

Donald G. McBrien

Application Program

H20-0326-0

General Purpose Simulation System/360 User's Manual

This publication is an extension and amplification of the GPSS/360 Introductory User's Manual (H20-0304-0). It provides a detailed description and explanation of the component parts and operation of the GPSS/360 program. Examples illustrating the uses of GPSS/360 are given in appropriate sections of the manual. This manual should enable the reader to construct and simulate models using the full capabilities of GPSS/360.

First Edition

Significant changes or additions to the specifications contained in this publication will be reported in subsequent revisions or Technical Newsletters.

Copies of this and other IBM publications can be obtained through IBM branch offices. Address comments concerning the contents of this publication to IBM, Technical Publications Department, 112 East Post Road, White Plains, N.Y. 10601

CONTENTS

CHAPTER 1: INTRODUCTION	1	Basic GPSS/360 Card Format	18
Outline of Contents	1	Remarks Cards	18
CHAPTER 2: PRINCIPLES OF THE GPSS/360 PROGRAM	4	CHAPTER 4: VARIABLE ENTITIES	20
Block Diagrams	4	General Nature of Variable Entities	20
Clock Time — Relative and Absolute	4	Arithmetic Variables	20
Advance Block Times	5	Input Format for Arithmetic Variable Definition Card	21
GPSS/360 Entities and Their Attributes	5	Examples of Arithmetic Variables Used in Simulation Models	21
Formal Concepts	5	Core Allocation for GPSS/360 Arithmetic Variables	22
GPSS/360 Entities	5	Redefinition of Arithmetic Variables	22
Block Entities	6	Floating-point Arithmetic Variables	23
Transaction Entities	6	Input Format for Floating-point Arithmetic Variables	23
Standard Numerical and Standard Logical Attributes	8	Boolean Variables	23
System-Wide Numerical Attributes	9	Operators	23
Uses of Standard Numerical Attributes	9	Core Allocation for GPSS/360 Boolean Variable Entities	24
Various Ways to Specify Constant Values	14	CHAPTER 5: FUNCTION ENTITIES	26
Indirect Addressing of Entity Indices	15	General Nature of Function Entities	26
Three Ways to Specify Transaction Parameter Values	15	Types of Functions	26
Chain and Set Memberships of Transactions	16	Free-Format Function Follower Card	28
Basic GPSS/360 Chains	16	Continuous Numerical Valued Functions (Cn)	28
Assembly Sets	16	Discrete Numerical Valued Functions (Dn)	29
Events and Status Changes	16	List Numerical Valued Functions (Ln)	29
CHAPTER 3: S/360 CORE STORAGE ALLOCATION AND BASIC CARD FORMAT	18	Discrete (En) and List (Mn) Attribute Valued Functions	30
Format of S/360 Words	18	Uses of Function Values	31
Basic Words for GPSS/360 Entities	18		

Function Selection Factor (Fn) in TRANSFER Blocks	31	Block Redefinition	42
Symbolic Block Values in Assembly Program Functions.	31	CHANGE and EXECUTE Blocks	42
Examples of Functions Used in Data Processing Simulation Models	31	CHAPTER 7: TRANSACTION ENTITIES	53
Probability Distributions	31	S/360 Core Allocation for Transaction Entities	53
Function Packing.	33	Standard Numerical Attributes of Transaction Entities	53
Seek Time Distribution of 1301 Disk Storage Unit.	34	Transaction Parameter — Pn, P*n*,n	53
Mapping Functions	34	Transaction Transit Time — M1	53
Uniform Distribution of a Random Variable	34	Parameter Transit Time — MPn, MP*n	53
Error Conditions with Functions	35	Transaction Priority — PR	55
Input Errors in FUNCTION Definition Card and Function Follower Cards	35	Standard Logical Attributes of Transaction Entities	55
Execution Errors	35	Transaction Printout	55
Core Allocation for Functions	36	Current Events Chain	55
Redefinition of Functions	36	Future Events Chain	55
Increased Speed of List Functions	36	User Chains	60
Random Number Generation.	36	Interrupt Chain	60
CHAPTER 6: BLOCK ENTITIES	38	Matching Chain	60
General Nature of Block Entities.	38	ADVANCE Block, in Which Transactions are Delayed for Positive Time	60
Execution of Block-type Subroutines	38	Mean Time (Field A)	60
GPSS/360 Assembly Program Block Definition Cards	38	Spread Time or Function Modifier (Field B)	61
Contents of Block Fields	39	Transactions That are PREEMPTed While in an ADVANCE Block	61
GPSS/360 Core Allocation for Block Entities	40	Transactions That Enter ADVANCE Blocks	61
Standard Numerical Attributes for Block Entities	40	ADVANCE Block Count Statistics.	62
Normal Block Statistical Output	41	Internal Operation of the ADVANCE Block	62

TRANSFER Block from Which Generalized Transfers to Other Blocks Can be Made	62	LINK Block	70
Unconditional (blank) Selection Mode	62	UNLINK Block	72
Fractional Selection Mode.	63	LINK/UNLINK Examples	74
BOTH Selection Mode.	63	User Chain Statistics.	78
ALL Selection Mode	64	Block Types That Modify Transaction Attributes	78
PICK Selection Mode	64	ASSIGN Block	78
Function Selection Mode (FN)	64	INDEX Block	80
Parameter Selection Mode (P)	64	MARK Block	81
Subroutine Selection Mode (SBR)	65	COUNT Block	81
Simultaneous Selection Mode (SIM)	65	SELECT Block	83
GENERATE Block to Create Transactions	66	Blocks That Modify the Sequential Block Flow of Transactions	83
Sequence of GENERATE Block Operations	66	LOOP Block.	83
Initialization Interval and Creation Limit	67	TEST Block	84
Redefinition of a GENERATE Block.	68	GATE Block.	85
Error Conditions	68	Blocks That Create and Process Members of Transaction Assembly Sets	86
Internal Operation of GENERATE Block	68	SPLIT Block	87
Recreating GENERATE Block Transactions after a CLEAR Card	68	ASSEMBLE Block	88
Reactivation of GENERATE Blocks	68	GATHER Block	91
Redefinition or Changing to a GENERATE Block	69	MATCH Block	93
TERMINATE Block to Destroy Transactions	69	GATE M and GATE NM Blocks	94
Control of Simulation Run Length	69	PRIORITY Block to Change Priority Level of Transactions	96
Block Types That Manipulate User Chains	69	BUFFER Option	96
		Internal Operation of PRIORITY Block.	96
		BUFFER Block	97
		Overall GPSS/360 Scan	97
		Update Clock (Figure 24)	98

Start Overall GPSS/360 Scan (Figure 25)	98	Standard Savevalue Statistical Output . . .	125
Try to Move Transaction (Figure 26).	99	PRINT Block Output	125
Examples of Buffer and Priority Buffer Blocks	104	Examples of Savevalue Entities and SAVEVALUE Blocks	125
Different Priority Levels for Transactions	106	CHAPTER 9: LOGIC SWITCH ENTITIES . . .	130
JOBTAPES	109	General Nature of Logic Switch Entities	130
WRITE Block	109	Standard Numerical Attributes.	130
HELP Block.	110	Standard Logical Attributes	131
GROUP Entity	114	LOGIC Block	131
JOIN Block	115	GATE LR and GATE LS Blocks	131
REMOVE Block	116	Effect of RESET , CLEAR, and JOB Cards	131
EXAMINE Block	117	Statistical Output	131
SCAN Block	118	Examples of LOGIC, GATE LR, and GATE LS Blocks	131
ALTER Block	119	Pushdown Delay Chains Formed by Conditional Entry GATE LR and GATE LS Blocks	136
CHAPTER 8: SAVEVALUE ENTITIES	120	Effect of LOGIC S Block on GATE LS Delay Chain	136
General Properties of Savevalue Entities	120	GATE LR Delay Chain and LOGIC R Blocks	137
Standard Numerical Attributes	120	Overall Pushdown Delay Chain Considerations	137
Standard Logical Attributes	120	CHAPTER 10: FACILITY ENTITIES. . . .	138
SAVEVALUE Block	120	General Nature of Facility Entities . . .	138
Replacement, Addition, and Subtraction in Savevalue Location . .	121	Standard Numerical Attributes	138
Matrix Savevalues	121	Standard Logical Attributes	138
MSAVEVALUE Block.	122	Internal Nonaddressable Attributes . . .	140
Redefinition of Matrices	123	SEIZE Block	140
INITIAL Card	123	Procedure When Facility Is Not in Use.	140
Assembly Program Coding of INITIAL Card	124		
Effect of RESET, CLEAR, and JOB Cards	125		

Procedure When Facility Is in Use	140	Examples of Facility Block Types	152
Interactions With Other Facilities and Transactions	140	CHAPTER 11: STORAGE ENTITIES	156
Status Change Flag and Reactivation of GATE U Delay Chain	140	General Nature	156
RELEASE Block	140	Standard Numerical Attributes	156
How a Wrong Transaction With the Correct Number Can RELEASE or RETURN a Facility	141	STORAGE Definition Card	156
Status Change Flag and Reactivation of SEIZE-GATE NU Delay Chain	141	Standard Logical Attributes	156
Cumulative Time Integral of Facility Utilization	142	Internal Nonaddressable Attributes	157
PREEMPT Block	142	ENTER Block	157
ADVANCE Block	144	Transaction Cannot Move into ENTER Block	157
MATCH, ASSEMBLE, or GATHER Block	144	Transaction Succeeds in Moving into ENTER Block	157
RELEASing of a PREEMPTed Facility by a Current Events Chain Transaction	144	Status Change Flag and Reactivation of GATE SF and/or GATE SNE Delay Chains	158
Status Change Flag and Reactivation of GATE I and GATE U Delay Chains	144	LEAVE Block	158
Extension of the GPSS/360 PREEMPT Block	145	Status Change Flag and Reactivation of GATE SNF, GATE SE, and ENTER Delay Chains	159
RETURN Block	148	GATE SNE, GATE SF, GATE SNF, and GATE SE Blocks	159
Cumulative Time Integral of Facility Utilization.	149	Statistical Printout	160
A SEIZing or PREEMPTing Transaction Attempts to SEIZE or PREEMPT the Same Facility	150	Average Contents of Storage	160
GATE U, GATE NU, GATE I, and GATE NI Blocks	150	Average Utilization of Storage Capacity	160
Statistical Printout.	151	RESETing and CLEARing of Cumulative Time Integral.	160
Average Utilization	151	Average Time per Transaction	161
Effect of RESET and CLEAR Cards	152	Effect of RESET and CLEAR Cards on Storage Statistics	161
		Redefinition of Storage Capacity with Storage Card	162
		Examples of ENTER and LEAVE Blocks	162

CHAPTER 12: QUEUE ENTITIES	168
General Nature	168
Standard Numerical Attributes	168
Standard Logical Attributes	168
Internal Attributes	168
QUEUE Block	168
Operations When a Transaction Enters a QUEUE Block	170
DEPART Block	170
QTABLE Card	171
Multiple Queues	171
Execution Warning Messages	172
Inclusion of ADVANCE Blocks between QUEUE and DEPART Blocks	172
Statistical Printout	172
Average Contents of Queue	172
RESEtting and CLEARing of Cumulative Time Integral	174
Average Time Per Transaction	174
Effect of RESET and CLEAR Cards on Queue Statistics	174
Examples of QUEUE and DEPART Blocks	174
CHAPTER 13: DISTRIBUTION TABLE ENTITIES	177
General Nature	177
Table Statistics	177
Standard Numerical Attribute	177
Standard Logical Attribute	177
TABULATE Block	177
TABLE Definition Card	179
Table Arguments	180

Internal Operation of Tables and TABULATE Blocks	181
Transaction Operators for RT Arrival Rate Tables	182
Subsequent Processing of the Arrival Rate Transaction	182
Table Statistical Printout	183
Mean Value	183
Standard Deviation of Sample	183
Percentage of Total Argument Values Which Lie Within Each jth Frequency Interval	183
Cumulative Percentage of Arguments Less than or Equal to the Upper Limit of Each Frequency Interval	183
Cumulative Remainder of Argument Values Greater than the Upper Limit of Each Frequency Interval	183
Upper Limit of Each Frequency Interval as a Multiple of Mean Argument Value	183
Upper Limit of Each Frequency Interval as a Deviation from Mean Argument Value	184
Average Value of Overflow	184
Effect of RESET and CLEAR Cards on Table Statistics	184
Redefinition of Tables	184
CHAPTER 14: STATISTICAL PRINTOUT BLOCKS	187
PRINT Block	187
TRACE and UNTRACE Blocks	187
CHAPTER 15: CONTROL CARDS	189
START Card	189
Print Suppression (Field B)	189

Snap Interval (Field C)	189	GPSS/360 Features	196
Transaction Printout Option (Field D)	190	GPSS/360 Assembly Program Extensions	196
Control of Running Time of a Simulation Model	190	New Entities, Block Types, and Extended Features of GPSS/360 . .	197
RESET Card	190	GPSS/360 Output Editor.	199
CLEAR Card	191	CHAPTER 17: PRACTICAL SUGGESTIONS ON THE USE OF GPSS/360 .	200
Recreating Transactions at GENERATE Blocks	191	Hand Coding	200
Recreating the Arrival Rate (RT) Table Operators	192	Initial Debugging Runs	200
JOB Card	192	Classic Errors in GPSS/360 Models . .	201
END Card	192	Analysis of GPSS/360 Statistics	201
JOBTAPE Card	192	Statistical Problems in Simulation . . .	203
REWIND Card	193	APPENDIX A: GPSS/360 Program Errors .	204
Operation of the Rewind Card	193	Assembly Program Errors	204
LIST/UNLIST Cards	193	Input Errors	207
Operation of the List/Unlist Cards . .	193	Execution Errors	210
GPSS/360 Read/Save Feature	193	Execution Warning Messages	214
CHAPTER 16: DIFFERENCE BETWEEN GPSS III AND GPSS/360	195	APPENDIX B: GPSS/360 ASSEMBLY PROGRAM	215
Modification of GPSS III Block Types . . .	195	Block and Entity Symbols	215
GENERATE Block	195	Location Field Arguments	215
PREEMPT Block	196	Operand Field Arguments	215
PRINT Block	196	Relative Addressing of Block Locations	216
SAVEVALUE Block.	196	Function Follower Cards	216
UNLINK Block.	196	Symbolic Entity Reference	216
Modification of GPSS III Control Cards . .	196	Assembly Phase Output	218
STORAGE Definition Card.	196	Macros	222
Selective RESET Card	196	Update Feature	224
Selective CLEAR Card	196	GPSS/360 Assembly Control Cards. . .	225

SIMULATE Card	225	INCLUDE Card	229
JOB Card	225	FORMAT Card	231
END Card	226	TEXT Card	232
Pseudo-operations	226	COMMENT Card	234
ORG (origin)	226	EJECT Card	234
ICT (Increment)	226	SPACE Card	234
SYN (Synonymous)	226	OUTPUT Card	234
ABS and ENDABS	226	Graphic Output	234
Error Statements	226	GRAPH Card	235
Multiple Definition of Block Symbols	227	ORIGIN Card	236
APPENDIX C: OUTPUT EDITOR	228	X Card	237
Selection of Statistics, Titling, Comments, and Spacing	228	Y Card	239
REPORT Card	228	STATEMENT Card	240
TITLE Card	229	ENDGRAPH Card	240
		INDEX	242

CHAPTER 1: INTRODUCTION

This manual gives a detailed explanation of the operation of simulation models written with the General Purpose Simulation System/360. The various statistics provided by GPSS/360 models are described and interpreted. Numerous examples are given of important features and block combinations that occur in simulation models of representative systems. Appendix B describes the GPSS/360 Assembly Program. GPSS/360 models will always be coded in assembly program language.

It is assumed that the reader is thoroughly familiar with General Purpose Simulation System/360 Introductory User's Manual (H20-0304). The introduction enables the user of GPSS/360 to develop a broad range of models. Inevitably, however, the GPSS/360 user will encounter complex models that require a more detailed understanding of the GPSS/360 program. This manual should resolve the numerous and, oftentimes, fascinating subtleties that arise in GPSS/360 simulation models.

Table 1 summarizes the twelve types of GPSS/360 control cards (START, RESET, CLEAR, JOB, READ, SAVE, END, SIMULATE, JOBTAPE, REWIND, and LIST, UNLIST) and the six types of definition cards (VARIABLE, FUNCTION, INITIAL, STORAGE, QTABLE, and TABLE).

OUTLINE OF CONTENTS

The contents of Chapters 1 through 17, and Appendixes A through C, are outlined below:

Chapter 1: Introduction. The relationship between this manual and the introduction manual is defined.

Chapter 2: Principles of the GPSS/360 Program. Various basic concepts are introduced, such as:

1. Nature of the GPSS/360 block diagram language.
2. The GPSS/360 simulator clock and the associated absolute and relative clock times.
3. The 14 entities in the GPSS/360, and their associated Standard Numerical and Logical Attributes.
4. The chain and set membership of transactions.

Chapter 3: S/360 Core Storage Allocation and basic Card Format. Describes the three-way partition of S/360 core among:

1. The GPSS/360 program.
2. The basic words required for each of the 14 types of entities.
3. Additional common words required for each entity type.

Chapter 4: Variable Entities. Arithmetic variable entities are defined by VARIABLE cards as arithmetic combinations of the various Standard Numerical Attributes (SNA). The available arithmetic operations are addition (+), subtraction (-), multiplication (*), division (/) and modulo division (@). The Boolean operators are "and" (&) and "or" (+).

Chapter 5: Function Entities. Function entities, which are defined by FUNCTION and function follower cards, involve a relationship between an independent variable, or argument, and the dependent variable Function value. Function arguments can be any one of the Standard Numerical Attributes, while the dependent Function value (FNj) is itself one of the Standard Numerical Attributes. There are five types of GPSS/360 Functions: numerical valued (Continuous, Cn; Discrete, Dn; and List, Ln) and attribute valued (Discrete, En; and List, Mn).

Chapter 6: Block Entities. The general properties of block entities are described, such as their core allocation; their two Standard Numerical Attributes, Wj and Nj; and the two block types associated with the block entities CHANGE and EXECUTE.

Chapter 7: Transaction Entities. This is the most extensive and important chapter. The following basic concepts are described:

1. The three Standard Numerical Attributes (Pn, Ml, and MPn), and the two Standard Logical Attributes (Mj and NMj) which are associated with transactions.
2. The current events chain, future events chain, user chain, interrupt chain, and matching chain.
3. The creation of transactions in GENERATE and SPLIT blocks, their temporary exit and entrance in LINK and UNLINK blocks, and their subsequent destruction in TERMINATE and ASSEMBLE blocks.
4. The nature of transaction assembly sets and the associated block types SPLIT, ASSEMBLE, GATHER, MATCH, GATE M, and GATE NM.
5. The overall GPSS/360 scan and the influence of BUFFER and PRIORITY blocks.

Chapter 8: Savevalue Entities. Each savevalue entity has one Standard Numerical Attribute (Xj) or (XHj) whose value is changed by SAVEVALUE blocks and by INITIAL definition cards. Each Matrix Savevalue also has one Standard Numerical Attribute MXj (a,b) or MHj (a,b) whose value is changed by MSAVEVALUE blocks and by MATRIX definition cards.

Chapter 9: Logic Switch Entities. Logic switch entities have two Standard Logical Attributes (LRj and LSj) whose true-false values are changed by

LOGIC blocks. GATE LS and GATE LR blocks control the flow of transactions as a function of the true or false values of the above attributes.

Chapter 10: Facility Entities. Facility entities can be SEIZED and RELEASED by only one Transaction at a time. In turn, facilities can be PREEMPTed and RETURNed by only one transaction at a time. Standard statistics are provided on the percentage of utilization of each facility, and the average time that each transaction uses the facility. Four Standard Numerical Attributes (Fj, FR, FC, and FT) and four Standard Logical Attributes (Uj, NUj, Ij, and NIj) are associated with each facility entity. GATE U, GATE NU, GATE I, and GATE NI blocks control the flow of transactions as a function of the true or false values of the Facility Logical Attributes.

