT

250 ILLEGAL SNA MNEMONIC

INPUT ERRORS (CONTINUED)

251 MISSING OPERATOR IN VARIABLE

252 E-FIELD NOT BLANK WHEN 'BV' SPECIFIED IN 'UNLINK' BLOCK

253 ILLEGAL MNEMONIC IN A-FIELD OF 'TRANSFER' BLOCK

254 FRACTION IN 'TRANSFER' BLOCK A-FIELD NOT 3 DIGITS

255 MATRIX 'INITIAL' CARD : ILLEGAL INDEX FOR ROWS

256 MATRIX 'INITIAL' CARD : ILLEGAL INDEX FOR COLUMNS

257 ILLEGAL QUEUE NUMBER

258 ILLEGAL JOBTAPE NUMBER

259 CYCLIC (RECURSIVE) DEFINITION OF A VARIABLE

260 VARIABLE NOT DEFINED

261 ILLEGAL VARIABLE NUMBER

262 CYCLIC (RECURSIVE) DEFINITION OF A FUNCTION

263 ILLEGAL FUNCTION NUMBER

264 UNDEFINED FUNCTION

265 ILLEGAL FUNCTION TYPE

266 A FUNCTION MUST HAVE MORE THAN ONE POINT

267 ILLEGAL TYPE SPECIFIED IN B-FIELD OF A 'FUNCTION' CARD

270 NO C-FIELD IN 'EXAMINE' BLOCK

271 ILLEGAL ENTITY NUMBER IN 'RESET' CARD

272 ILLEGAL ENTITY TYPE REQUESTED ON 'RESET' CARD

273 SEQUENCE ERROR ON 'RESET' CARD

274 ILLEGAL REQUEST ON SELECTIVE 'CLEAR' CARD

275 ILLEGAL SAVEVALUE NUMBER ON 'CLEAR' CARD

276 ILLEGAL RANGE OF SAVEVALUES ON 'CLEAR' CARD

277 ILLEGAL HALFWORD SAVEVALUE ON 'CLEAR' CARD

278 ILLEGAL RANGE OF HALFWORD SAVEVALUES ON A 'CLEAR' CARD

279 'READ/SAVE' IDENTIFIER NOT FOUND ON SPECIFIED 'READ' DEVICE

280 ILLEGAL ALLOCATION OF ENTITIES ON 'READ' FILE OR DEVICE

282 ERROR IN BLOCK REDEFINITION

283 ILLEGAL BLOCK NUMBER

284 ILLEGAL XACT REFERENCED IN A CHAINING ROUTINE (PROBABLE SYSTEM BUG)

285 ILLEGAL FREQUENCY CLASS DESIGNATION ON 'TABLE' DEFINITION CARD

290 ILLEGAL REFERENCE TO GPSS/360 COMMON (PROBABLE SYSTEM BUG)

291 ILLEGAL SNA REFERENCED (PROBABLE SYSTEM BUG)

293 ILLEGAL SNA REFERENCED (PROBABLE SYSTEM BUG)

297 ILLEGAL RANDOM NUMBER GENERATOR REFERENCED

298 ATTEMPT TO READ PAST END-OF-FILE ON READ/SAVE FILE OR DEVICE

299 ILLEGAL 'RMULT' CARD SPECIFICATION

301 FUNCTION 'Y' VALUES ARE NOT CONSTANT

302 ILLEGAL TABLE ARGUMENT

EXECUTION ERRORS

401 NO NEW EVENT IN THE SYSTEM

402 ILLEGAL XACT ON THE FUTURE EVENTS CHAIN (PROBABLE SYSTEM BUG)

403 ILLEGAL XACT IN THE FUTURE EVENTS CHAIN (PROBABLE SYSTEM BUG)

404 ILLEGAL XACT IN THE FUTURE EVENTS CHAIN (PROBABLE SYSTEM BUG)

405 NUMBER OF PARAMETERS EXCEEDED
AN XACT CAN HAVE NO MORE THAN 100 PARAMETERS

406 ILLEGAL PRIORITY COMPUTED FOR XACT BEING CREATED AT A 'GENERATE' BLOCK

406 PRIORITIES MUST BE NON-NEGATIVE AND LESS THAN 128

413 ILLEGAL ENTRY TO A GENERATE BLOCK

415 FACILITY RELEASED BY AN XACT WHICH DID NOT SEIZE IT

416 FACILITY RELEASED BY AN XACT WHICH DID NOT SEIZE IT

417 INTERRUPT COUNT IS NEGATIVE (PROBABLE SYSTEM BUG)

421 FACILITY RETURNED BY AN XACT WHICH IS NOT PREEMPTING IT

425 XACT AT A 'LEAVE' BLOCK DECREMENTS STORAGE CONTENTS TO LESS THAN ZERO

428 XACT AT A 'DEPART' BLOCK DECREMENTS QUEUE CONTENTS TO LESS THAN ZERO