Chapter 11: Storage Entities. Storage entities can be simultaneously ENTERed by one or more transactions. These same transactions, or possibly different ones, will subsequently LEAVE the storage. The capacities of storage entities are defined by STORAGE definition cards. Standard statistics are provided on the average contents of each storage, the percent utilization of the storage capacity, and the average storage capacity. Seven Standard Numerical Attributes (Sj, Rj, SRj, SAj, SMj, SCj, and STj) and four Standard

Logical Attributes (SEj, SNEj, SFj, and SNFj) are associated with each storage entity. GATE SE, GATE SNE, GATE SF, and GATE SNF blocks control the flow of transactions as a function of the true or false values of the Storage Logical Attributes.

Chapter 12: Queue Entities. Queue entities provide statistics on transactions that are delayed by one or more causes. Transactions add to queue contents via QUEUE blocks and remove units from the queue contents via DEPART blocks. Seven Standard Numerical Attributes (Qj, QAj, QMj, QCj, QZj, QTj, and QXj) are associated with each queue entity. Standard statistics are provided on the maximum observed queue contents, the average queue contents, and the average time that each unit entry was delayed in the queue. The complete distribution of queue delay times can be obtained with QTABLE cards.

Chapter 13: Distribution Table Entities. Distribution table entities, which are defined by TABLE cards, provide statistics on the frequency distribution of the Standard Numerical Attribute arguments which are specified by the TABLE cards. TABULATE blocks activate a tabulation of the table argument. Standard table statistics include the average value and standard deviation of the observed values of the table argument. The

TABLE 1: SUMMARY OF GPSS/360 BLOCK SYMBOLS AND CARD TYPES

Table 8 in Chapter 6 describes the format of the various block types and their corresponding block symbols. A reference is given to the page on which each block type is discussed.

Listed below are the various definition and control cards used in GPSS/360 along with the chapter where each is discussed.

Definition Cards

FUNCTION card	(Ch. 5)
Function follower card	(Ch. 5)
VARIABLE card	(Ch. 4)
BVARIABLE card	(Ch. 4)
FVARIABLE card	(Ch. 4)
TABLE card	(Ch. 13)
STORAGE card	(Ch. 11)
INITIAL card	(Ch. 8)
MATRIX card	(Ch. 8)

Control Cards*

START
RESET
CLEAR
JOB
END
JOBTAPE
REWIND
LIST/UNLIST
READ/SAVE
SIMULATE (Appendix B)

*All except SIMULATE are discussed in Chapter 15.

table mean (TBj), the entry count (TCj) and the standard deviation (TD) are available as Standard Numerical Attributes.

Chapter 14: - Statistical Printout Blocks.

PRINT blocks can initiate the printout of a part of the standard GPSS/360 statistics. TRACE and UNTRACE blocks provide a means of printing out data on each block move which a transaction makes.

Chapter 15: Control Cards. Twelve control cards govern the overall operation of GPSS/360 simulation models: SIMULATE, START, RESET, CLEAR, JOB, END, JOBTAPE, REWIND, LIST, UNLIST, READ, and SAVE.

Chapter 16: Difference between GPSS III and GPSS/360. This chapter explains the transition from GPSS III to GPSS/360. Included are the

extensions of existing features; new entities and block types; and the extended features of GPSS/360.

Chapter 17: Practical Suggestions on the Use of GPSS/360. Practical suggestions are provided on the basis of extensive experience with GPSS III simulation models.

Appendix A: Error Conditions. The various GPSS/360 errors are summarized.

Appendix B: GPSS/360 Assembly Program. Complete operating instructions for the assembly program are provided (including a description of assembly program errors). A sample of assembly program input is given.

Appendix C: Output Editor. An explanation of the various card types necessary for user specified output with titles and/or deletions.

BLOCK DIAGRAMS

Block diagrams or flow diagrams are widely used to describe the structure of systems. They consist of a series of blocks, each of which describes some step in the action of the system. Lines which join the blocks indicate the flow of traffic through the system, or describe the sequence of events to be carried out. Alternative courses of action that arise in the system are represented by having more than one line leaving a block. Conversely, one block may have several lines entering it to represent the fact that this block is a common step in two or more sequences of events. The choice of paths, where an alternative is offered, may be a probabilistic event or a logical choice, depending upon the state of the system at the time of the choice. Both of these possible methods of selection can be used in the GPSS/360 program.

The units of traffic that move through the system depend upon the system being simulated. Units might be messages in a communication system, electrical pulses in a digital circuit, work items in a production line, or any number of other units. These units upon which the system operates in the GPSS/360 program will be called "transactions." The GPSS/360 program also has various other entities (facilities, storages, queues, tables, etc.) whose attributes are changed by the movement of transactions through the various block types.

Although a block diagram is a commonly used means of describing a system, the notation used in normal block diagrams depends upon the system and the person who is describing the system. For the purpose of the GPSS/360 program, certain conventions and systems concepts have been defined, each corresponding to some basic action or condition that generally occurs in systems. Statistical variations may be introduced in the block diagram, and many statistical sampling procedures are provided. Levels of priority may be assigned to transactions and complex logical decisions may be made during the simulation. It is also possible to simulate interdependence of variables in the system, such as queue lengths and input rates, or dollar value and processing time. In order to simulate a system, it must first be represented in terms of these concepts and block types. The program then creates transactions, moves them through the specified blocks, and executes the actions associated with each block. Table 9 in Chapter 6 summarizes the block symbols and coding formats which identify the various block types.

CLOCK TIME—RELATIVE AND ABSOLUTE

The GPSS/360 program operates by moving transactions from block to block of the simulation model in a manner similar to the way in which the units of traffic they represent progress in the real system. Each such movement is an event that is due to occur at some point in time. The GPSS/360 program maintains a record of the times at which these events are due to occur, then proceeds by executing the events in their correct time sequence. When transactions are blocked and cannot move at the time they should, the program moves them as soon as the blocking condition or conditions change.

In order to maintain the events in the correct time sequence, the GPSS/360 program simulates a clock that is recording the instant of time that has been reached in the model of the real system. The number shown by this clock at any instant is referred to as the "absolute clock time". Another clock time, the "relative clock time" is one of the Standard Numerical Attributes which can be externally addressed by the analyst. It is identified by the mnemonic symbol C1. All times in the simulation model are given as integral numbers. The unit of system time which is represented by a unit change of clock time is implied by the user, who enters all data relating to times in terms of the time unit he has selected. Whatever unit of time is chosen, such as millisecond or tenth of an hour, it must be used consistently throughout a simulation model. Fractional units of time are not permitted.

The GPSS/360 program does not simulate the system at each successive interval of time. Instead, it updates the absolute clock to the time at which the next most imminent event is to occur. The controlling factor in the amount of computing time that is used by the program is, therefore, the number of events to be simulated, not the length of real-system time over which the simulation is being made. For example, in a reliability model, 30 years of system operation may be simulated in 15 minutes, while in a programming model, one minute of system operation may require 30 minutes to simulate.

As described in Chapter 15, the relative clock time (C1) is set back to zero when a RESET card is encountered, and both the relative and absolute clock times are set back to zero when a CLEAR card is read.

ADVANCE BLOCK TIMES

Only one block type, the ADVANCE block (see Chapter 7) is able to delay transactions for a finite amount of clock time. Transactions which enter all other block types attempt to move to some next block in zero time, after completing the actions associated with the particular block type. These transactions may, however, be delayed in these blocks because of a blocking condition in the next block(s) to which they are trying to move. Eventually these transactions should leave the block in which they are delayed when the blocking condition is removed.

The ADVANCE block is able to assign a positive time delay to each transaction which enters the ADVANCE block. The delay time that is computed for each transaction as it enters an ADVANCE block is added to the absolute clock time at the instant of entering to produce a "block departure time." This represents the time at which the transaction will attempt to leave the ADVANCE block. It may not be the actual time of departure since the system may prevent the transaction from entering the next sequential block following the ADVANCE block. In that event, the transaction leaves the ADVANCE block as soon as the obstructing condition is removed.

Because the ADVANCE block time is computed separately for each transaction, transactions which enter the same ADVANCE block may not always leave in the order in which they arrived at the block. A later arrival may have shorter delay time, which will cause it to depart earlier than a predecessor.

When a transaction moves into a block at which the block time is zero (either because the block is not an ADVANCE block, or because a zero delay time is computed for an ADVANCE block), the program attempts to move the transaction immediately through this block and into the following block. This process is repeated as many times as possible, until either the Transaction reaches an ADVANCE block with a nonzero delay time, or the transaction is blocked and unable to move further. When the program begins moving a transaction, therefore, it moves the transaction as far as it can go during the current instant of clock time before proceeding to any other transaction that may also be due to move at that instant (the BUFFER block and the PRIORITY block with the buffer option are special exceptions to this rule, as described in Chapter 7).

It is possible that the movement of other transactions at the current instant of clock time may remove the blocking condition described for the above transaction. This original transaction may, therefore, begin moving at the same instant of clock time at which it was previously blocked. Indeed, it is quite possible that this may happen several times to a single transaction during a given

clock time. Eventually, the condition is reached in a simulation model where no further transactions can move into some next block. It is at this point that the GPSS/360 program updates the absolute clock to the time at which the next most imminent event is to occur. This event will almost always involve the departure of one or more transactions from one or more ADVANCE blocks in which they have been delayed for some finite time.

GPSS/360 ENTITIES AND THEIR ATTRIBUTES

Formal Concepts

A GPSS/360 block diagram model can formally be considered as a set of interrelated logical and mathematical symbols which represent those aspects of a system which are of interest. Each model consists of various elemental abstractions, called entities, by which the system is represented. GPSS/360 has 14 types of entities which are described below. Each of these entities has associated with it a set of properties or attributes that describes its status at any given time. These attributes have either numerical or logical values. It is the values of these numerical and logical attributes that the systems analyst is interested in, because they describe the performance of the system being modeled. Many of the entity attributes are externally addressable by the analyst with mnemonics such as Qj, Sj, Wj, etc. These Standard Numerical and Logical Attributes can be used in a variety of ways, as described later in this chapter. For example, queue number 10 is an entity and one of its attributes is its current contents, which may have a value, say of 5, at some given time. The current contents of queue 10 may be addressed as the Standard Numerical Attribute Q10. As the simulation progresses, these entities will interact with one another, thus producing transformations on the numerical or logical values of their various attributes. These transformations are called events. Entities may also be dynamically grouped into chains or sets of entities whose memberships will also change during the course of a simulation. Transactions, for example, can belong to the current events chain or to the future events chain. They are also members of a unique assembly set.

GPSS/360 Entities

GPSS/360 has 14 entities with which models can be constructed. These entities may be divided into the following six categories:

1. Basic entities
1. Blocks
2. Transactions

- | | |
|---------------------------|-------------------------|
| 2. Equipment entities | 3. Facilities |
| | 4. Storages |
| | 5. Logic Switches |
| 3. Computational entities | 6. Arithmetic Variables |
| | 7. Boolean Variables |
| | 8. Functions |
| 4. Statistical entities | 9. Queues |
| | 10. Frequency Tables |
| 5. Reference entities | 11. Savevalues |
| | 12. Matrix Savevalues |
| 6. Chain entities | 13. User Chains |
| | 14. Groups |

The block entities (Chapter 6) and transaction entities (Chapter 7) are truly basic, for practically all of the status changes and statistics which are gathered in a GPSS/360 model result from the movement of transactions into blocks, followed by the execution of the subroutine associated with each particular block type.

Block Entities

Block entities (described in Chapter 6) consist of 43 distinct block types, which are outlined in Table 8 in Chapter 6. Each block type is associated with only one of the 14 types of entities. This also includes block entities themselves because the EXECUTE and CHANGE blocks only refer to other blocks.

In the GPSS/360 assembly program each block is defined by an input card with the following basic format:

1	2	LOCATION	7	8	OPERATION	19	A, B, C, D, E, F, G, H, I
		Symbolic Block Location			Block Type Name		Arg A, Arg B, Arg C, Arg D
		Examples:			ASSIGN		3, FN10
		NEXT			SEIZE		4
		LOOP			SAVEVALUE		39, Q*3
		INNER					

The GPSS/360 program automatically keeps statistics (for each block in a model) on the total number of transactions entering the block, and on the number of transactions currently in the block. The normal GPSS/360 program for a 128k machine has core allocated for 500 blocks.

Transaction Entities

The units of traffic that are created and moved through blocks by the GPSS/360 program are called "transactions" (described in Chapter 7). In simulations transactions can represent:

Real physical entities

1. Messages in computing equipment and/or communication lines.
2. Segments of messages, which may be processed as separate entities.
3. Input/output operations on auxiliary storage equipment, that is, disk units, magnetic tape units.
4. Units and components in a reliability simulation model.
5. Parts moving through a manufacturing facility.
6. Automobiles being processed at a service station.
7. Cargo ships using a harbor facility.
8. People queueing for service at a bank, supermarket, or theater.

Nonphysical programming entities

1. Central control program in a real-time computer.
2. Channel control programs (one transaction for each data channel in a computer).
3. Message exchange control program.
4. Polling discipline in a communication line (one transaction for each line attached to a message exchange).

Each transaction possesses parameters (0-100) as its key attributes. Parameters may contain integers with one of two possible ranges; $-(2^{31}-1)$ to $(2^{31}-1)$ or $-(2^{15}-1)$ to $(2^{15}-1)$. The choice of size is determined by the user. These parameters have a variety of interpretations which again are determined by the user. Parameters do not have built-in significance. For example, the user may reserve one parameter for the character length of a message in a communication system; use another parameter to represent the terminal from which the message was sent, and a third to indicate the destination of the message. Another user may wish a parameter to contain the dollar value of an order and a second parameter to carry the due date for the order. It is not necessary for the GPSS/360 program to know what significance the user assigns to each parameter. A block type, the ASSIGN block, is provided to transform the values of the transaction parameters.

The Standard Numerical Attribute Pn refers to the value of parameter n of the transaction currently being processed by the GPSS/360 program.

Another important attribute of each transaction is its priority level. The priority level is an integer between zero and 127 inclusive. When two transactions are in competition for equipment (facilities, storages, and logic switches) the one with the higher priority will be the first to be processed. If both transactions have the same priority the transaction that has been delayed the longest will be selected first. A block type, the PRIORITY block, is provided to set the priority level of a transaction. A transaction retains its priority level until it enters another PRIORITY block. The initial priority level of transactions may be specified in field E of the GENERATE block at which they are created. If no field E priority level is specified in the GENERATE block the transactions have zero priority.

The following very important fact should always be remembered:

THE GPSS/360 PROGRAM AT ANY TIME IS ATTEMPTING TO MOVE ONLY ONE TRANSACTION.

The following block types transform the attributes (see Chapter 7) of transactions: ADVANCE, TRANSFER, GENERATE, TERMINATE, SPLIT, ASSEMBLE, GATHER, MATCH, PRIORITY, BUFFER, ASSIGN, INDEX, MARK, TRACE, UNTRACE, GATE, TEST, LOOP, LINK, UNLINK, COUNT, SELECT, JOIN, REMOVE, ALTER.

The standard GPSS/360 program for a 128k machine has core allocated for 600 transactions.

Facility Entities

The facility entities (described in Chapter 10) are provided to simulate equipment entities which can be used by only one transaction at a time. Central processing units, data channels, disk storage arms, and communication lines are typically simulated by facilities because they can only process one message at a time. The GPSS/360 program automatically provides a variety of statistics on the performance of facilities, including the percentage of time in use, the average time that transactions used each facility, and the number of uses of each facility. Four block types (SEIZE, RELEASE, PREEMPT, and RETURN) are associated with facilities. GATE blocks can also control the flow of transactions by testing whether or not facilities are in use or are being PREEMPTed. The normal GPSS/360 program for a 128k machine has core allocated for 150 facilities.

Storage Entities

The storage entities (described in Chapter 11) simulate equipment entities which have multiple capacities; that is, they can simultaneously process more

than one transaction at a time. The core storage in central processors and message exchanges is typically simulated with storages; generally, more than one message will occupy parts of their capacity. The GPSS/360 program automatically provides the same types of statistics as for facilities. Two block types, ENTER and LEAVE, and the STORAGE definition card, are associated with storages. GATE blocks can control the flow of transactions by testing whether or not storages are empty or are full. The normal GPSS/360 program for a 128k machine has core allocated for 150 storages.

Logic Switch Entities

Logic switch entities (described in Chapter 9) are used to simulate simpler physical and logical situations than facilities and storages. Logic switches can be in one of two binary states: reset or set. Logic switches are often used in place of facilities, since each logic switch requires only three bytes of S/360 core while each facility requires 28 bytes. No statistics are gathered, however, on the usage and states of logic switches. The LOGIC block puts logic switches into either a set or a reset condition. GATE blocks can control the flow of transactions by testing whether logic switches are in a set or a reset condition. The normal GPSS/360 program for a 128k machine has core allocated for 400 logic switches.

Variable Entities

Variable entities (described in Chapter 4) permit the computation of arithmetic combinations of the Standard Numerical Attributes described later in this chapter. The value of a variable is one of the standard numerical attributes, and is identified by the mnemonic Vj. These variable expressions are FORTRAN-like and employ the operators: +=addition, -=subtraction, *=multiplication, /=division, and @= modulo division. Variable cards define the above expressions. The normal GPSS/360 program for a 128k machine has core allocated for 50 arithmetic variables.

Boolean Variable Entities

Boolean variable entities (described in Chapter 4) permit the user to make decisions at a single GPSS/360 block based on the status and/or value of many GPSS entities. The elements which make up the Boolean variable are interpreted as 1 if nonzero and 0 if zero. Conditional statements (i.e., 'G', 'LE', 'NE', etc.), the Boolean operators "and" (*) and "or" (+), parentheses and indirect addressing are allowed in Boolean variable statements. The value

of a Boolean variable is itself one of the Standard Numerical Attributes, and is identified by the mnemonic BVj. The standard GPSS/360 program for a 128k machine has core allocated for 10 Boolean variables.

Function Entities

Function entities (described in Chapter 5) permit the computation of continuous or discrete functional relations between an independent variable, which is one of the Standard Numerical Attributes, and the dependent values of the function. This function value is also one of the Standard Numerical Attributes and is identified by the mnemonic FNj. Consequently, one function can use the value of another function as its argument. The normal GPSS/360 program for a 128k machine has core allocated for 50 functions.

Queue Entities

Queue entities (described in Chapter 12) can be referenced for the purpose of gathering statistics at any point in a model where blocking delays can occur. Transactions will increase the contents of a queue entity when they enter a QUEUE block. They reduce the queue contents by subsequently moving into a DEPART block. It should be assumed that somewhere between these QUEUE and DEPART blocks there are other blocks which might cause blocking delays, that is, SEIZE, PREEMPT, ENTER, GATE, and TEST blocks. The GPSS/360 program will automatically provide statistics on each referenced queue entity, such as average queue contents, maximum queue contents, average time per transaction in the queue, and the percentage of the transactions which went through the queue in zero time. The normal GPSS/360 program for a 128k machine has core allocated for 150 queues.

Table Entities

Frequency distribution Tables (described in Chapter 13) are the key statistical entities in GPSS/360. The frequency distributions of the following typical random variables can be obtained through TABULATE blocks and TABLE cards:

1. Transit time through the entire system or through any intermediate parts.
2. Distribution of queue contents and delay times.
3. Distribution of storage occupancy.
4. Distribution of the time intervals between entries into a table, e.g., an interarrival time distribution at a TABULATE block.

5. Distribution of the number of entries into a table during a specified unit time period, e.g., an arrival rate distribution at a TABULATE block.

Each table has as an argument one of the Standard Numerical Attributes. Each table has a specified number of intervals into which the values of the argument can fall. Each time a transaction moves into a TABULATE block and references a table, the value of the argument is determined as well as the table interval in which it falls. A count is maintained of the number of times that argument values fall into the various table intervals. At the end of each simulation run, the absolute and relative frequencies of argument values in each interval are printed out, as well as the overall mean value and standard deviation of the table argument. The normal GPSS/360 program for a 128k machine has core allocated for 30 tables.

Savevalue Entities

Savevalue entities (described in Chapter 8) are of two types; fullword and halfword. They are represented by either 2 or 4 byte areas into which the value of any of the Standard Numerical Attributes can be saved (via a SAVEVALUE block) for future references. This saved value is also one of the Standard Numerical Attributes whose value is given by the mnemonic Xj or XHj. The normal GPSS/360 program for 128k machine has core allocated for 400 fullword savevalues and 200 halfword savevalues.

STANDARD NUMERICAL AND STANDARD LOGICAL ATTRIBUTES

Each of the 14 types of GPSS/360 entities requires a fixed amount of core storage, in which its attributes are stored and transformed during a simulation run. Most of these attributes are only internally addressable by the GPSS/360 program. A significant subset, however, is also externally addressable by the analyst. These entity attributes are referred to as "Standard Numerical Attributes". Tables 2 and 3 summarize their properties. Chapters 4 through 13 give detailed explanations of the standard attributes associated with each type of entity.

The abbreviations "SNAj" and "SNA*n" are used extensively throughout this manual. For example:

1	2	LOC	7	8	OPERATION	19	A	INDEX NUMBER OF STORAGE	[NUMBER OF UNITS TO ENTER STORAGE]	B
					LEAVE			SNAj, SNA*n k, *n	[SNAj, SNA*n k, *n]	

SNA_j means that any one of the Standard Numerical Attributes (S_j, R_j, F_j, Q_j, FN_j, etc.) outlined in Table 2 can be used as a field A or B argument of the LEAVE block. SNA*n means that the entity index of any one of the Standard Numerical Attributes can be supplied indirectly by the value of one of the transaction parameters, e.g., Q*1, FN*11, V*3 (see "Indirect Addressing of Entity Indices" later in this chapter).

In this chapter the allowable values of numerical constant "k" are described under "Various Ways to Specify Constant Values", and the ways to specify the values of Transaction Parameters *n, Pn, and K*n are described under "Three Ways to Specify Transaction Parameter Values".

The Standard Numerical Attribute W_j, the number of transactions currently at Block j, can have any value from zero to (2¹⁵-1). The Standard Logical Attributes, on the other hand, have only two possible values: true and false. The Standard Numerical Attributes are restricted to integer values, with the following two important exceptions:

1. A function time modifier, FN_j, in field B of ADVANCE or GENERATE blocks can have noninteger values (see Chapter 7).
2. A function modifier, FN_j, in field C of an ASSIGN block can have noninteger values (see Chapter 7).

System-Wide Numerical Attributes

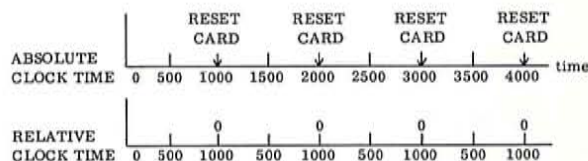
The following three Standard Numerical Attributes are system-wide; i.e., they are not associated with any specific entity: C1 = relative clock time since last RESET or CLEAR card, or since the start of a run if there have been no RESET or CLEAR cards.

Kn or n = Positive constant whose value is the number n, which cannot exceed 2147483647 or 2³¹-1.

RN(x) = One of eight random numbers (1 ≤ x ≤ 8) whose values are uniformly distributed: 0 ≤ RN(x) ≤ 999; each of the numbers has an equal probability, 1/1000, of occurring. As a function argument RN(x) is a decimal fraction whose values are uniformly distributed between 0 ≤ RN(x) ≤ 1.0.

"RESET Card" and "CLEAR Card" in Chapter 15 give detailed explanations of how the RESET and CLEAR cards change the value of the relative clock,

C1. The following simple example illustrates the difference between the relative clock time, C1, and the absolute clock time, which is not available as a Standard Numerical Attribute.



At absolute clock times 1000, 2000, 3000, and 4000, RESET cards are read. Just before these cards are read, the relative clock time, C1, is 1000, and immediately thereafter C1 is set back to zero.

The "RESET Card" and "CLEAR Card" sections in Chapter 15 describe how the RESET and CLEAR cards affect the random number generator. A RESET card or CLEAR card leaves the current random number seed undisturbed, so that a different sequence of random numbers will be generated in the next simulation run.

The JOB card resets the random number seed back to its original input value. Consequently, the initial sequence of random numbers will be generated again.

Uses of Standard Numerical Attributes

The Standard Numerical Attributes (Table 2) are used in GPSS/360 models in the following five ways:

1. As fields A, B, C, D, E, or F arguments of the 43 block types.

1	2	LOC	7	8	OPERATION	19	A	B	C	D
	10				SEIZE		*3			
	11				TRANSFER		ALL	90	160	10
	12				ASSIGN		4	FN10		
	39				ADVANCE		FN30	FN15		

2. As an argument (independent variable) of a function (see Chapter 5). The function argument is coded in field A of FUNCTION definition cards.

1	2	LOC	7	8	OPERATION	19	A	B
	10				FUNCTION		RN1	D10
	12				FUNCTION		P3	C6

Function Argument

3. As the Y_i values (dependent variable) of an attribute valued function (see Chapter 5). These function values (FN_j) are coded in columns 7-12, 19-25, ... 67-72 of FUNCTION follower cards.

1	2	7	8	13	19	25	31
	3	FUNCTION			P6, E3		
2		F	N80	6	Q10	10	V10

DEPENDENT FUNCTION VALUES

4. As a table argument (random variable) whose values are tabulated in a distribution table (see Chapter 13). The table argument is coded in field A of TABLE definition cards.

1	2	LOC	7	8	OPERATION	19	A	B	C	D
		3			TABLE	V30	1	1	10	
		10			TABLE	M1	0	100	30	

Table Argument

5. As an operand in an arithmetic variable expression (see Chapter 4).

1	2	LOC	7	8	OPERATION	19
		33			VARIABLE	Q9*FN10
		16			VARIABLE	P13/K100 + V10

The most important use of Standard Numerical Attributes is as block arguments, where they can be used in four ways:

1. Block argument as the index j of some entity.

1	2	LOC	7	8	OPERATION	19	A
		33			SEIZE	FN10	SEIZE Facility FN10
		9			LEAVE	*6	LEAVE Storage *6
		54			TABULATE	V7	TABULATE in Table V7
		40			LOGIC S	5	Put Logic Switch 5 in a Set condition

PR

The priority of the transaction currently being processed by the program. (0-127). This quantity may be changed by the PRIORITY block.

2. Block argument as the entity index j of a Standard Logical Attribute. This use occurs only with GATE blocks.

1	2	LOC	7	8	OPERATION	13	19	A	B	C
		310			GATE	LS	V10			Is Logic Switch V10 Set?

Chains

CHj

The current count, which is the number of the transactions on a specified user chain. This count is automatically maintained by the program.

TABLE 2: STANDARD NUMERICAL ATTRIBUTES

The following itemizes all of the available attributes. The mnemonic letter(s) is indicated, and the presence of the j which follows indicates the required integer.

Transactions

Pj

a parameter of the transaction currently being processed by the the program. Example: P5 for parameter 5. Parameters may be modified by the ASSIGN, INDEX, LOOP, and MARK blocks.

Ml

the transit time of the transaction currently being processed by the program. The quantity may be modified by the MARK block. When referenced the transit time is calculated as: $Ml = \text{current clock} - \text{mark time}$.

MPj

an intermediate transit time of the transaction currently being processed by the program. Accumulation of an intermediate transit time in a parameter of a transaction may be initiated by passing the transaction through a MARK block which has the desired parameter specified in field A. Example: MP8 for the intermediate transit time contained in parameter 8. When referenced the parameter transit time is calculated as: $MPI = \text{current clock} - Pj$.

TABLE 2: (Continued)

CAj	The average number of transactions on user chain j (truncated to an integer, i.e., $1.23 = 1$).
CMj	The maximum number of transactions on user chain j.
CCj	Total number of entries on user chain j.
CTj	Average time per transaction on user chain j (truncated).

Blocks

Nj	The entry count of the total number of transactions which have entered a specified block in the block diagram. This count is automatically maintained by the program. Example: N\$SAM for the entry count at block SAM. This count does not include the transaction currently in process at its current block.
Wj	The wait count, which is the number of transactions currently waiting at a specified block of the block diagram. This count is also maintained automatically by the program. Example: W\$HOLD for the current wait count at block HOLD. This count is also exclusive of the transactions currently in process at its current block.

System Attributes

Quantities (not directly altered by the block diagram).

Kj	An indication that the integer is a constant. Example K3276 for the integer 3276 or K0 for the integer zero.
RN(x) ($1 \leq x \leq 8$)	A computed random number. The value of the number is an integer between 0 and 999, inclusive, unless the quantity is to be used as the independent variable of a function. In that case, the number is a fraction greater than or equal to zero,

TABLE 2: (Continued)

C1	The current value of the simulator clock. This quantity is automatically maintained by the program.
<u>Equipment Attributes</u>	
<u>Storages</u>	
Sj	The contents of a specified storage in the block diagram. The quantity may be modified by ENTER and LEAVE blocks. Example: S2 for the contents of storage (number) 2.
Rj	The number of available units of space in the specified storage. This quantity may be modified by ENTER and LEAVE blocks. Example: R195 for the space remaining in storage 195.
SRj	Utilization of storage j in parts per thousand, i.e., if the utilization was .65 the computed value would be 650.
SAj	Average contents of storage j (truncated).
SMj	Maximum contents of storage j. This quantity is automatically maintained by the program.
SCj	Number of entries for storage j. This quantity is automatically maintained by the program.
STj	Average time each transaction used storage j (truncated).
<u>Facilities</u>	
Fj	The status of the specified facility in the block diagram. This value is zero if the facility is available; otherwise, it is one. This quantity may be modified by SEIZE, RELEASE, PREEMPT, and RETURN blocks. Example: F20 for the status of facility 20.

TABLE 2: (Continued)

FRj	Utilization of facility j in parts per thousand, i.e., if the utilization was .88 the value of FRj would be 880.
FCj	Number of entries for facility j.
FTj	Average time each transaction used facility j (truncated).
Groups	
Gj	The current number of members of group j.
<u>Statistical Attributes</u>	
Queues	
Qj	The length of a specified queue in the block diagram. This quantity may be modified by the QUEUE and DEPART blocks. Example: Q50 for the contents of queue 50.
QAj	Average contents of queue j (truncated).
QMj	Maximum contents of queue j. This quantity is automatically maintained by the program.
QCj	Number of entries in queue j. Automatically maintained.
QZj	Number of zero entries in queue j. Automatically maintained.
QTj	Average time each transaction was on queue j (including zero entries). When referenced the value will be truncated to an integer.
QXj	Average time each transaction was on queue j (excluding zero entries). Truncated.

TABLE 2: (Continued)

Tables	
TBj	The computed mean value of a specified histogram-type table which is defined by the user. The TABULATE block is used to enter values in one of these tables. Although the computed average can possess a fractional part, it is not retained unless the computed average is to be used as the independent variable of a function. Example: TB42 for the computed mean value of table 42.
TCj	Number of entries in table j.
TDj	Computed standard deviation of table j.
Savevalues	
Xj	The contents of fullword savevalue j.
XHj	The contents of halfword savevalue j.
MXj(a,b)	The contents of fullword matrix savevalue j, row a, column b. (a and b can be any other SNA)
MHj(a,b)	The contents of halfword matrix savevalue j, row a, column b.
Computational Attributes	
FNj	A computed function value. Only the integer portion is retained except when used as a function modifier in GENERATE, ADVANCE or ASSIGN blocks.
Vj	An arithmetic combination of Standard Numerical Attributes which is called a variable statement and is defined by the user. Only the integer portion is retained. (See Chapter 4.)
BVj	The computed value (1 or 0) of Boolean variable j.

TABLE 3: RANGE OF THE STANDARD NUMERICAL ATTRIBUTES

<u>Entity</u>	<u>Symbol</u>	<u>Range</u>	<u>Meaning</u>
Transactions	P	$\pm(2^{31}-1)$ $[(\pm(2^{15}-1))]$	Parameter, fullword [halfword]
	PR	$\emptyset -127$	Priority
	M1	$2^{31}-1$	Transit time
	MP	$2^{31}-1$	Parameter transit time
Blocks	N	$2^{23}-1$	Total entry count
	W	$2^{15}-1$	Current count
Facilities	F	Boolean 1 or \emptyset	Status of facility
	FR	$\emptyset -999$	Utilization (parts/thousand)
	FC	$2^{31}-1$	Entry count
	FT	$2^{31}-1$	Average time/transaction*
Storages	S	$2^{31}-1$	Current contents of storage
	R	$2^{31}-1$	Remaining contents
	SR	$\emptyset-999$	Utilization (parts/thousand)
	SA	$2^{31}-1$	Average contents*
	SM	$2^{31}-1$	Maximum contents
	SC	$2^{31}-1$	Entry count
	ST	$2^{31}-1$	Average time/transaction*
	Queues	Q	$2^{31}-1$
	QA	$2^{31}-1$	Average contents*
	QM	$2^{31}-1$	Maximum contents
	QC	$2^{31}-1$	Total entry count
	QZ	$2^{31}-1$	Number of zero entries
	QT	$2^{31}-1$	Average time/transaction*
	QX	$2^{31}-1$	Average time/transaction excluding zero
Tables	TB	$2^{31}-1$	Table mean*
	TC	$2^{31}-1$	Entry count
	TD	$2^{31}-1$	Standard deviation*
Savevalues	X	$\pm(2^{31}-1)$	Fullword savevalue
	XH	$\pm(2^{15}-1)$	Halfword savevalue
Matrix savevalues	M(a, b)	$\pm(2^{31}-1)$	Fullword matrix
	MH(a, b)	$\pm(2^{15}-1)$	Halfword matrix a = row b = column
Groups	G	$\pm(2^{15}-1)$	Number of items in group
User's chains	CA	$2^{31}-1$	Average number on chain*
	CH	$2^{15}-1$	Current number on chain
	CM	$2^{15}-1$	Maximum number on chain
	CC	$2^{31}-1$	Total entries
	CT	$2^{31}-1$	Average time entry*
Functions	FN	$\pm(2^{31}-1)$	Function

TABLE 3: (Continued)

Variables	V	$\pm(2^{31}-1)$	Arithmetic variable
	V	10^{-78} to 10^{75}	Floating-point variable ¹
	BV	1 or 0	Boolean variable
Random numbers	RN1-RN8	0 to .99999	As function argument
		0 to .999	Otherwise
Clock	C1	0 to $2^{31}-1$	Clock time relative to last RESET or CLEAR card

*Truncated to an integer.

¹Must have been defined as floating-point variable on VARIABLE definition card.

3. Block argument as the number $n=1, 2, \dots, j$, of a parameter of the transaction currently in the block. This occurs in field A of ASSIGN, INDEX, LOOP, MARK, COUNT, and SELECT blocks.

1	2	LOC	7	8	OPERATION	19	A	B
					ASSIGN	6	Q9	Parameter 6
					INDEX	V10	K9	Parameter V10
					LOOP	*11	203	Parameter *11
					MARK	FN2		Parameter FN2

The Xi and Yi values of fixed-field function followers (see Chapter 5) are specified in six-column fields so that these constants have values in the range from -99999 through 999999. Constants of attribute valued functions are limited to the range from 0 through 999999.

However, by an indirect method which uses INITIAL or VARIABLE cards, values in the range $\pm(2^{31}-1)$ can be used. The INITIAL card (see "INITIAL Card" in Chapter 8 for details) permits any number in the range $\pm(2^{31}-1)$ to be loaded into a fullword savevalue location at the start of a simulation run.

4. Block argument as the value of an attribute.

	A	B
INITIAL	X10	93764398

Here, the constant 93, 764, 398 is loaded into savevalue location 10. Its value can be referenced as a Standard Numerical Attribute, X10, in any block field. Example:

1	2	LOC	7	8	OPERATION	19	A	B	C
	301				ASSIGN	6	FN10		ASSIGN value of FN10 to Parameter 6.
	499				SAVEVALUE	30	Q9		SAVE the value of Q9 in Savevalue 30.
	97				ENTER	51	V3		ENTER Storage 51 with value of V3.
	263				ADVANCE	FN10			Spend a time interval equal to the value of Function 10.
	69				TRANSFER		FN3		TRANSFER to a next block whose number is the value of Function 3.

2	8	A	B
30	ADVANCE	X10	FN1

Here, a mean time of 93,764,398 can be specified for ADVANCE block 30, which is greater than the limit imposed by the values of constants as Standard Numerical Attribute block arguments.

An arithmetic variable (see Chapter 4) can similarly be defined to have any constant value in the range from $-(2^{31}-1)$ to $+(2^{31}-1)$.

Various Ways to Specify Constant Values

Constant values can be specified directly in two ways in the various GPSS/360 block and definition cards; Kn or n. The values of these constants are restricted to the range from zero to $(2^{31}-1)$.

Negative constants cannot be used directly as Standard Numerical Attributes. Negative constants as low as $-(2^{31}-1)$ can, however, be introduced indirectly into a model by INITIAL cards or VARIABLE definition cards as described below. Negative constants as low as -99999 can be specified as the Yi values of numerical valued functions.

1	2	LOC	7	8	OPERATION	19
	3				VARIABLE	36000000 = 1 hour in decimillisecons
	13				VARIABLE	-500000

The Standard Numerical Attributes V3 and V13 can now be addressed in a simulation model and they will have the constant values 36000000 and -500000.

Indirect Addressing of Entity Indices

The index numbers of entities are required for two reasons:

1. As the entity number j of a block argument.

1	2	LOC	7	8	OPERATION	19	A
					SEIZE	10	SEIZE Facility 10
					LEAVE	V3	LEAVE Storage V3
					TABULATE	14	TABULATE in Table 14

2. As the entity index j of an attribute (SNA j) which is referenced in any of the definition cards: block card, VARIABLE, TABLE, and FUNCTION.

- a. Q10=Current contents of queue 10
- b. S9 = Current contents of storage 9
- c. FN2=Value of function 22

To give greater flexibility and power to GPSS/360 models, the concept of indirect addressing of entity index numbers has been developed. By coding an asterisk followed by the number n (* n instead of j), the entity index number is now determined by the value of transaction parameter n . Each transaction in the GPSS/360 program has 0 to 100 parameters, so that *1, *2 . . . *100 are allowable indirect references. Repeating the above examples:

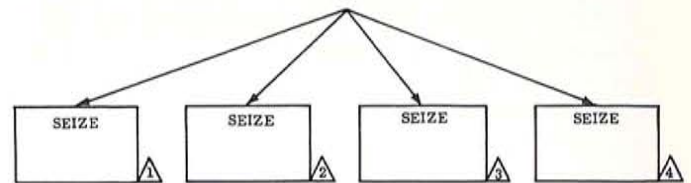
1. Indirect addressing of the entity number n .

7	8	19	A
		*6	SEIZE the Facility, whose index number is the value of the Transaction Parameter 6.
		*8	LEAVE the Storage whose index number is the value of Transaction Parameter 8.
		*1	TABULATE in Table whose index number is the value of Transaction 1.

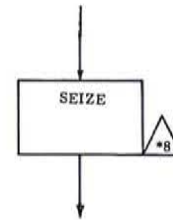
2. Indirect addressing of the entity index n of an attribute.

- a. Q*3 = Current contents of queue whose index number is the value of transaction parameter 3.
- b. S*12 = Current contents of storage whose index number is the value of transaction parameter 12.
- c. FN*4=Value of function whose index number is the value of transaction parameter 4.

Indirect addressing is used most often and powerfully to reduce the size of GPSS/360 models. As a simple example, assume that there are 4 facilities (numbers 1, 2, 3, 4) in a GPSS/360 model. At a certain point in the model each may be SEIZED by one of 50 transaction types, each of which is associated with a specific facility. Without indirect addressing it might be necessary to code four separate SEIZE blocks as follows:



These four SEIZE blocks can be reduced to one by indirect addressing. First, assume that at one or more points in the model an ASSIGN block assigns one of the numbers 1, 2, 3, 4 to transaction parameter 8, thereby associating the transaction with one of the four facilities. All of these transactions can now move through the single SEIZE block.