429 PARAMETER SPECIFIED IN A-FIELD OF A 'LOOP' BLOCK IS ZERO AS AN XACT ENTERS THE BLOCK

432 ILLEGAL HALFWORD SAVEVALUE NUMBER

433 ILLEGAL FULLWORD SAVEVALUE NUMBER

434 WEIGHTING NOT SPECIFIED IN D-FIELD OF 'TABLE' DEFINITION CARD FOR THE TABLE BEING REFERENCED IN THE CURRENT BLOCK

435 ILLEGAL 'TABLE' NUMBER

436 'TABLE' NOT DEFINED BY A 'TABLE' CARD

437 ILLEGAL XACT NUMBER REFERRED TO UNDER A BLOCKED CONDITION (PROBABLE SYSTEM BUG)

438 ATTEMPT TO PLACE AN XACT ON A DELAY CHAIN WHEN THE XACT IS ALREADY ON A DELAY CHAIN (PROBABLE SYSTEM BUG)

442 NO PREEMPT COUNT IN XACT RETURNED FROM PREEMPT CONDITION (PROBABLE SYSTEM BUG)

443 NUMBER OF XACTS EXCEEDED (See Section 4.13, page IV-48.)

453 ATTEMPT TO REMOVE XACT FROM ILLEGAL CHAIN (PROBABLE SYSTEM BUG)

463 ATTEMPT TO REMOVE XACT FROM ILLEGAL CHAIN (PROBABLE SYSTEM BUG)

466 ATTEMPT TO REMOVE XACT FROM ILLEGAL CHAIN (PROBABLE SYSTEM BUG)

467 ATTEMPT TO REMOVE XACT FROM ILLEGAL CHAIN (PROBABLE SYSTEM BUG)

468 NUMBER OF XACTS EXCEEDED(See Section 4.13, page IV-48.)

469 NUMBER OF XACTS EXCEEDED(See Section 4.13, page IV-48.)

470 ILLEGAL XACT NUMBER REFERENCED (PROBABLE SYSTEM BUG)

471 ILLEGAL XACT NUMBER REFERENCED (PROBABLE SYSTEM BUG)

472 ILLEGAL XACT NUMBER REFERENCED (PROBABLE SYSTEM BUG)

474 PREEMPT COUNT EXCEEDS 127

475 ILLEGAL XACT NUMBER REFERENCED (PROBABLE SYSTEM BUG)

476 ATTEMPT TO REMOVE AN INTERRUPT ON AN XACT WHICH HAS NOT BEEN INTERRUPTED (PROBABLE SYSTEM BUG)

477 ATTEMPT TO REMOVE AN INTERRUPT ON AN XACT WHICH HAS NOT BEEN INTERRUPTED (PROBABLE SYSTEM BUG)

478 AN XACT PREVIOUSLY TERMINATED WHILE SEIZING OR PREEMPTING A FACILITY REFERRED TO AT THE CURRENT BLOCK

479 AN XACT PREVIOUSLY TERMINATED WHILE SEIZING OR PREEMPTING A FACILITY REFERRED TO AT THE CURRENT BLOCK

480 ILLEGAL RANDOM NUMBER GENERATOR REFERENCED

492 ILLEGAL XACT PARAMETER NUMBER

497 ILLEGAL USER CHAIN REFERENCED

498 ILLEGAL FACILITY NUMBER REFERENCED

499 ILLEGAL STORAGE NUMBER REFERENCED

500 ILLEGAL QUEUE NUMBER REFERENCED

501 ILLEGAL LOGIC SWITCH NUMBER REFERENCED

505 NEGATIVE TIME DELAY COMPUTATION ('ADVANCE' OR 'GENERATE' BLOCK)

506 CYCLIC (RECURSIVE) FUNCTION EVALUATION

507 ILLEGAL FUNCTION NUMBER

508 FUNCTION NOT DEFINED BY A 'FUNCTION' CARD

509 ILLEGAL INDEX EVALUATED FOR A LIST TYPE FUNCTION

512 ENTERING UNDEFINED BLOCK

514 ILLEGAL VARIABLE NUMBER

515 ARITHMETIC VARIABLE NOT DEFINED BY A 'VARIABLE' CARD

516 CYCLIC (RECURSIVE) EVALUATION OF ARITHMETIC VARIABLE

518 TOO MANY LEVELS OF INTERRUPT

530 SPREAD EXCEEDS MEAN IN TIME-DELAY COMPUTATION ('ADVANCE' OR 'GENERATE' BLOCK)

560 ILLEGAL MATRIX SAVEVALUE

561 ILLEGAL ROW IN 'MSAVEVALUE' BLOCK

562 ILLEGAL COLUMN SPECIFIED IN MSAVEVALUE' CARD

599 LIMITS OF GPSS/360 COMMON CORE REGION EXCEEDED
(See Section 4.13, page IV-48.)