This single SEIZE block behaves like the four SEIZE blocks shown above. Each transaction now attempts to enter the single SEIZE block and obtain the facility whose index number is given by the value of transaction parameter 8.

In general, neither the block nor the entering transaction is in any way altered by execution of the indirect addressing feature. The values of the transaction parameter are subject to the same restrictions as a direct specification of an entity index. For instance, if the GPSS/360 program has 200 facilities (numbered 1, 2, . . . 200), and a transaction entered the above SEIZE *8 block with a parameter 8 value of 300, execution error number 498, "illegal facility number", would occur. Table 8, in Chapter 6 shows which card fields can be indirectly addressed.

Three Ways to Specify Transaction Parameter Values

The values of transaction parameters can generally be specified in three ways in a GPSS/360 model:

1. * n
2. P n
3. P* n

The GPSS/360 analyst should recognize that * n and P* n are completely different. The specification * n means "the value of transaction parameter n ". On the other hand, P* n means "the value of the transaction parameter m , whose index number m is given by the value of transaction parameter n ". For example, assume that the current transaction

Donald G. McBrien

being processed by the GPSS/360 program has the value 8 in parameter 5 and the value 37 in parameter 8. The attribute P*5 then has the value 37.

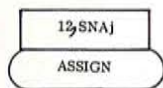
The indirect addressing of the entity index number in a block field can be achieved in all three ways:

SEIZE	*8
SEIZE	P8
SEIZE	P*8

However, the indirect addressing of the entity index number of an attribute can be achieved in only one way:

1. Q*3
2. S*12
3. FN*6

GPSS/360 does not provide any mnemonic mechanisms such as Q(SNAj) which might be interpreted as "the current length of the queue whose index is given by the value of Standard Numerical Attribute j". The analyst must achieve this by using a spare transaction parameter, e.g., parameter 12, as follows:



CHAIN AND SET MEMBERSHIPS OF TRANSACTIONS

Basic GPSS/360 Chains

GPSS/360 transactions may, at any time, be in one of the following transaction chains:

1. Current Events Chain. The overall GPSS/360 scan will always try to move these transactions into some next block(s).

Transactions in the current events chain will always be in one of two states:

- a. Active scan status; that is, the overall GPSS/360 scan will always try to move these transactions into some next block(s). The scan status indicator is zero.
- b. Delay status; that is, the transaction has been blocked from moving into some next block and has therefore been temporarily deactivated from the overall GPSS/360 scan. This is accomplished by linking the transaction into one of the eleven delay chains (described in Chapter 7) on the overall GPSS scan. The scan status indicator of these transactions is one.

2. Future Events Chain. The Future Events Chain consists of transactions which represent events that are to occur at some future time.

(Incipient successor transactions will be at block 0 in the future events chain, before entering the GPSS/360 model at a GENERATE block.)

3. User Chain. Transactions on this type of chain are in a temporarily inactive state. Transactions are placed on or removed from user chains by means of the LINK and UNLINK blocks.

4. Interrupt Chain. Transactions may also be in an interrupt status; i.e., they belong to neither the current nor the future events chain. These transactions have been PREEMPTed at facilities which they have SEIZED (see "PREEMPT Block" in Chapter 10).

5. Matching Chain. Transactions may also be in a matching status. These transactions:

- a. are in ASSEMBLE or GATHER blocks waiting for a specified number of transactions to be ASSEMBLED or GATHERed (see Chapter 7).
- b. are in MATCH blocks waiting for a mate transaction to appear in a conjugate MATCH block (see Chapter 7).

Assembly Sets

Each GPSS/360 transaction will also belong to one assembly set, which consists, minimally, of just itself. Whenever transactions enter a SPLIT block, one or more additional transactions are created. Each of these additional transactions may, in turn, SPLIT off further transactions. All of these offspring transactions and the original parent transaction will be chained together in a closed assembly set. As these transactions are destroyed, in TERMINATE or ASSEMBLE blocks, their assembly set linkages are also destroyed, thus leaving the surviving assembly set members still chained together.

EVENTS AND STATUS CHANGES

A GPSS/360 model goes through a sequence of events involving the following three major types of status changes:

1. Status change which creates or destroys a transaction entity (transactions are the only temporary entities among the 14 entity types).

Creating blocks: GENERATE, SPLIT

Destroying blocks: TERMINATE, ASSEMBLE

A CLEAR card will also destroy all transactions when it is encountered (see Chapter 15).

2. Status change of the values of one or more attributes, numerical or logical, of one or more entities. Most GPSS/360 status changes are of this kind.

3. Status change in the chain or set memberships of one or more transactions.

Two basic types of events are:

1. Internal events - these status changes occur within the GPSS/360 program primarily as the result of transactions entering blocks, which result in the execution of one of the block-type sub-routines.

2. External events - these status changes result from reading input cards from the various input devices.

Block definition cards - change the attributes of blocks

Other definition cards - FUNCTION, VARIABLE, FVARIABLE, BVARIABLE, STORAGE, TABLE, QTABLE, INITIAL, MATRIX

Control cards - START, CLEAR, RESET, JOB, END, JOBTAPE, REWIND, LIST/UNLIST, READ/SAVE

CHAPTER 3: S/360 CORE STORAGE ALLOCATION AND BASIC CARD FORMAT

The GPSS/360 program is divided into eleven separate "load modules", namely:

1. DAG01 - Control
2. DAG01A - Reallocate
3. DAG01B - Update
4. DAG02 - Assembly PASS1
5. DAG03 - Assembly PASS2
6. DAG04 - Input
7. DAG04B - Input
8. DAG04C - Input
9. DAG05 - Execution
10. DAG06 - Normal and Special Output
11. DAG07 - Graphic Output

Of the above, CONTROL is the only load module which remains in core for the duration of the GPSS/360 job. All other load modules are called, as needed, by the module currently being executed. There are two additional areas which will remain in core for the entire run. These include the entity area where the information for the 14 GPSS/360 entities is maintained and the area of GPSS/360 COMMON where additional storage is obtained as needed for the various entities.

FORMAT OF S/360 WORDS

The basic addressable unit in S/360 is called a "byte". Each byte contains eight bits numbered zero to seven from left to right. S/360 provides instructions which operate on fullwords (4 bytes), halfwords (2 bytes), or single bytes. Other instructions are provided which manipulate individual bits. For this reason, all GPSS/360 attributes are stored in either fullwords, halfwords, or bytes depending on the range of values expected for the attribute. Most indicators, such as a transaction's preempt flag, use only one bit. Therefore, some bytes are broken down into individual bits.

BASIC WORDS FOR GPSS/360 ENTITIES

Each of the 14 types of GPSS/360 entities requires a certain number of basic bytes; e.g., each

facility requires 28 bytes, each storage requires 40 bytes, etc. These basic core requirements are outlined in Table 6, and are discussed in detail in each appropriate chapter.

In addition to the basic core requirements, each entity may require variable amounts of additional core, i.e., bytes for transaction parameters. For this reason, a large area of core is set aside as GPSS/360 COMMON. Words or bytes of this COMMON area are assigned to the various entities as required and is returned when no longer needed.

BASIC GPSS/360 CARD FORMAT

With a few exceptions, the basic format for the cards defining a GPSS/360 model is as follows:

1. Name field - columns 2-6
2. Operation field - columns 8-18
3. Operand field - columns 19-72

The contents of each field and the possible exceptions are explained in detail in the following chapters which describe each block definition, entity definition, and control card.

Remarks Cards

All input cards will be printed as part of the printed output in the order in which they appear. An asterisk (*) in column 1 of any card indicates a remarks card. Columns 1-72 of this card are printed and the card is not examined further by the GPSS/360 program. The analyst may also add comments to any type of GPSS/360 card by leaving at least one blank column to the right of the required operand fields.

Appendix B describes the input format of the GPSS/360 assembly program.

TABLE 4: NORMAL QUANTITY OF GPSS/360 ENTITIES

<u>Entity Type</u>	Basic Core Allocation Per Item (bytes)	<u>Normal Quantity</u>		
		<u>64K</u>	<u>128K</u>	<u>256K and up</u>
Transactions	16*	200	600	1200
Blocks	12	120	500	1000
Facilities	28	35	150	300
Storages	40	35	150	300
Queues	32	70	150	300
Logic Switches	6	200	400	1000
Tables	48	15	30	100
Functions	32	20	50	200
Variables	48	20	50	200
Savevalues (fullword)	4	100	400	1000
Savevalues (halfword)	2	50	200	500
User Chains	24	20	40	100
Groups	4	5	10	25
Boolean Variable	32	5	10	25
Matrix Savevalue (full)	24	5	10	25
Matrix Savevalue (half)	24	5	10	25

*Add 20 bytes of common for every active transaction plus additional words for parameters.

CHAPTER 4: VARIABLE ENTITIES

GENERAL NATURE OF VARIABLE ENTITIES

Variables and functions (Chapter 5) are the two computational entities in GPSS/360. Because of their extensive use, particularly as block arguments, these are the first entities to be described in this manual.

In constructing the model of a system, the user may wish to express complex logical or mathematical interrelationship between system attributes. The program provides variable statements for this purpose.

The three types of variable statements provided by GPSS/360 are:

- Arithmetic Variables
- Floating-point Arithmetic Variables
- Boolean Variables

ARITHMETIC VARIABLES

Arithmetic variables are FORTRAN-like arithmetic combinations of the values of the various Standard Numerical Attributes, including other arithmetic variables. The definition of an arithmetic variable is accomplished by entering a single card, called a VARIABLE definition card, with the desired statement on it.

For example, the following VARIABLE definition card defines Arithmetic Variable 10:

```
10 VARIABLE Q9+3-P7*FN3
```

Whenever the value of Arithmetic Variable 10 is referenced, by V10, its value would be computed as the current length of Queue 9 (Q9) plus the constant 3, minus the product of Transaction Parameter 7 (P7) of the transaction currently being processed, multiplied by the value of Function 3 (FN3).

The values of Arithmetic Variables, Vj, are used in GPSS/360 models for the following five basic purposes:

1. Argument of a block field
2. Argument of a function
3. Dependent value of an attribute function
4. Argument of a table
5. Operand in another arithmetic variable.

As block arguments (item 1), the arithmetic variable values can represent:

1. An entity index j
2. The entity index j of a logical attribute (GATE block)

3. The index $n = 1, 2, \dots, 12$ of a transaction parameter (ASSIGN, INDEX, LOOP, MARK, and SPLIT blocks)

4. The value of an attribute.

Arithmetic variables may be indirectly addressed, e.g., V*12 is the value of the arithmetic variable whose index number is given by the value of transaction parameter 12.

Five arithmetic operator symbols are recognized in arithmetic variables:

- + denotes algebraic addition
- denotes algebraic subtraction
- * denotes algebraic multiplication
- / denotes algebraic division in which the remainder, if any, is discarded immediately before any further operations (only the quotient is retained)
- @ denotes modulo division in which the quotient is discarded and the remainder, considered positive, is retained.

NOTE: The symbol for modulo division used in GPSS III (a left parenthesis) is accepted if no other parentheses are used in the variable statement.

Any number of combinations of the above operations may be specified. The resultant sign of the computed value is determined by normal algebraic conventions. Negative values of variable statements are permitted. Variable statements are evaluated from left to right. Multiplication, division, and modulo division take precedence over addition and subtraction. Each element of the variable statement is evaluated and truncated before any arithmetic operations take place.

All operations are algebraic, and unsigned quantities are considered positive. Division by zero is not considered an error, and the result of the division is always zero.

Any of the Standard Numerical Attributes may appear in any arithmetic variable statement, including another arithmetic variable (V), with the obvious restriction that no arithmetic variable may refer to itself during its computation; otherwise, Execution Error 516 will occur. Constant values can be coded as either Kn or n. All Standard Numerical Attributes are considered integers, including function values (FNj) and distribution table means (TBj) which are truncated to form integers.

GPSS/360 allows the use of parenthetical expressions within arithmetic variable statements. Parentheses may be used to group terms or to denote multiplication. There can be no more than five sets of parentheses within a given variable statement (not including those used to define matrix save-values). The quantity defined within the set of parentheses denoted by the rightmost left parenthesis is evaluated first. There must be an equal number of left and right parentheses. An error will be given during input if an improper number of parentheses is used.

5*FN3+5*V6+5*P11

may now be written

5*(FN3+V6+P11)

If the variable is written 5(FN3+V6+P11), the parentheses denote multiplication.

INPUT FORMAT FOR ARITHMETIC VARIABLE DEFINITION CARD

The ARITHMETIC VARIABLE definition card consists of the following three fields:

1. The location field, beginning in column 2 contains the VARIABLE index number or symbol which will be used to reference the VARIABLE.
2. The operation field, beginning in column 8 contains 'VARIABLE'.
3. The operand field, beginning in column 19 contains the VARIABLE definition statement.

No blanks are permitted between characters, and the statement is terminated by the first blank encountered. There is no limit to the number of items in any given arithmetic variable statement, except that each statement must end by column 71. A longer statement can be accommodated by defining a second arithmetic variable statement on a second VARIABLE definition card with a different variable index number (k), and including the value of the second arithmetic variable statement (Vk) as one of the operands in the first arithmetic variable statement (j). The two system-wide variables, relative clock time and random number, must be codes as C1 and RN1-RN8 respectively. For example, consider the following three VARIABLE definition cards:

```
11 VARIABLE Q9+K3...+V2...-V29... FN10
 2 VARIABLE K9+TB13-FN19*Q10
29 VARIABLE FN9+S3*R7-Q9
```

Arithmetic Variable 11 requires a full 71 columns to be defined. However, V11 includes Arithmetic Variables 2 and 29 as elements, i. e. , V2 and V29.

The input phase of the GPSS/360 program will reject a VARIABLE definition card which violates the above rules.

An input error message will be printed below the listing of the incorrect VARIABLE statement card. Any VARIABLE statement definition error will cause subsequent deletion of the simulation run. A complete list of VARIABLE statement errors can be found in Appendix A.

EXAMPLES OF ARITHMETIC VARIABLES USED IN SIMULATION MODELS

1. 13 VARIABLE P10+K25

Whenever the value of Arithmetic Variable 13 is referenced by V13, its value is computed as the value contained in Transaction Parameter 10 of the transaction currently being processed, plus the constant 25. This offsetting variable is often used to relate two entity indices.

2. 7 VARIABLE X*4/100@10

Whenever the value of Arithmetic Variable 7 is referenced by V7, its value is computed as the value of the savevalue location X*4, whose number is given by Transaction Parameter 4 of the transaction currently being processed, divided by the constant 100 and then divided modulo by the constant 10. This serves to "unpack" a middle digit from the value of the savevalue location. For example, suppose Transaction Parameter 4 of the transaction currently being processed contains the value 125. Also, suppose savevalue location 125 contains the value aaabcc. When V7 is referenced, the following occurs:

- a. Transaction Parameter 4 is evaluated and yields the value 125.
- b. SAVEX location 125 (X*4) is evaluated giving aaabcc.
- c. This value is divided by 100, giving aaab as the result.
- d. The value aaab is divided modulo by the constant 10, giving a result of b.

Thus, V7 yields the value b, the desired middle digit.

3. 15 VARIABLE FN22+FN*11+V23

Whenever the value of Arithmetic Variable 15 is referenced by V15, its value is computed as the value of Function 22, plus the value of the function whose number is contained in Transaction Parameter 11 of the transaction currently being processed, plus the value of Arithmetic Variable 23.

4. 21 VARIABLE P12+V27/V27*K50

Whenever the value of Arithmetic Variable 21 is referenced by V21, its value is computed as:

a. The value of Transaction Parameter 12 of the transaction currently being processed plus the constant 50 if Arithmetic Variable 27 is greater than or equal to one (i.e., $V27/V27 = 1$).

or,

b. The value of Transaction Parameter 12 of the transaction currently being processed plus the constant zero if Arithmetic Variable 27 is zero (i.e., $V27/V27 = 0$).

Thus, Arithmetic Variable 27 has been used as a Boolean variable within Arithmetic Variable 21. Observe that $V27/V27$ will equal 0 even if V27 is nonzero but less than one. In this case, V27 is truncated to zero before the division operation is performed.

CORE ALLOCATION FOR GPSS/360 ARITHMETIC VARIABLES

The input phase of the GPSS/360 program contains a FORTRAN-like compiler which scans each VARIABLE definition card and compiles a program within GPSS/360 itself to compute the value of the arithmetic variable. Each such program requires the following number of 360 COMMON storage words:

1. Three words (twelve bytes) for each Standard Numerical Attribute element. Five words if it is a matrix savevalue.
2. Five words for each set of parentheses.
3. Two words for most operations (+, @, -, /, *). This may vary between one and three depending on the order of the operations statement.

(FN3+5)5 requires 14 words

5*(FN3+5) requires 12 words

(5+FN3)*5 requires 13 words

Each available arithmetic variable requires twelve basic internal words whether or not it is actually defined by a VARIABLE card.

These twelve basic internal words (V1-V12) will be used as shown below:

V1 and V2 are set up during input.

V1: The first byte of V1 is used as both a floating-point indicator and a cyclic indicator. The second byte is used to count the total number of instructions in the compiled program. The third and fourth bytes are used to store the total length of the compiled program.

V2: This word is used to save the address of the first instruction of the compiled program.

V3-V12 are used during execution.

V3 and V4: These words are used as accumulators during computation of the arithmetic variable statement.

V5-V9: These five words are used to save the computed values of the quantities within the parentheses. If five sets of parentheses are used, the first set that is evaluated will have its result placed in V9. The last set to be computed has its value placed in V5.

In the example of Variable 17 below there are four sets of parentheses used. The first set to be computed is the set following the rightmost left parenthesis. This would be $(5*4)$ since its left parenthesis is in column 37. The result (20) would be stored in V8 since it is the first of four sets to be computed. The next set $(P1-(5*4))$ is then computed and its value placed in V7. The value of $(FN3-4)$ is placed in V6 and the value of $(19(FN3-4))$ is put in V5.

V10: This word is used to store the return address.

V11 and V12: These words are used to store base registers 10 and 15.

17 VARIABLE (19(FN3-4) 17(P1-(5*4)))

REDEFINITION OF ARITHMETIC VARIABLES

If an arithmetic variable is redefined by another VARIABLE definition card during a simulation job, the compiler will first compile the necessary program. It will then compare the additional words required for this new program with the additional words used for the first definition of the arithmetic variable. This word count is contained in the third and fourth bytes of word V1. If the new word requirement is less than or equal to the original word count, the new program is stored in the block of common words allocated for the first VARIABLE card which defines the arithmetic variable. However, if more words are required for the second VARIABLE definition, the required words are obtained from the words still remaining in common and the core used for the original VARIABLE

definition is returned to the GPSS/360 common pool.

FLOATING-POINT ARITHMETIC VARIABLES

Floating-point arithmetic variables are similar to the arithmetic variables previously described, except that the elements are not truncated before arithmetic operations are performed. Likewise, the results of intermediate arithmetic operations are not truncated. Truncation occurs only when the final result has been determined.

INPUT FORMAT FOR FLOATING-POINT ARITHMETIC VARIABLES

The input format of the floating-point arithmetic variable definition card is identical to the format described previously for the arithmetic variable definition card except that 'FVARIABLE' is specified in the operation field. The input format rules for floating-point arithmetic variables are identical to those previously described for arithmetic variables. A floating-point variable can not have the same Variable index number as an arithmetic variable. If this is done, the latter of the two definitions is the one which is used in evaluation.

The use of the Floating-point Variables can be seen in the first example below. Variable 1 will be equal to 36 since the result of the division will not be truncated. Ten will be multiplied by 3.67 and the result, 36.7 will then be truncated. Variable 2 will be equal to 30 since the result of the division will be truncated to 3.

```
1 FVARIABLE      10(11/3)
2 VARIABLE       10(11/3)
```

NOTE: Modulo division is not permitted in floating-point variables.

The Standard Numerical Attribute V is used to reference both arithmetic variables and floating-point arithmetic variables. Evaluation of the Variable Vn is determined by the variable definition card which describes Vn.

Core allocation for GPSS/360 floating-point arithmetic variables is identical to that shown previously for arithmetic variables. Likewise, the operations performed when a floating-point arithmetic variable is redefined are identical to those described for redefinition of an arithmetic variable.

BOOLEAN VARIABLES

To increase the logical power and capabilities of GPSS/360, Boolean variable statements are provided. This makes it possible to make decisions at a single GPSS block based on the status and value of many GPSS entities.

Boolean variables are Boolean combinations of the values of the various Standard Numerical Attributes, including other variables.

Boolean variables are used and defined in the same manner as arithmetic variables. Instead of computing a value as in the arithmetic variable, the Boolean variable tests one or more logical conditions. It gives a result of one if the conditions specified are met and a result of zero if they are not met as specified.

Operators

Three types of operators are allowed in Boolean variables, namely; logical, conditional, and Boolean.

The logical operators are associated with the equipment entities of GPSS and are used to determine the status of these entities. The logical operators are:

FUn or Fn	1 if facility SEIZED or PREEMPTed, otherwise 0
FNUn	1 if neither SEIZED nor PREEMPTed, otherwise 0
FIn	1 if PREEMPTed, otherwise 0
FNIn	1 if not PREEMPTed, otherwise 0
SFn	1 if storage full, otherwise 0
SNFn	1 if storage not full, otherwise 0
SEn	1 if storage empty, otherwise 0
SNEn	1 if storage not empty, otherwise 0
LRn	1 if logic switch RESET, otherwise 0
LSn	1 if logic switch SET, otherwise 0

*EXAMPLE OF LOGICAL OPERATORS

```
3 BVARIABLE      FN12
4 BVARIABLE      SF3
```

Boolean Variable 3 tests to determine whether Facility 2 meets the condition of "not being PREEMPTed". If this is the case then BV3 equals one. If Facility 2 is PREEMPTed, then BV3 equals zero. Boolean Variable 4 tests whether storage 3 is full. If it is full the condition is met and the result is 1. If it is not full, the result is zero.

Conditional operators make algebraic comparisons between operands. The operands may be either constants or System Numerical Attributes. All conditional operators are stated within quotes.

The conditional operators are:

'G' greater than
 'L' less than
 'E' equal to
 'NE' not equal to
 'LE' less than or equal to
 'GE' greater than or equal to

In the example below, Boolean Variable 1 equals one if V2 is greater than 5. Otherwise, it equals zero. Boolean Variable 2 tests whether or not FN3 is less than or equal to P4. If the condition is met, BV2 equals one; otherwise, zero.

***EXAMPLES OF CONDITIONAL OPERATORS**

1 B VARIABLE V2'G'5
 2 B VARIABLE FN3'LE'P4

The Boolean operators are + representing "or" and * representing "and." An "or" operation tests whether either or both of the conditions are met. "And" requires both conditions to be met.

***EXAMPLES OF BOOLEAN OPERATORS**

5 B VARIABLE FN12+SF3
 6 B VARIABLE FN12*SF3
 7 B VARIABLE (V2'G'5)*(FN12+LR7)
 8 B VARIABLE (FNU3+FNU5+BVS'NE'0)
 9 B VARIABLE FNU3*FNU4*FNU5
 10 B VARIABLE FI2+(SNE4*SNE5)

BV5 equals one if either or both of the conditions are met. BV6 equals one only if both conditions are met. BV7 equals one if Variable 2 is greater than 5 and either Facility 2 is not PREEMPTed or Logic Switch 7 is RESET. BV8 equals one if any of the three conditions are met. A Boolean variable can refer to any SNA including another Boolean variable. BV9 equals one only if all three conditions are met. BV10 equals one if either Facility 2 is PREEMPTed, or both Storage 4 and Storage 5 are not empty.

Notice that parentheses are essential in expressing certain Boolean operations.

For example:

FI3*(FI2+FI4) requires Facility 3 to be PREEMPTed and either Facility 2 or Facility 4 to also be PREEMPTed. This statement: FI3*FI2+FI4

would equal one if either Facility 4 was PREEMPTed, or both Facilities 1 and 3 were PREEMPTed.

Parentheses should be used only where necessary. Use of unnecessary parentheses wastes both time and core storage.

Any quantity within parentheses will be interpreted as zero if equal to zero or as one if not equal to zero. If a Boolean variable is defined by just one SNA as in:

2 B VARIABLE V4

it will be computed as one if nonzero or as zero if equal to zero.

Boolean variables can be very useful in testing logical conditions of the system. For example, in a teleprocessing system a message cannot be transmitted to the CPU until three conditions are met simultaneously; namely: the channel must be free to transmit the message, the CPU must be free to issue the command, and there must be core storage available for the message. These conditions could be tested by the following blocks:

WAIT TEST GE R1, P1
 GATE NU CHAN1
 GATE NU CPU
 TRANSFER SIM, WAIT

However, the three conditions tested above could all be tested by the following TEST block:

TEST E BV1,1

Where BV1 is defined as:

1 B VARIABLE R1'GE'P1*FNU\$CHAN1*FNU\$CPU

CORE ALLOCATION FOR GPSS/360 BOOLEAN VARIABLE ENTITIES

During the input phase a program is compiled to evaluate the Boolean variable. The number of words required for this compiled program is computed by the following rules.

1. Three words for each logical operator such as FN13.

2. Five or six words for each set of parentheses depending on the particular operations performed within the operations.

3. Two words for the first Boolean operator and one for every Boolean operator after that.

For example:

FI3 + SNE2 * (FV3 + FV4)

Two words are allowed for the first "or" operator, one word for the "and" and one word for the next "or."

4. Six words for a conditional operation between two constants such as 17'GE'15. Two additional words are required for each of the operands that is not a constant.

For example:

V2'GE'FN3

Ten words of core would be required to bring in the result.

5. One additional word is always required.

For each Boolean variable, eight words of core are reserved whether or not the Boolean variable is defined or referred to.

These eight words are used as shown below.

BV1 (four bytes) is used to store the total length of the compiled program.

BV2 (four bytes) is used to store the address of the first instruction of the compiled program.

BV3 (four bytes) is used to store the return address from execution. This word is also used to test for cyclic definition. If BV3 is zero, the Boolean variable is not being evaluated. If there is an address in BV3, it is being evaluated.

BV4 is a one-byte accumulator.

BV5 (1 byte) contains the total number of instructions in the compiled program.

The two bytes of BV6 and the four of BV7 are used to store the values of the quantity within a set of parentheses. Every quantity defined by a set of parentheses will result in a value of one or zero. If it is evaluated as nonzero, it is given a value of one.

BV8 is a four-byte accumulator.

BV9 and BV10 (four bytes each) store base registers 10 and 15.

CHAPTER 5: FUNCTION ENTITIES

GENERAL NATURE OF FUNCTION ENTITIES

Functions and arithmetic variables (Chapter 4) are two computational entities of GPSS/360. There are no block types associated with functions. The values of functions (FNj) are the second most widely used Standard Numerical Attributes in GPSS/360 simulation models. Transaction parameter values are the most widely used (the indirect addressing, *n, of entity indices accounts for the greater use of parameter values). The widespread use of function values is not too surprising since many interrelationships in systems can be described in terms of functional relations between two variables. Each GPSS/360 function relates the values of a function argument, which is some independent variable in the simulation model, to dependent variable values of a function (FNj). The function arguments can be any of the Standard Numerical Attributes, including the values of other functions (FNk). The only exception to this rule is the GPSS/360 matrix savevalue.

Another important use of functions is the generation of random variable values. For such probability distribution functions, the argument is the random number, $0 \leq RN < 1$, while the dependent function values (FNj) are some random variable elements in the GPSS/360 simulation model.

TYPES OF FUNCTIONS

The five types of functions in GPSS/360 are listed below. Figure 1 illustrates the form of the three basic types.

	Field B of FUNCTION Card <u>Contains</u>
1. Continuous numerical valued	Cn
2. Discrete numerical valued	Dn
3. List numerical valued	Ln
4. Discrete attribute valued	En
5. List attribute valued	Mn

Table 5 illustrates how these five types of functions are defined by FUNCTION definition cards and function follower cards. Field A of the FUNCTION card defines the argument (independent variable) of the function. The argument may be any of the Standard Numerical Attributes. If the random number, RN, is used as a function argument, it has fractional values which are uniformly distributed between $0 \leq RN < 1$. Observe that RN will never equal 1.0. (In all other uses, the value of RN is $0 \leq RN \leq 999$.) Field B of the FUNCTION card defines the type of function and the number of points, n: Cn, Dn, Ln, En, Mn. Each function must have at least two points defined, even if the two Y_i values are equal.

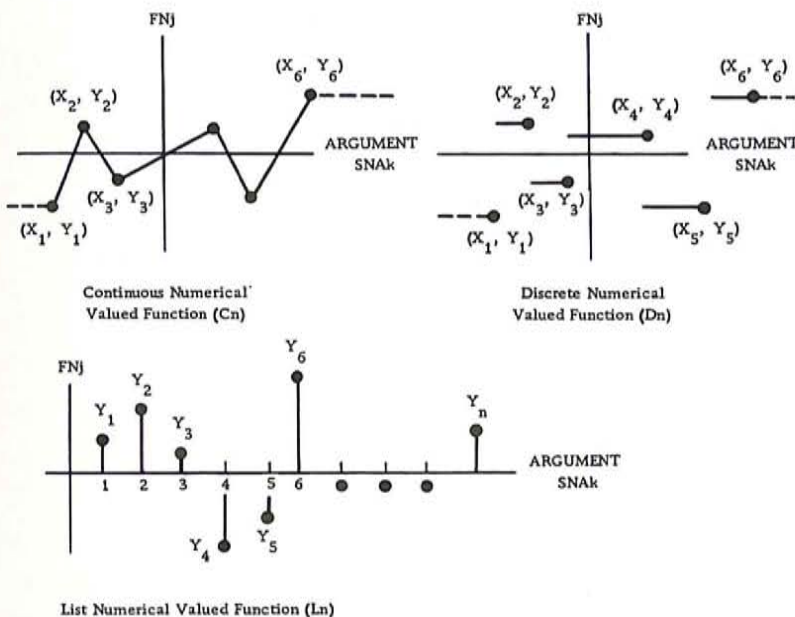


Figure 1. Types of GPSS Functions

TABLE 5: FUNCTION DEFINITION CARDS
FOR
VARIOUS TYPES OF FUNCTIONS

1. Continuous Numerical Valued Function (C8:8 points)

LOC	OPERATION		A	B							
j	FUNCTION		Arg	C8							
X1	Y1	X2	Y2	X3	Y3	X4	Y4	X5	Y5	X6	Y6
X7	Y7	X8	Y8								

2. Discrete Numerical Valued Function (D11:11 points)

LOC	OPERATION		A	B							
j	FUNCTION		Arg	D11							
X1	Y1	X2	Y2	X3	Y3	X4	Y4	X5	Y5	X6	Y6
X7	Y7	X8	Y8	X9	Y9	X10	Y10	X11	Y11		

3. List Numerical Valued Function (L9:9 points)

LOC	OPERATION		A	B							
j	FUNCTION		Arg	L9							
	Y1		Y2		Y3		Y4		Y5		Y6
	Y7		Y8		Y9						

4. Discrete Attribute Valued Function (E7:7 points)

LOC	OPERATION		A	B							
j	FUNCTION		Arg	E7							
X1	SNA1	X2	SNA2	X3	SNA3	X4	SNA4	X5	SNA5	X6	SNA6
X7	SNA7										

5. List Attribute Valued Function (M10:10 points)

LOC	OPERATION		A	B							
j	FUNCTION		Arg	M10							
	SNA1		SNA2		SNA3		SNA4		SNA5		SNA6
	SNA7		SNA8		SNA9		SNA10				

Each FUNCTION definition card must be immediately followed by a sufficient number of function follower cards to define the individual points (pairs of X_i , Y_i values) of the function. Each function follower card can define six pairs of X_i , Y_i values in twelve six-column fields: X_1 , in columns 1-6; Y_1 , in columns 7-12; X_2 , in columns 13-18; Y_2 , in columns 19-24; X_6 , in columns 61-66 and Y_6 , in columns 67-72. Observe that function points must be coded in columns 1 and 7. No comment cards may appear between the function follower cards for a particular function.

Both the X_i and Y_i values in the function follower cards can be specified as noninteger values, for example, as follows:

LOC	OPERATION		A	B							
5	FUNCTION		RNI	C5							
0	0	.33	.45	.40	1.60	.70	2.75	1.00	3.90		One Follower Card Required
X_1	Y_1	X_2	Y_2	X_3	Y_3	X_4	Y_4	X_5	Y_5		

Each successive X_i value must be greater than the preceding value, i. e., $X_i < X_{i+1}$. Otherwise

the function will not be defined by the input phase of the GPSS/360 program and Input Error 223 will occur.

Because of the six-column width of the fields in the fixed-field function follower card, the maximum range of X_i or Y_i values in functions is: $-99999 \leq X_i$ or $Y_i \leq 99999$. However, a free-format function follower card is accepted in GPSS/360.

FREE-FORMAT FUNCTION FOLLOWER CARD

In GPSS/360 the user may specify the X and Y coordinates of functions in a free format. This enables the user to specify more than six characters for any point thereby increasing the maximum input coordinate values to $\pm 2^{31}-1$. The input routines of GPSS/360 will determine the type of function input specified by the user and will indicate any format errors. It will not be necessary for the user to give any special indication if the free format is used. However, certain rules must be followed with the free format:

1. The first entry must start in column 1.
2. The last entry must occur in or before column 71.
3. The X_i and Y_i coordinates of a point are separated by a comma.
4. The sets of coordinates are separated by a slash (/).
5. The X_i and Y_i coordinates for a particular point must occur on the same function follower card.

Examples of Free-Format:

```

1      2      8      19      72
      5      FUNCTION RN1, E3
      0, -913/.542, 15739688/1.0, V10

10     FUNCTION *3, D5
      0, 5/1, 10/2, 15/3, 20/4, 25

15     FUNCTION RN8, C4
      0, 0/5, 12/.68, 15/1.0, 20
  
```

20 FUNCTION Q7, C16
0, 0/5, 1/10, 2/20, 330, 4/40, 5/50, 6/.../100, 11
110, 12/120, 13/130, 14/140, 15

CONTINUOUS NUMERICAL VALUED FUNCTIONS (Cn)

Figure 1 illustrates that the successive (X_i, Y_i) points of continuous numerical valued functions (Cn) are connected by straight lines. Whenever an argument value lies between two successive X_i points ($X_i < \text{argument} < X_{i+1}$) the GPSS/360 program interpolates linearly to obtain a dependent function value (FNj) between the two associated Y_i points: $Y_i > \text{FN}_j > Y_{i+1}$ or $Y_i < \text{FN}_j < Y_{i+1}$ (see Figure 2).

The Y_i values are stored in core words as floating-point values and all internal interpolations are done with noninteger, floating-point arithmetic. However, the internal value is finally truncated, so that the function value FN_j is always an integer, with the following two important exceptions:

1. A function time modifier (FN_j) in field B of ADVANCE and GENERATE blocks may have noninteger values (see Chapter 7).
2. A function modifier (FN_j) in field C of an ASSIGN block may have noninteger values (see Chapter 7).

A continuous numerical valued function (except for the above two cases which involve nonintegers) actually has the graphic form shown in Figure 3.

Figure 3 shows two successive points, (X_i, Y_i) and (X_{i+1}, Y_{i+1}) of a continuous numerical valued function. All four values may be nonintegers. The function values FN_j , however, are restricted to successive integer values: $Y_0, Y_1, Y_2, \dots, Y_{n-1}, Y_n$. The truncation performed by GPSS/360 program is such that, for $X_j \leq \text{argument value} < X_{j+1}$, function value = Y_j . Either or both of the points X_j and X_{j+1} can be nonintegers.

Figure 3 shows that for all argument values less than the second point (X_2) the function has a constant value equal to the first Y value, i. e.,

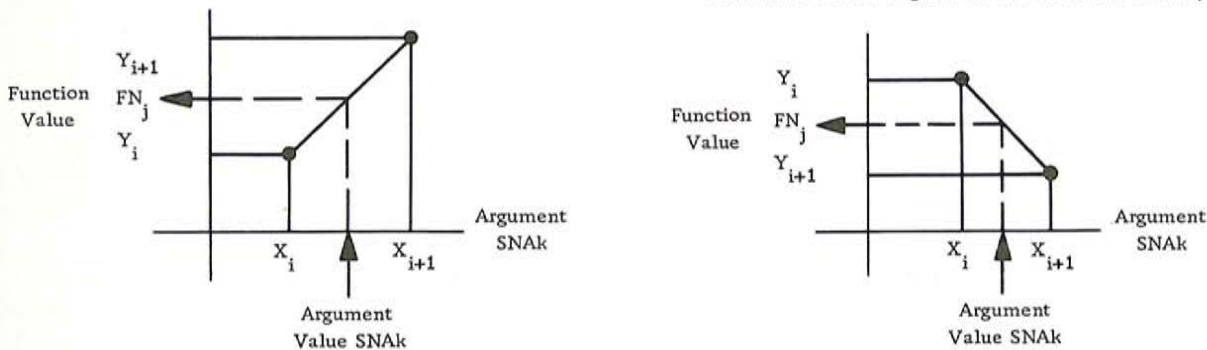


Figure 2. Linear Interpolation Between Adjacent Function Points

$FN=Y_1$. Similarly, for all argument values greater than the last point (X_n) the function has a constant value equal to the last Y value ($FN=Y_n$).

DISCRETE NUMERICAL VALUED FUNCTIONS (Dn)

Figure 4 shows that discrete numerical valued functions have the same function value, $FN_j=Y_i$, for all argument values in the interval between X_{i-1} and X_i : i. e., $X_{i-1} < \text{argument value} \leq X_i$.

No interpolation is performed, and the value at the right-hand end of the interval is used. As in the case of continuous functions, for argument values less than the first X_1 point and greater than the last X_n point, the following occurs:

1. $FN_j=Y_1$ for all argument values $\leq X_1$
2. $FN_j=Y_n$ for all argument values $\geq X_n$

(Assume in Figure 4 that the Y_i values coded in the function follower card are integer values. Otherwise, noninteger Y_i points would be truncated to integer Y_i values.)

LIST NUMERICAL VALUED FUNCTIONS (Ln)

In many cases the X_i argument values of functions will be the successive integers 1, 2, 3, ... n.

As described in "Examples of Functions Used in Data Processing Simulation Models" later in this chapter, the execution time to compute the value (FN_j) of such functions can be greatly reduced by coding these functions as list numerical valued functions. The X_i argument values in the function follower cards will not be examined by the GPSS/360 input program. Instead, they will be assumed to be the successive integers: $X_1 = 1$, $X_2 = 2$, ... $X_i=i$, ... $X_n=n$ (see figure 5).

The corresponding Y_i values must, however, be coded in the appropriate fields of the function follower cards. To make the function follower cards more understandable, the analyst can code the values $X_i=i$, although they will never be examined. If the argument value lies outside the range 1 to n, Execution Error 509 will result.