601 NEXT SEQUENTIAL BLOCK NUMBER IS ILLEGAL

603 ILLEGAL BLOCK NUMBER

604 ILLEGAL TABLE ARGUMENT

607 A-FIELD ASSEMBLY OR GATHER COUNT IS ZERO AT 'ASSEMBLE' OR 'GATHER' BLOCKS

609 XACT WHICH IS THE ONLY MEMBER OF IT'S ASSEMBLY SET IS AT A 'MATCH', 'ASSEMBLE', 'GATHER', 'GATE-M', OR 'GATE-NM' BLOCK

610 UPPER LIMIT LESS THAN LOWER LIMIT IN 'COUNT' OR 'SELECT' BLOCK

611 ILLEGAL XACT PARAMETER NUMBER REFERENCED

612 ILLEGAL BLOCK NUMBER REFERENCED IN 'TRANSFER BOTH' OR TRANSFER ALL' BLOCK

613 ILLEGAL USER CHAIN REFERENCED

614 PRIORITY EXCEEDS MAXIMUM ALLOWED (127)

615 ERROR IN CORE ASSIGNMENT (PROBABLE SYSTEM BUG,

616 XACT PARAMETER ZERO REFERENCED

617 ILLEGAL MATRIX NUMBER REFERENCED

618 CYCLIC (RECURSIVE) DEFINITION OF A MATRIX

619 MATRIX NOT DEFINED BY 'MATRIX' CARD

620 ILLEGAL MATRIX COLUMN NUMBER

621 ILLEGAL MATRIX ROW NUMBER

622 ILLEGAL BOOLEAN VARIABLE NUMBER

623 BOOLEAN VARIABLE NOT DEFINED BY 'BVARIABLE' CARD

624 CYCLIC (RECURSIVE) DEFINITION OF BOOLEAN VARIABLE

626 ILLEGAL GROUP NUMBER

627 C-FIELD LESS THAN B-FIELD IN 'TRANSFER-PICK' BLOCK

644 ERROR IN 'WRITE' BLOCK OPTION

669 IMPROPER QUEUE ASSIGNMENT (PROBAELE SYSTEM BUG)

670 IMPROPER QUEUE ASSIGNMENT (PROBAELE SYSTEM BUG)

698 ILLEGAL CHANGE IN 'CHANGE' BLOCK

699 ILLEGAL ARGUMENT IN 'EXECUTE' BLOCK

OUTPUT ERRORS

702 ILLEGAL FACILITY NUMBER

704 ILLEGAL USER CHAIN NUMBER

708 ILLEGAL LOGIC SWITCH NUMBER

712 ILLEGAL FULLWORD SAVEVALUE NUMBER

713 ILLEGAL HALFWORD SAVEVALUE NUMBER

714 ILLEGAL FACILITY NUMBER

715 ILLEGAL STORAGE NUMBER

716 ILLEGAL QUEUE NUMBER

717 ILLEGAL GROUP NUMBER

718 ILLEGAL USER CHAIN NUMBER

722 ILLEGAL STORAGE NUMBER

723 ILLEGAL QUEUE NUMBER

724 ILLEGAL TABLE NUMBER

726 ERROR IN SQUARE ROOT ROUTINE

727 ILLEGAL HALFWORD MATRIX SAVEVALUE NUMBER

728 ILLEGAL FULLWORD MATRIX SAVEVALUE NUMBER

729 ILLEGAL ENTRY TO OUTPUT (DAG07 CONTROL SECTION)
(PROBABLE SYSTEM BUG)

769 ILLEGAL 'TITLE' CARD

WARNING MESSAGES

850 ****WARNING****
ATTEMPT TO STORE INTEGER OF MAGNITUDE GREATER THAN 32767
IN A HALFWORD PARAMETER. CONTENT IS MAINTAINED MODULO 2**15

851 ****WARNING****
ATTEMPT TO STORE INTEGER OF MAGNITUDE GREATER THAN 32767
IN A HALFWORD SAVEVALUE. CONTENT IS MAINTAINED MODULO 2**15

852 ****WARNING****
ATTEMPT TO STORE INTEGER OF MAGNITUDE GREATER THAN 32767
IN A HALFWORD MATRIX SAVEVALUE. CONTENT MAINTAINED MODULO 2**15

853 ****WARNING****
XACT ATTEMPTING TO ENTER A 'QUEUE' BLOCK IS ALREADY A MEMBER OF
FIVE QUEUES. THE CONTENTS OF THE QUEUE SPECIFIED IN THE A-FIELD
WILL BE INCREMENTED BY THE AMOUNT SPECIFIED IN THE B-FIELD.
TESTING FOR MAXIMUM CONTENTS WILL BE MADE AND UPDATING DONE
ACCORDINGLY. STATISTICS ON AVERAGE TIME PER XACT, NUMBER OF ZERO
ENTRIES, PERCENT ZEROS, AND AVERAGE TIME PER XACT EXCLUDING ZERO
ENTRIES WILL BE IN ERROR.

854 ****WARNING****
XACT AT A 'DEPART' BLOCK IS NOT A MEMBER OF THE QUEUE SPECIFIED
IN THE A-FIELD. THE QUEUE CONTENTS ARE DECREMENTED BY THE
AMOUNT SPECIFIED IN THE B-FIELD AND THE XACT PROCEEDS
TO THE NEXT SEQUENTIAL BLOCK

861 ****WARNING****
END-OF-FILE REACHED ON JOBTAPE 1 (LOGICAL UNIT 1)
ALL XACTS ON THE CURRENT FILE HAVE ENTERED THE MODEL.

862 ****WARNING****
END-OF-FILE REACHED ON JOBTAPE 2 (LOGICAL UNIT 2)
ALL XACTS ON THE CURRENT FILE HAVE ENTERED THE MODEL.

863 ****WARNING****
END-OF-FILE REACHED ON JOBTAPE 3 (LOGICAL UNIT 3)
ALL XACTS ON THE CURRENT FILE HAVE ENTERED THE MODEL.

Appendix D

Mnemonics for the PRINT Block[Da]

Information to be Printed[Db]	Mnemonic
Block Counts[a]	
Current	W
Total	N
Current and Total	B
Chains[a]	
Current Events	MOV
Future Events	FUT
Clocks (Relative and Absolute)[a]	C
Logic Switches (Settings)[b]	LG
Savevalues(b)	
Halfword	XH
Fullword	X(or blank)
Statistics[b]	
Facilities	F
Queues	Q
Storages	S
Tables	T

---

[a] A and B Operand defaults must be used with the PRINT Block for this information.

[b] The PRINT Block A and B Operands specify the inclusive lower and upper limits, respectively, of the range of entity numbers for which the information will be printed. In Case of default, information for the entire range of the type of entity involved will be printed.

[Da] For a discussion of PRINT Block use, see Section 2.41.

[Db] This listing includes information applicable only to those GPSS entities actually discussed in this advanced printing.

* Also see page I-4.

---

* Also see page I-4.

---

* Also see page I-4.

Index for Case Study Use of Various GPSS Feature

GPSS Features	Case Studies																	
Entities(Ia)	2A	2B	2C	2D	2E	2F	3A	3B	3C	4A	4B	4C	4D	4E	5A	5B	5C	5D
Facilities	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓		✓	✓			✓
Functions																		
Continuous								✓	✓	✓	✓	✓	✓	✓		✓	✓	
Discrete							✓			✓			✓			✓		✓
List																		✓
Logic Switches																	✓	✓
Queues	✓	✓	✓			✓			✓	✓	✓	✓	✓				✓	
Savevalues															✓	✓	✓	
Storages			✓		✓	✓		✓			✓	✓	✓	✓			✓	✓
Tables																✓	✓	✓
Variables																		
Boolean																		✓
Integer															✓	✓	✓	✓
Real																✓		
Blocks(Ia)																		
ASSIGN											✓	✓	✓			✓	✓	✓
GATE																	✓	
MARK																		✓
PRIORITY													✓					
SELECT												✓						
SPLIT														✓				
TEST																✓	✓	✓
TRANSFER																		
BOTH mode								✓										
Statistical						✓					✓							✓
Unconditional				✓	✓	✓	✓					✓			✓	✓		✓
Control Cards(Ic)																		
CLEAR						✓	✓	✓	✓			✓		✓	✓	✓	✓	
RMULT									✓			✓		✓		✓	✓	

(Ia) Blocks corresponding to "Entities" are not shown. For example, because "Queues" appears as an entry under "Entities", the Blocks QUEUE and DEPART are not shown under "Blocks". The Blocks GENERATE, ADVANCE, and TERMINATE are used in each Case Study, therefore they have also been deleted from the "Blocks" list.

(Ib) Use of the ASSIGN Block implies use of Transaction Parameters in the same Case Study.

(Ic) The Control Cards SIMULATE, START, and END are used in each Case Study, therefore they have been deleted from the "Control Cards" list. REALLOCATE, RESET, and JOB have not been used in any of the Case Studies.