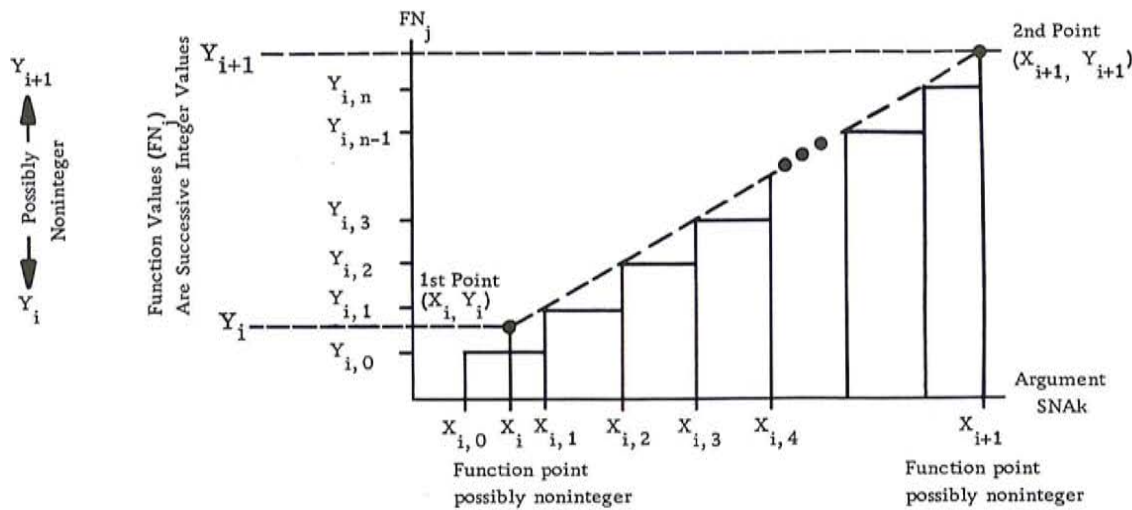


Figure 3. Form of Continuous Numerical Valued Functions

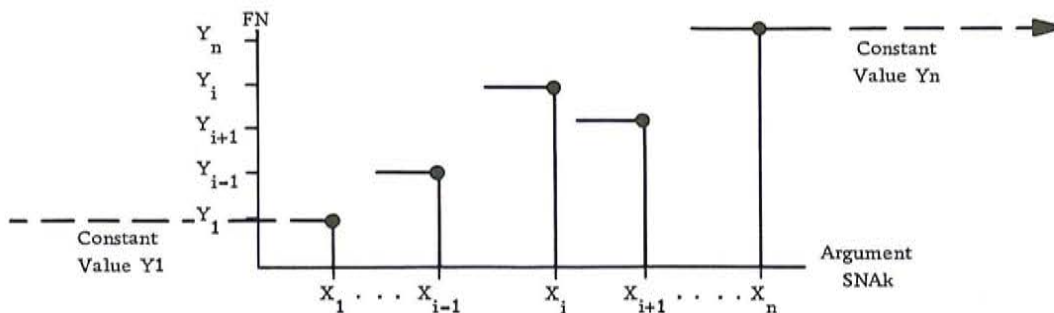


Figure 4. Discrete Numerical Valued Functions

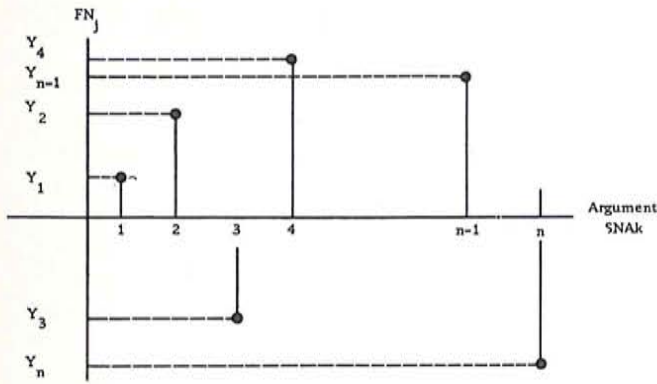


Figure 5. List Numerical Valued Functions

DISCRETE (En) AND LIST (Mn) ATTRIBUTE VALUED FUNCTIONS

The preceding sections have described the three types of numerical valued functions whose Y_i points are defined as pure numbers. There are two more types of functions whose Y_i point may be any one of the Standard Numerical Attributes:

1. Discrete attribute valued functions (En), which are analogous to discrete numerical valued functions (Dn).

2. List attribute valued functions (Mn), which are analogous to list numerical valued functions (Ln).

It is no longer possible to illustrate attribute valued functions in a two-dimensional plane. Instead, they must be portrayed by the pictorial device which is shown in Figure 6.

Attribute valued functions can also be illustrated by the following tabular arrays:

DISCRETE		LIST	
$X_i = \text{Argument (SNA}_k)$	$Y_i = \text{FN}_j$	$X_i = \text{Argument (SNA}_k)$	$Y_i = \text{FN}_j$
X_1	SNA_1	(1)	SNA_1
X_2	SNA_2	(2)	SNA_2
X_3	SNA_3	(3)	SNA_3
\vdots	\vdots	\vdots	\vdots
X_n	SNA_n	(n)	SNA_n

For the discrete attribute valued functions, the function value FN_j has the value of the i^{th} Standard

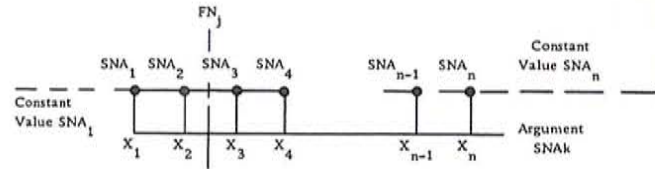


Figure 6. Attribute Valued Functions

Numerical Attribute (SNA_i) for all argument values in the interval, $X_{i-1} < \text{argument value} \leq X_i$. For the list attribute valued functions, the argument X_i points are implicitly assumed to be the successive integers 1, 2, . . . n.

Consequently, for an Argument value= i , the function value is $\text{FN}_j = \text{SNA}_i$. The notations Pn and *n are both acceptable as argument values in a function. Constants may have values in the range from -99999 to 999999.

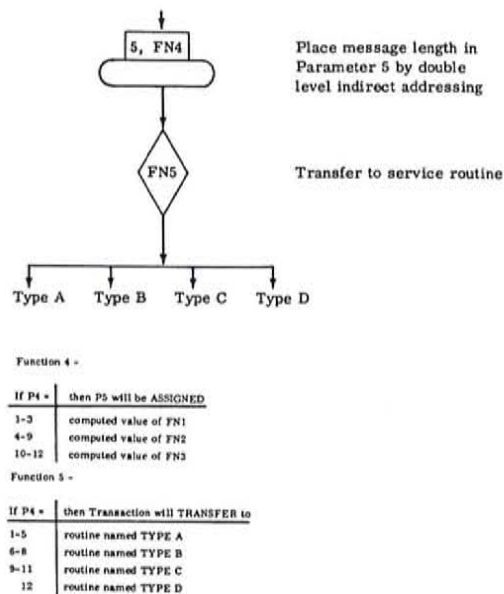
The following example illustrates one of the many possible uses of attribute valued functions:

In a system there are twelve message types with the following message lengths:

Types 1-3 80-126 characters represented in FN1
 4-9 97-320 characters represented in FN2
 10-12 50-150 characters represented in FN3,
 and are serviced by the following routines:

Types 1-5 TYPE A
 6-8 TYPE B
 9-11 TYPE C
 12 TYPE D

Assuming that the message type is in Parameter 4, the following block sequence would assign the



message length and direct the message to its service routine.

In the example above Function 4 provides a double level of indirect addressing. In Function 5 the function points are block addresses. These symbolic addresses are converted to actual numerical values by the assembly program before control is transferred to GPSS/360.

USES OF FUNCTION VALUES

The values of functions (FN_j) are used in GPSS/360 models for the five basic purposes described in Chapter 2.

1. A, B, C, D, E, or F argument of a block
2. Argument of another function
3. Dependent Y_i point of an attribute valued function
4. Argument of a table
5. Operand in an arithmetic variable.

As block arguments (item 1) the function values can represent:

1. An entity index (j)
2. The entity index (j) of a logical attribute (GATE block)
3. The index n=1, 2, . . . 100 of a transaction parameter (ASSIGN, INDEX, LOOP, MARK, and SPLIT blocks)
4. The value of a Standard Numerical Attribute, FN_j.

FUNCTION SELECTION FACTOR (FN) IN TRANSFER BLOCKS

The function selection factor (FN) of TRANSFER blocks is discussed in detail in Chapter 7.

TRANSFER FN, SNA_k, SNA₁

The GPSS/360 program will attempt to move the transaction into a next block whose number is the sum of:

The value of the function FN (SNA_k) whose index number (j) is given by the value of the Standard Numerical Attribute SNA_k specified in field B.

Plus, the value of the field C Standard Numerical Attribute SNA₁

Therefore, next block=FN (SNA_k) + SNA₁.

Examples:

LOC	OPERATION	A	B	C	Next Block Equals Value Of
	TRANSFER	FN	10		FN10
		FN	10	30	FN10 + 30
		FN	*2		FN*2
		FN	*3	FN3	FN*3 + FN3

SYMBOLIC BLOCK VALUES IN ASSEMBLY PROGRAM FUNCTIONS

The GPSS/360 assembly program permits the coding of symbolic block locations as Y_i values in function follower cards. This is necessary because function values (FN_j or FN*n) may be used in the following block types to indicate a nonsequential block number to which the transactions are to move:

2	LOC	7	8	OPERATION	19	A	B	C
				TRANSFER		FN	j	SNA _k
				TRANSFER		(Mode)	FN _j	
				LOOP		SNA _k	FN _j	
				SPLIT		SNA _k	FN _j	
				GATE Auxiliary		SNA _k	FN _j	
				TEST Optor		SNA _k	SNA ₁	FN _j

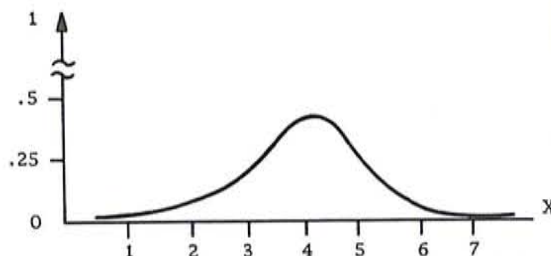
Since it is difficult to know what block numbers the assembly program will assign to the various blocks in a model, the use of symbolic locations as Y_i values becomes a virtual necessity.

EXAMPLES OF FUNCTIONS USED IN DATA PROCESSING SIMULATION MODELS

Probability Distributions

One of the most common types of functions in all simulation models is a probability distribution. A probability distribution is concerned with some sort of random variable (RV) which can be represented by real numbers.

$$f(x) = P(X=x)$$

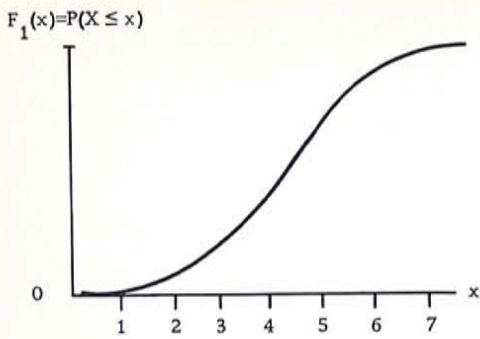


Possible values of random variable RV

Figure 7. Probability Density Function

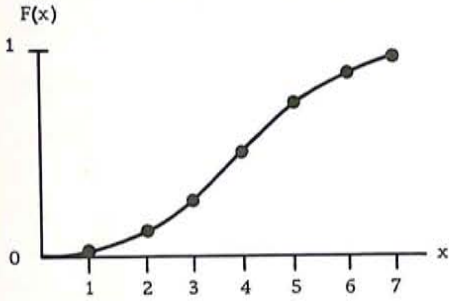
In Figure 7, the probability density function of RV is shown. For X=5, f(x)=0.25.

Probability distributions can be represented graphically in several ways. One of the most common is to graph the Density Function of RV (see Figure 7).



Possible values of random variable RV

Figure 8. Cumulative Distribution Function



Possible values of random variable RV

Figure 9. Approximation of the Cumulative Distribution Function

Another way of representing the Probability Distribution graphically is to graph the cumulative distribution function (see Figure 8). In this case, for any real number x , $F(x) = P(RV \leq x)$; i.e., the probability that the random variable RV is less than or equal to x is $F(x)$.

The cumulative distribution function can be approximated in Figure 9 by straight line segments.

Possible Values of Random Variable RV

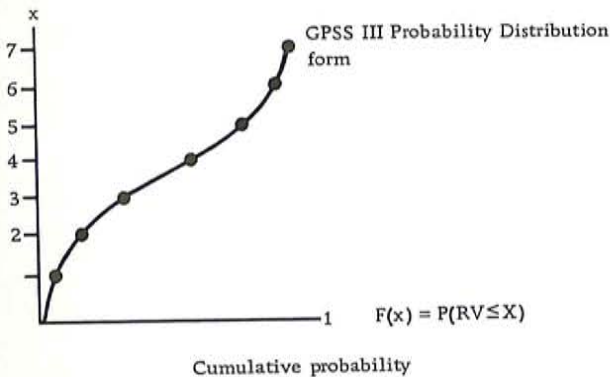


Figure 10. Alternative Form of Cumulative Distribution Function

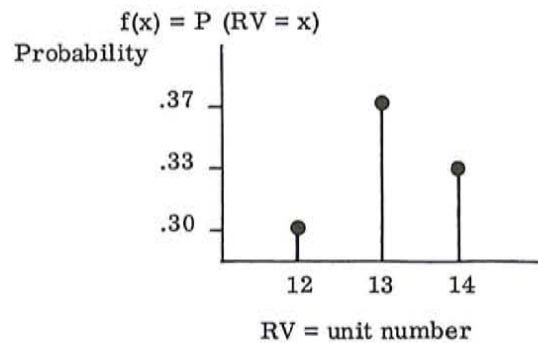
By plotting additional points, the accuracy of the fitted straight-line approximation will be increased.

The values of $F(x)$ in Figure 9 are numbers from zero to one inclusive. The curve can be flip-flopped (see Figure 10) so that the $F(x)$ values are now the independent variables and the random variable values (x) are the dependent variables.

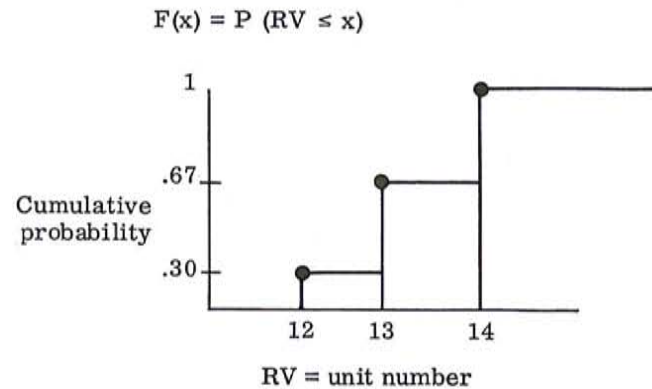
Figure 10 is the type of function which is used in GPSS/360 models to represent a probability distribution.

Example: Probability Distribution

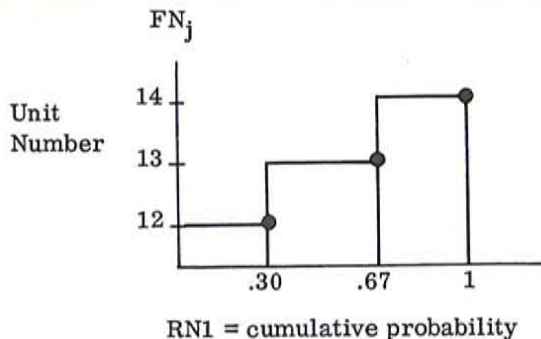
As an example of a discrete probability distribution consider the following case. One of three I/O units is to be chosen. The probability of choosing unit number 12 is .30, the probability of choosing unit number 13 is .37, and the probability of choosing unit number 14 is .33. The probability density function is of the following form:



The cumulative distribution function has the following form:



By changing the cumulative probability to the independent variable, the probability distribution takes the following GPSS/360 form of a probability distribution:



By using the GPSS/360 Function Argument RN1, the following results are obtained:

Unit number 12 if $0 \leq RN1 \leq .29$

Unit number 13 if $.29 < RN1 \leq .66$

Unit number 14 if $.66 < RN1 < 1$

Thus, unit number 12 will be chosen with probability .30, unit number 13 will be chosen with probability .37, and unit number 14 will be chosen with probability .33, which are just the desired probabilities.

The function would be coded in the following manner:

```
j      FUNCTION    RN1, D3
      .29, 12/.66, 13/1, 14
```

Example: Exponential Distribution

In many applications the number of transactions which arrive in a system per unit time have a Poisson distribution. The interarrival times have a negative exponential distribution of the form $\frac{1}{m} e^{-t/m}$, where m is the mean interarrival time.

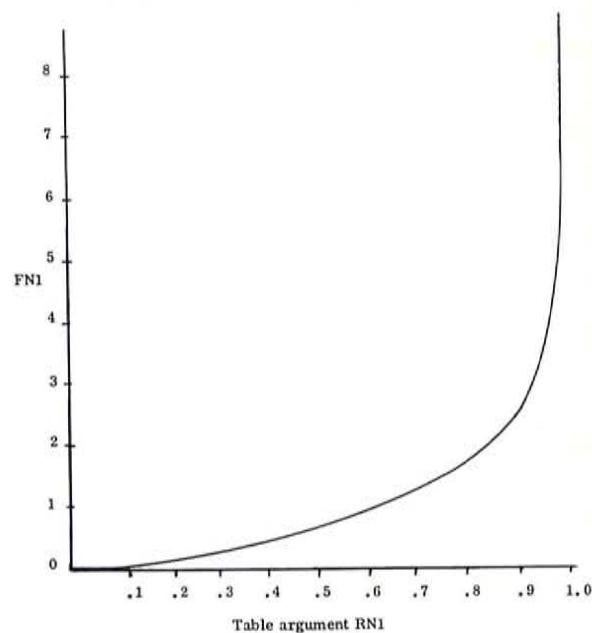
This distribution may be used in a GENERATE block with the field A mean (m) and the field B modifier (FN1). The same function modifier may be used to produce an exponential service time distribution in an ADVANCE block. The exponential distribution is represented in the graph shown below.

The function gives results which are accurate to within .1 percent for $m \leq 250$, and 1 percent for $m \leq 45$.

Function Packing

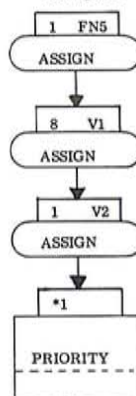
Data processing simulation models require many functions. Often, several operands must be assigned from functions which have the same function argument. Some of these operands can be packed into the same function, thus reducing the number of functions required.

2		8										19	
1		FUNCTION										RN1, C24	
1	7	13	19	25	31	37	43	49	55	61	67		
0	0	.1	.104	.2	.222	.3	.355	.4	.509	.5	.69		
.6	.915	.7	1.2	.75	1.38	.8	1.6	.84	1.83	.88	2.12		
.9	2.3	.92	2.52	.94	2.81	.95	2.99	.96	3.2	.97	3.5		
.98	3.9	.99	4.6	.995	5.3	.998	6.2	.999	7	.9998	8		



Each Y_i ordinate value can have a maximum of six digits in the fixed format function follower card and ten digits in the free format function follower card. Each six- or ten-digit ordinate can contain one or more operands. If a Y_i ordinate contains more than one operand, the function is said to be "packed."

Example 1:



```
5 FUNCTION RN1, D4
(1,11) (4,21) (8,31) (1,42)
```

V1=P1/K10 Division quotient = message type

V2=P1(K10 Modulo division remainder = Priority

Each Function value has the packed form a b, where

a = message type = 1, 2, 3, 4

b = Priority = 1, 2

The use of a packed function requires some method of shifting out the packed values (or, "unpacking" the function). This is done by use of arithmetic variables. Division by 10^n causes a right-shift truncation by n positions. Division modulo 10^n causes a left-shift truncation by n positions.

The above sequence assigns a message-type number to Parameter 8 and a priority-level number to Parameter 1 of each transaction from a probability distribution given in FN5. A typical ordinate value, say 21, is interpreted to mean message-type 2 with priority 1. The cumulative distribution function is shown below:

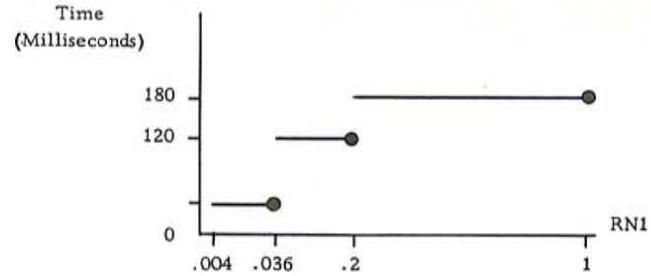
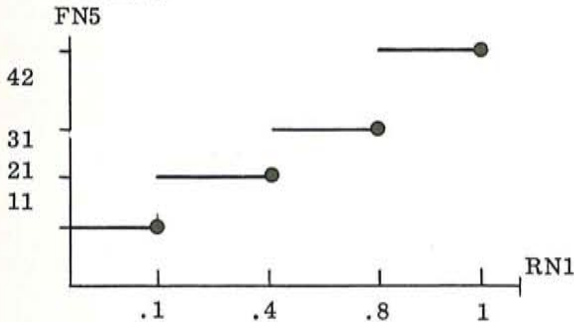


Figure 11. 1301 Seek Time Distribution

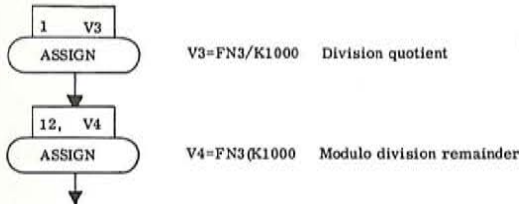
MAPPING FUNCTIONS

Another useful function is one that maps a sequence of integers (a, a+1, . . . b) into a second sequence of integers (c, c+1, . . . d), and conversely, maps the sequence (c, c+1, . . . d) into (a, a+1, . . . b). This is easily done, provided that no integer belongs to both sequences. The function shown in Figure 12 may be used.

In this function, an argument of 'a' would produce a FN_j value of 'c'. An argument of 'a+1' would produce a Y value of 'c+1', etc. Also, an argument of 'c' would produce a Y value of 'a', etc.

Example 2:

3 Function P3, D3
(1, 200007) (2, 222100) (3, 005001)



In the above example, Parameters 1 and 12 of the entering transaction are assigned on the basis of Transaction Parameter 3 which is the argument of Function 3. For instance, if Parameter 3 contains a 2, Parameters 1 and 12 will be assigned 222 and 100 respectively.

Seek Time Distribution of 1301 Disk Storage Unit

A very useful function is one that describes the move time for an access arm of the 1301 Disk Storage Unit. For random access using the 1301, this distribution is the following discrete GPSS/360 function:

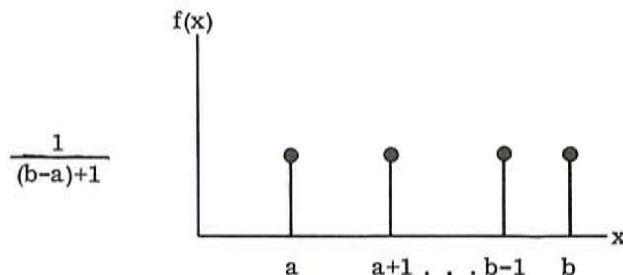
```
2 FUNCTION RN1, D4
.004, 0/.036, 50/.2, 120/1, 180
(This function is shown graphically in Figure 11.)
```

The 1301 has arm movement times of 0, 50, 120, or 180 milliseconds.

Figure 12. Mapping Function

Uniform Distribution of a Random Variable

Another very useful function in GPSS/360 simulation models is a probability distribution which is uniform over a number of sequential integers a, a+1, . . . b. Each value Y in the sequence has the probability $f(x) = \frac{1}{(b-a)+1}$, which is plotted below.



This could be represented in GPSS/360 by a discrete cumulative probability function with RN1 as its argument. However, since, only successive integral values are desired, the following continuous function can be used to reduce 360 running time (see Figure 13). The function values are truncated to give the desired integral values.

An RN1 argument value from 0 to $1/(b-a)+1$ would produce the value a after truncation. An argument value from $1/(b-a)+1$ to $2/(b-a)+1$ would produce the value a+1. The crucial thing to observe here is that since RN1 cannot equal 1.0, the function will never have the value b+1.

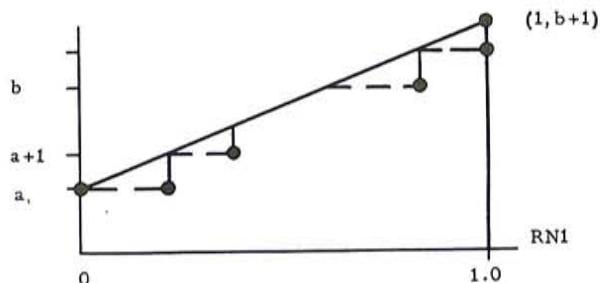


Figure 13. Uniform Distribution Function

ERROR CONDITIONS WITH FUNCTIONS

Input Errors in FUNCTION Definition Card and Function Follower Cards

The FUNCTION definition card will be rejected and the function will be left undefined (with no COMMON storage allocated) if:

1. The index j in the location field is greater than the maximum allowable function entity number.
2. An illegal alphanumeric symbol is used for the function argument in field A.
3. Field B contains characters other than Cn, Dn, En, Ln, or Mn.

The proper number of function follower cards must directly follow the FUNCTION card to define the n points. The fixed-format card is limited to six points (pairs of X_i , Y_i values) per follower card. The free-format card may specify a variable number of points. The X_i values must be monotonically increasing; i. e., each successive X_i value must be greater than the preceding X_{i-1} value.

Execution Errors

Three errors which involve functions can occur during the actual running of a GPSS/360 model (see Appendix A for listing of all error conditions).

An error condition which is detected by the GPSS/360 program results in a printout with the following format:

```
ERROR NO _____
      TRAN_FROM_TO_CLOCK ____
      TERMINATIONS TO GO _____
```

The second line gives the number of the transaction being processed, the numbers of the blocks FROM and TO which it is attempting to move, the clock time and the termination count remaining in the run. The meaning of the numbers which appear in the first line is given below.

<u>ERROR NUMBER</u>	<u>SIGNIFICANCE</u>
506	Cyclic Function Definition
507	Illegal Function index j Example: In the standard GPSS/360 program for a 128K machine j must not be greater than 50.
508	Function Not Defined Error 508 can occur when the analyst forgets to define the function.

Observe that cyclic Error 506 can occur in four ways:

1. FN_j is referenced directly as the argument in the field A of the FUNCTION definition card.

Example:

	A
10	FUNCTION FN10

2. FN_j is referenced as an operand in an arithmetic variable which is the argument of the function.

Example:

	A	B
10	FUNCTION	V5
5	VARIABLE	... FN10...

3. FN_j is one of the Y_i points in an attribute valued function.

Example:

	A	B
10	FUNCTION	Argument En or Mn
...	FN10	

4. FN_j is an operand in an arithmetic variable which is one of the Y_i points in an attribute valued function.

Example:

	A	B
10	FUNCTION	Argument En or Mn
...	V5...	
5	VARIABLE	... FN10...

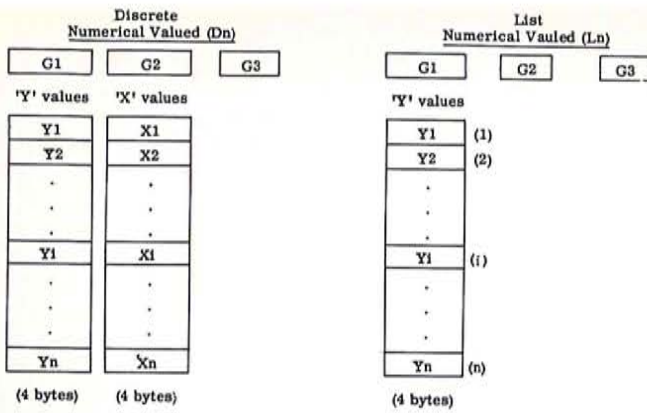


Figure 14. Core Allocation for Function Points

CORE ALLOCATION FOR FUNCTIONS

Each function entity requires eight basic words (32 bytes), whether or not it is actually defined by a function definition card and function follower card. In the Standard GPSS/360 program for a 128K machine, functions are numbered FN1, FN2, and so on, the eight basic words are set up for each function as follows:

G1 (4 bytes)			
G2 (4 bytes)			
G3 (4 bytes)			
G4 (2 bytes)		G5	G6
G7 (4 bytes)			
G8 (4 bytes)			
G9 (4 bytes)			
G10 (4 bytes)			

G5 & G6 are one byte each.

Table 6 shows how the eight basic words are used for each function entity.

The words G1, G2, and G3 serve as addresses to blocks of COMMON storage (Figure 14) in which the following values are stored:

G1. Y_i values for continuous, discrete and list functions.

G2. X_i values for continuous and discrete functions.

G3. Computed slopes (between adjacent points) of continuous functions.

Redefinition of Functions

If a function is redefined by another function definition card during a simulation job, the GPSS/360 input program will free the core used by previously defined function points first. Then the words that are necessary to store the redefined function points are obtained from the COMMON core blocks.

Increased Speed of List Functions

Figure 14 shows why the values of list functions can be computed more rapidly than discrete function values. For discrete functions the GPSS/360 program must compare the argument value (SNAK) through the sequence of successive X_i words until it finds $X_i < \text{Argument} \leq X_{i+1}$. The function value is then Y_{i+1} .

For list functions, the GPSS/360 program uses the argument value i as a direct index to the i^{th} word in the block of COMMON storage in which the Y_i values of the list function are stored. This method is much faster when compared to a comparable discrete function if many points are to be evaluated.

Random Number Generation

All random numbers used in GPSS/360 are calculated from a set of eight base numbers called seeds. The user can specify any one of these seeds (RN1-RN8).

Whenever a random number is requested, the following procedure is followed:

1. The specified seed is multiplied by a multiplier which is initially one.
2. The low-order 31 bits of this result are then stored for future use as a multiplier.
3. The seed index is computed from the high-order 16 bits and saved for future use.
4. The result of modulo division by 1000 is the normal random number 0 to 999. The fractional number used in FN_j is obtained by dividing this random number by 1000. Additional details appear in the IBM manual Random Number Generation and Testing.

NOTE: The random number seeds are not reset to their original value by either the CLEAR or RESET card. The JOB card, however, will reset all eight random number seeds.

TABLE 6: CORE ALLOCATION FOR FUNCTION ENTITIES

Symbol	Bytes	Quantity Stored	Source of the Quantity
G1	4	Address of 'Y' values	This field is set up at input time when the dependent variable Y_i values are read in. The block of storage is obtained from the chain of COMMON storage words.
G2	4	Address of 'X' values	This field is set up at input time when the independent variable values are read in if the function is continuous or discrete. The block of storage is obtained from the chain of COMMON storage words. If the function is list, this field will contain zeroes.
G3	4	Address of slope values	This field is set up at input time when the slopes are computed if the function is continuous. For other functions, this field will contain zeroes.
G4	2	Number of points in function	This field is set up at input time.
G5	1	bit 0 cyclic indicator bit 1 floating value indicator	This field is set up at execution time.
G6	1	Field B argument	This field is set up at input time and contains 'C', 'D', 'E', 'L', or 'M'.
G7	4	Field A SNA to be used as the independent variable	This field is set up at input time.
G8	4	Base for function entity words	This field is set up at execution time to save base address for function entity words.
G9	4	Return address	This field is set up at execution time to save return address to calling routine.
G10	4	Base register for variable, and matrix savevalue routines	This field is set up at execution time to save base register for variable, or matrix savevalue when the function routine is called from either.

CHAPTER 6: BLOCK ENTITIES

GENERAL NATURE OF BLOCK ENTITIES

Blocks and transactions are the two basic entities which form the basis for GPSS/360. Table 8 shows the card formats and coding symbols for the 43 distinct block types in the GPSS/360 program. Practically all of the status changes in a GPSS/360 simulation model result from the entry of transactions into blocks and the consequent execution of the GPSS/360 subroutine which is associated with each block type.

Three major status changes occur as the result of transactions entering blocks:

1. Transaction entities are created or destroyed by the block subroutine (transactions are the only temporary entities in the GPSS/360 program).

- a. Creating blocks - GENERATE, SPLIT
- b. Destroying blocks - TERMINATE, ASSEMBLE

NOTE: The CLEAR control card will also destroy all transactions (see Chapter 15).

2. The attributes, numerical or logical, of one or more entities are changed by the subroutine. This is the most common status change. The following types of attributes may be changed:

- a. The attributes of the transaction entering the block. All of the block types on transaction entities, discussed in Chapter 7, belong to this category.
- b. The attributes of other transactions are changed. These block types include SPLIT, ASSEMBLE, MATCH, GATHER, and TERMINATE. Also included are SEIZE, RELEASE, PREEMPT, RETURN, and LOGIC blocks which can affect the delay chain status of other transactions.
- c. The attributes of other entities are changed. Chapters 8-13 describe the block types which affect savevalue, logic switch, facility, storage, queue, and table entities.

The current transaction count (W_j) and total transaction count (N_j) of each block are also changed as transactions enter and move through a block (see "Standard Numerical Attributes for Block Entities" in this chapter).

Execution of Block-type Subroutines

All block-type subroutines (except for the GENERATE block) are executed when the transaction initially succeeds in entering the block. The block-type subroutine is never executed at any succeeding time. After this initial execution, the transaction either succeeds in moving directly into some next block or else it is delayed because it cannot enter a next block. This includes offspring transactions in a SPLIT block which are unconditionally created when the parent transaction enters the SPLIT block. (In the case of MATCH, ASSEMBLE, or GATHER blocks, transactions may be delayed until other SPLIT members of the assembly set allow it to attempt to move to a next sequential block).

All block types (except TEST, GATE, SEIZE, PREEMPT, and ENTER) will unconditionally allow an unlimited number of transactions to enter them. Under various conditions, TEST, GATE, SEIZE, PREEMPT, and ENTER blocks will refuse entry to transactions which are attempting to enter them from preceding blocks. This refusal will involve executing a major portion of the block subroutine, even though the transaction fails to enter the block.

GPSS/360 ASSEMBLY PROGRAM BLOCK DEFINITION CARDS

In the GPSS/360 assembly program, each block is defined by a block definition card with the following general format:

| 1 | 2 | LOCATION | 7 | 8 | OPERATION | 19 | A, B, C, D, E, F, G,

There are three basic fields on the input card: location field (columns 2-6), operation field (columns 8-18), and the operand field (columns 19-71). An asterisk (*) in column 1 indicates a remarks card, while a minus sign (-) indicates that the printout of the block count statistics is to be suppressed.

Symbols may be used for block locations in the location field. A symbol must consist of three to five alphanumeric, nonblank characters, the first three of which must be letters. The restriction is needed to avoid confusing assembly program block symbols with the mnemonic codes for Standard Numerical Attributes.

The field A, B, C, D, E, F and G-arguments which make up the Operand field are entered left-justified, starting in column 19, and are separated

by commas. If any leading or intermediate fields are to be left blank, this may be indicated by showing only the separating comma. For example:

PRINT , ,MOV has only a field C entry.

In this case, the first comma would be in column 19.

TRANSFER BOTH, ,QUEI has no field B entry.

The entire (operand) field, which starts in column 19 and must end by column 71, is terminated by the first blank encountered.

RELEASE 25 needs no commas in the (operand) field because the only entry is in field A.

Five block types require the specification of auxiliary operators or mnemonics in the operation field which may begin in column 13 for TEST, LOGIC and GATE, and in column 14 for the COUNT and SELECT blocks.

TEST Block

Auxiliary Relational Operator

L- Less than
LE- Less than or equal
G- Greater than
GE- Greater than or equal
E- Equal
NE- Not equal

LOGIC Block

Auxiliary Logic State

R- Reset
S- Set
I- Invert

GATE Block

Logical Attribute Mnemonic

M, NM- Match, No Match
U, NU- Facility In Use, Not In Use
I, NI- Facility Preempted, Not Preempted
SF, SNF- Storage Full, Not Full
SE, SNE- Storage Empty, Not Empty
LR, LS- Logic Switch Reset, Set

COUNT or SELECT Blocks

Auxiliary Relational Operator

L Less than
LE Less than or equal
G Greater than
GE Greater than or equal
E Equal
NE Not equal
MAX Maximum (SELECT block only)
MIN Minimum (SELECT block only)

Logical Attribute Mnemonic

U, NU, I, NI, SF
SNF, SE, SNE, LR, LS

CONTENTS OF BLOCK FIELDS

Each GPSS/360 block may have from zero to seven arguments, which are coded left to right in fields A, B, C, D, E, F, and G. Table 8 shows the allowable arguments in each field and the following basic conventions apply to all the block types described in that Table.

1. If a block field in Table 8 is blank, it is not interpreted by the simulator program
2. If a block field contains an argument which is not in brackets, the described information must be entered.

The items in fields A, B, C, D, E, F, and G of blocks can be:

- a. Any Standard Numerical Attribute (SNA_j or SNA*_n), or simply the constant k
 - b. Only the constant k
 - c. Special BCD characters, e.g., TRANSFER block selection modes (PICK, ALL, BOTH, etc.); BUFFER in PRIORITY block field B, etc.
 - d. Certain Standard Numerical Attributes, e.g., only k, X_j, XH_j, V_j, FN_j are allowed in the GENERATE block field B.
3. If a block field in Table 8 contains a description in brackets, the described information is optional, and a blank field will be interpreted according to conventions for that block type described in Chapters 7-13.
 4. If a block field in Table 8 contains an asterisk(*), an asterisk may be entered for indirect addressing. In this case, a transaction parameter number n must follow the asterisk to indicate which parameter will supply the argument value.

5. SNA means any of the Standard Numerical Attributes, with the convention that if no alphabetic characters precede the numerical part (other than *), a k will be assumed. Indirect addressing of constants is permitted. Thus, P6 and K*6 are equivalent. Note that P*6, however, implies a second level of indirect reference to Transaction Parameter n, whose index (n) is given by the value of Parameter 6.

GPSS/360 CORE ALLOCATION FOR BLOCK ENTITIES

Each GPSS/360 block has three words (twelve bytes) allocated to it (see Table 7). The standard GPSS/360 program for a 128k machine allows 500 blocks. If a block has only one variable field this field will be stored in the second of the three basic words (B2), otherwise additional words are obtained from the COMMON area of storage and the address of this area of contiguous bytes is placed in B2. The number of bytes obtained depends upon the number of fields specified and whether or not any of the fields is a matrix savevalue. If no matrix savevalue is specified in the variable field, the GPSS/360 program obtains four bytes per argument. Otherwise, twelve bytes are obtained for each argument of that block. Figure 15 outlines how the various block attributes are stored.

STANDARD NUMERICAL ATTRIBUTES FOR BLOCK ENTITIES

Each block has two externally addressable Standard Numerical Attributes which are shown in Figure 15.

1. W_j = Current number of Transactions at block j
2. N_j = Total number of transactions entering block j since last RESET or CLEAR card (or since start of simulation run if no RESET or CLEAR cards have been read in)

Each of these counts, W_j and N_j , is automatically accumulated by the GPSS/360 program. These counts have maximum values of $2^{15}-1$ (32,767) for the current count (W_j) and $2^{24}-1$ (16,777,215), for the total count (N_j).

Whenever the transaction succeeds in moving through a non-ADVANCE block into some next block without any blocking, only the total block count (N_j) of the non-ADVANCE block is incremented. The W_j and N_j counts, at the block where the transaction currently being processed by the GPSS program is located, do not include the current transaction. This is because the W_j and N_j counts are not incremented until:

1. The current transaction is blocked for the first time in entering some next block, or until:

TABLE 7: EFFECT OF RESET AND CLEAR CARDS ON BLOCK ATTRIBUTES

Word	Length	RESET Card	
		Attribute Value Before RESET Card	Result of RESET Card on Attribute Value
B1	4 bytes	Current Count (last two bytes)	Unchanged
B2	4 bytes	Address of COMMON where block arguments are stored	Unchanged
B3	4 bytes	Total Count (last three bytes)	Set to zero

CLEAR Card

Both the current count (last two bytes of word B1) and the total count (last three bytes of word B3) are set to zero. All other bytes remain unchanged.

SYMBOLIC
LOCATION

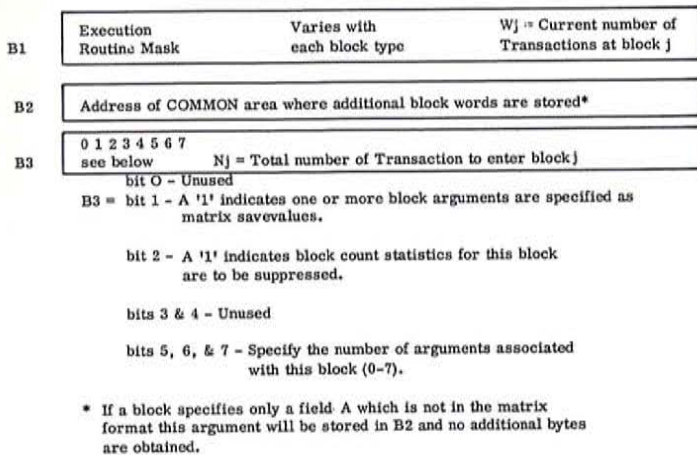


Figure 15. Core Allocation for Block Entities

2. A positive time is computed in the case of an ADVANCE block and the transaction is merged into the future events chain.

NORMAL BLOCK STATISTICAL OUTPUT

The normal statistical output at the end of a simulation run will contain block counts for all blocks, except those at which printing is specifically suppressed. A minus sign (-) in column 1 will suppress printing of the block counts. The GPSS/360 input program stores a 1 in bit 2 of the first byte of B3 to indicate suppression of the block counts printout to the output phase of the GPSS/360 program.

Shown below is a portion of a block count statistical output.

RELATIVE CLOCK	5028055	ABSOLUTE CLOCK	5028055					
BLOCK COUNTS				TOTAL	BLOCK CURRENT	TOTAL	BLOCK CURRENT	TOTAL
BLOCK CURRENT								
1	0	10002	11	0	10001	21	0	10000
2	0	10002	12	0	10001	22	0	10000
3	1	10002	13	0	10001	23	0	10000
4	0	10001	14	1	10001	24	0	10000
5	0	10001	15	0	10000	25	0	10000
6	0	10001	16	0	10000			
7	0	10001	17	0	10000			
8	0	10001	18	0	10000			
9	0	10001	19	0	10000			
10	0	10001	20	0	10000			

To facilitate the analysis of the block count statistical output, the blocks are divided into groups of up to 50 blocks each until all defined blocks are exhausted. These groups of 50 are further divided into five columns of ten sequential blocks each (i.e., blocks 1-10, 11-20, 21-30, . . . 51-60, 61-70, etc.). The CURRENT column is the current number (Wj) of transactions in the block, while the TOTAL column is the total number (Nj). In the example below there is currently one transaction at block 3 and one transaction at block 14.

In an error printout, the two block counts at the current block of the error-causing transaction will generally not include the error transaction, since this transaction has neither left the block nor been delayed.


Block Redefinition

Any GPSS/360 block except a GENERATE, MATCH, ASSEMBLE, or GATHER, may be redefined to any other block type with a block definition card for the second type. MATCH, ASSEMBLE, and GATHER blocks may be redefined in the same manner only if the current count (Wj) at the block is zero when the second block definition card is encountered. If the current count is not zero Input Error 282 will be given. A GENERATE block may be redefined only by another GENERATE block definition card. If the user attempts to redefine a GENERATE block to any other block type, Input Error 282 will be given. If there is a transaction currently at the GENERATE block waiting to enter the system, it is returned to the chain of inactive transactions and a new transaction is activated by the GENERATE block input routine using the block arguments of the new GENERATE block.

CHANGE and EXECUTE Blocks

There are two blocks which are associated solely with block entities: CHANGE and EXECUTE.

CHANGE Block

2	LOC	7	8	NAME	19	A	B
				CHANGE	From block no. k, SNAJ, SNA*n, *n Symbolic Block	To block no. k, SNAJ, SNA*n, *n Symbolic Block	

The CHANGE block provides a means of changing the blocks in a model during the course of a simulation run. The value of the field A argument is interpreted as a block number j. This block j is CHANGED to an identical copy of the block whose number (i) is given by the value of the field B argument. If the block indicated by field B is undefined, then the changed field A block will also become undefined. The field B block remains unchanged.

The current count, W_i, of the field A block is not changed. Only the total count, N_i, is set to zero.


Example:

5	ASSIGN	9	V6
30	CHANGE	5	60
60	SAVEVALUE	20	V7

In the above example, block 5 (ASSIGN) will be changed to a SAVEVALUE (block 60) with a field A argument of 20 and a field B argument of V7. The current count of block 5 will remain the same, and the total count will be set to zero.

In the case of a CHANGE block error, the GPSS/360 program will give the error statistical printout along with Execution Error 698, "Illegal Change in Change Block".

EXECUTE Block

2	LOC	7	8	NAME	19	A
				EXECUTE	Block no. j k, SNAJ, SNA*n, *n Symbolic Block	

The EXECUTE block allows the entering transaction to perform the operation of any other specific block without diverting the transaction from its normal sequential flow (see exceptions below). The value of the EXECUTE block field A is interpreted as a block number j and the operation of that block is performed as though the transaction had actually entered it. The transaction then proceeds sequentially from the EXECUTE block.

It is important to note that the transaction will be delayed in the EXECUTE block, not in the block executed, whenever:

1. The executed field A block is an ADVANCE block with a nonzero time delay.
2. The executed field A block is a SEIZE block, and during the same clock time that the SEIZE is executed, another transaction PREEMPTS the same facility.
3. The executed block is a MATCH block and a matching transaction is not found at its conjugate MATCH block.
4. The executed block is an ASSEMBLE or GATHER block and the specified number of transactions has not yet been assembled or gathered.

Thus, for example, a transaction which executes a MATCH block and is delayed because no match exists initially, will proceed only when another member of its assembly set attempts to MATCH a transaction at the EXECUTE block (not the executed MATCH block).

Exceptions:

1. Execution of a block which does not direct transactions to the next sequential block (TRANSFER, or LOOP with the looping parameter

not decremented to zero) will cause the transaction to transfer or loop, and it will not proceed sequentially from the EXECUTE block.

2. Execution of a GENERATE block is not

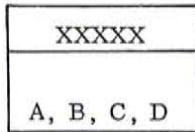
permitted and will cause a simulator error if attempted.

Internally, the EXECUTE block operates as if its core words are the same as core words of the block whose number j is specified in the field A.

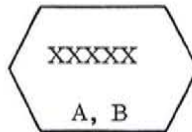
TABLE 8: GPSS/360 BLOCK FORMATS AND SYMBOLS

The following pages describe the format of the block types used in GPSS/360 and their corresponding block symbols. Reference is given to the page on which the operation of each block type is discussed.

If the user wishes a more convenient and less time-consuming method of developing block diagrams, use of the five basic blocks shown below is suggested. Below each block symbol appears a list of the block types found on the following pages which each symbol represents.



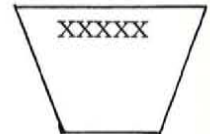
- ADVANCE
- ASSEMBLE
- BUFFER
- CHANGE
- COUNT
- DEPART
- ENTER
- EXECUTE
- GATHER
- JOIN
- LEAVE
- LINK
- LOGIC
- MATCH
- MSAVEVALUE
- QUEUE
- PREEMPT
- RELEASE
- REMOVE
- RETURN
- SEIZE
- SELECT
- SPLIT
- TABULATE
- UNLINK



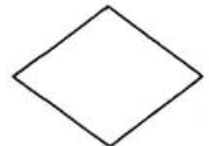
- ALTER
- ASSIGN
- INDEX
- LOOP
- MARK
- PRIORITY
- SAVEVALUE



- GENERATE
- TERMINATE


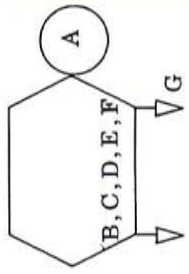
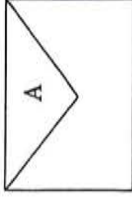
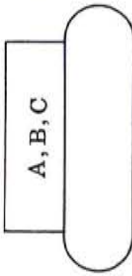
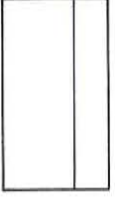
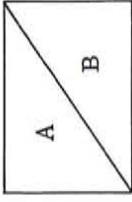


- PRINT
- TRACE
- UNTRACE
- WRITE

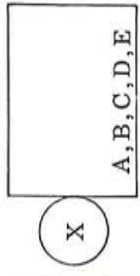
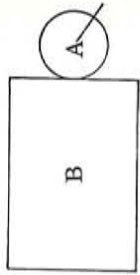

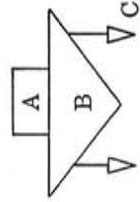
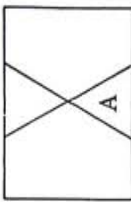


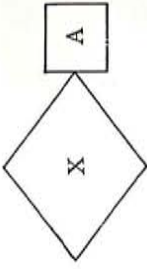
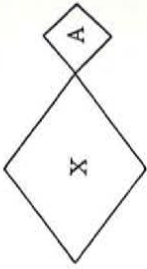
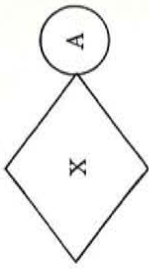
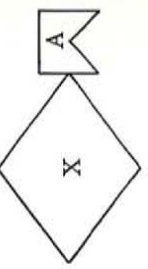
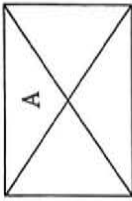
- EXAMINE
- GATE
- SCAN
- TEST
- TRANSFER

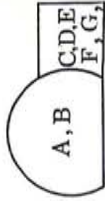
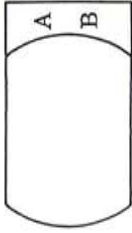
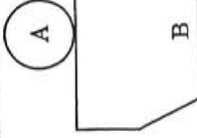
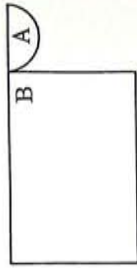
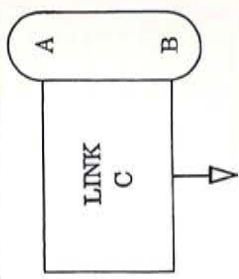
TABLE 8 (Continued)



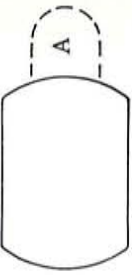
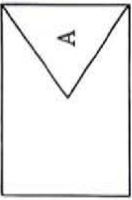
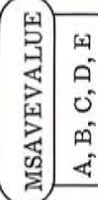
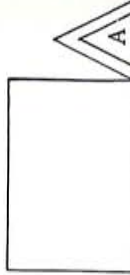
OPERATION	A	B	C	D	E	F	REFERENCE	BLOCK SYMBOL
ADVANCE	mean [SNA (*)]	modifier or spread [SNA(*)]					page 60	
ALTER	Group no. SNA(*)	count SNA(*), ALL	attribute Pn, PR	argument SNA(*)	attribute Pn, PR	[argument SNA (*)]	page 119	
ASSEMBLE	no. to assemble SNA (*)						page 88	
ASSIGN	parameter no. SNA (≠)	source integer SNA (*)	Function Modifier [SNA (*)]				page 78	
BUFFER							page 97	
CHANGE	'from' block no. SNA (*)	'to' block no. SNA (*)					page 42	

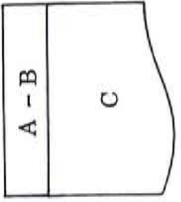
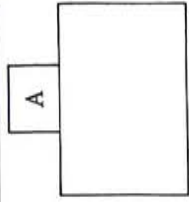
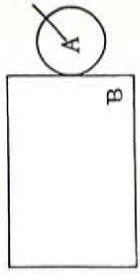
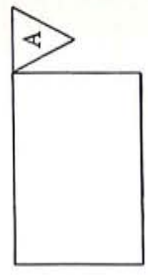
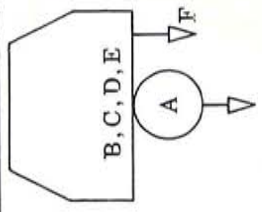
[field G specifies an optional next block]


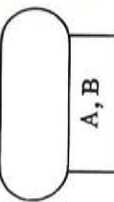
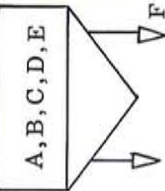


OPERATION	A	B	C	D	E	F	REFERENCE	BLOCK SYMBOL
COUNT 'X'	parameter no. SNA (*)	lower limit SNA (*)	upper limit SNA (*)	match argument [SNA (*)]	SNA mnemonic to be counted		page 81	
DEPART	Queue no. SNA (*)	units [SNA (*)]					page 170	
ENTER	Storage no. SNA (*)	units [SNA (*)]					page 157	
EXAMINE	Group no. SNA (*)	numeric quantity [SNA (*)]	alternate exit SNA (*)				page 117	
EXECUTE	Block no. SNA (*)						page 42	

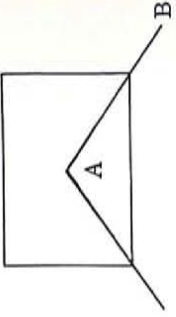
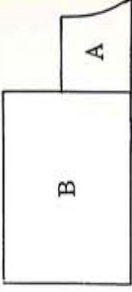
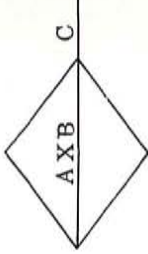
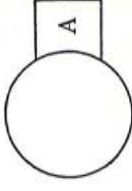
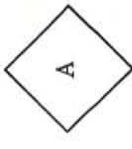
OPERATION	A	B	C	D	E	F	REFERENCE	BLOCK SYMBOL
(X) GATE LS LR	Logic Sw. no. SNA (*)	Next block if condition is False [SNA (*)]					page 131	
(X) GATE I NU U	Facility no. SNA (*)	Next block if condition is False [SNA (*)]					page 150	
(X) GATE SE SF SNE SNF	Storage no. SNA (*)	Next block if condition is False SNA (*)					page 159	
(X) GATE M NM	(match) Block no. SNA (*)	Next block if condition is False [SNA (*)]					page 94	
GATHER	no. to gather SNA (*)						page 91	

OPERATION	A	B	C	D	E	F	REFERENCE	BLOCK SYMBOL
GENERATE	[mean] Note 1:	[modifier] Fields A-F may be a constant or FNj, Vj, Xj or XHj. Field G may only be blank, 'F' or 'H' to specify parameter type	[offset]	[count]	[priority]	[no. parameters]	page 66	
INDEX	parameter no. SNA	increment SNA					page 80	
JOIN	Group no. SNA (*)	numeric quantity [SNA (*)]					page 115	
LEAVE	Storage no. SNA (*)	no. of units [SNA (*)]					page 158	
LINK	Chain no. SNA (*)	ordering of chain LIFO, FIFO Pj	alternate Block exit [SNA (*)]				page 70	

OPERATION	A	B	C	D	E	F	REFERENCE	BLOCK SYMBOL
(X) S R I LOGIC	Logic Sw. no. SNA (*)						131	
LOOP	parameter no. SNA (*) no.	next block B SNA (*) no.					83	
MARK	[parameter no. SNA (*) no.]						81	
MATCH	conjugate block SNA (*) no.						93	
MSAVE - VALUE	Matrix no. SNA (*) [±]	row no. SNA (*)	column no. SNA (*)	value SNA (*)	[H]		122	
PREEMPT	Facility no. SNA (*) no.	['PR']	[Block no.]	[parameter] no.	['RE']		142	

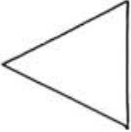
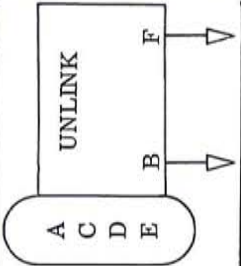
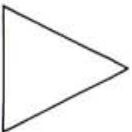
OPERATION	A	B	C	D	E	F	REFERENCE	BLOCK SYMBOL
PRINT	lower limit [SNA (*)]	upper limit [SNA (*)]	Entity mnemonic	[paging indicator]			page 187	
PRIORITY	priority SNA (*)	[BUFFER]					page 96	
QUEUE	Queue no. SNA (*) no.	units [SNA (*)]					page 168	
RELEASE	Facility no. SNA (*)						page 140	
REMOVE	Group no. SNA (*)	count [SNA (*), ALL]	numeric quantity [SNA (*)]	Transaction attribute [Pn, PR]	matching value [SNA (*)]	alternate exit [SNA (*)]	page 116	

OPERATION	A	B	C	D	E	F	REFERENCE	BLOCK SYMBOL
RETURN	Facility no. SNA (*)						page 148	
SAVEVALUE	Savevalue number SNA (*) no. [±]	attribute SNA (*)	[H]				page 120	
SCAN	Group no. SNA (*)	Transaction attribute Pn, PR	match argument SNA (*)	desired attribute Pn, PR	parameter no. SNA (*)	alternate exit [SNA (*)]	page 118	
SEIZE	Facility no. SNA (*)						page 140	
SELECT 'X'	parameter no. SNA (*)	lower limit SNA (*)	upper limit SNA (*)	match argument SNA (*)	SNA mnemonic	alternate exit [SNA (*)]	page 83	

OPERATION	A	B	C	D	E	F	REFERENCE	BLOCK SYMBOL
SPLIT	no. copies SNA (*)	next block B SNA (*)	parameter for serial numbering [SNA (*)]	number of parameters [SNA (*)]			page 87	
TABULATE	Table no. SNA (*) no.	units SNA (*)					page 177	
TEST (X) E NE GE LE G L	ARG 1 SNA (*)	ARG 2 SNA (*)	next block B [SNA (*)]				page 84	
TERMINATE	units [SNA (*)]						page 69	
TRANSFER	selection mode	next block A SNA (*) ¹	next block B [SNA (*)] ¹	[indexing factor]			page 62	

¹With ALL selection mode direct specification must be used.

TABLE 8 (Continued)

OPERATION	A	B	C	D	E	F	REFERENCE	BLOCK SYMBOL
TRACE							page 187	
UNLINK	User Chain no.	next block A	unlink count SNA(*) ['ALL']	parameter no. SNA (*) ['BACK']	[match argument]	[next block B]	page 72	
UNTRACE							page 187	
WRITE	Jobtape no.						page 109	

CHAPTER 7: TRANSACTION ENTITIES

Some of the basic attributes of transaction entities such as the transaction parameters and priority level have been described in Chapter 2. As was previously mentioned practically all of the status changes in GPSS/360 models occur as the result of transactions entering blocks, and the consequent execution of the GPSS/360 subroutines associated with these blocks. This chapter describes the attributes transformed by the following block types:

ADVANCE, TRANSFER, GENERATE,
TERMINATE, SPLIT, ASSEMBLE, GATHER,
MATCH, PRIORITY, BUFFER, ASSIGN,
INDEX, MARK, GATE, TEST, LOOP, LINK,
UNLINK

S/360 CORE ALLOCATION FOR TRANSACTION ENTITIES

Experience has shown that the systems analyst can use all the power and flexibility of GPSS/360 only when he understands all the basic attributes of transactions and how they are changed in the course of a simulation run.

Each transaction in the GPSS/360 program consists of four contiguous basic control words (T1 - T4). These four words are allocated regardless of the status of the transaction, i.e., inactive, future event chain, current event chain, etc. When a transaction becomes activated additional contiguous bytes are required to carry transaction information. This includes 20 bytes for transaction words T5 - T14 plus additional bytes for transaction parameters.

The number of bytes obtained for parameters is dependent upon the number and type of parameters (two bytes/halfword parameter and four bytes/fullword parameter). The address of this contiguous area is stored in T3 of the basic transaction words.

Table 9 describes in detail the contents of both the basic and additional transaction words.

STANDARD NUMERICAL ATTRIBUTES OF TRANSACTION ENTITIES

Of the many transaction attributes contained in each set of transaction words, four attributes are externally addressable as Standard Numerical Attributes.

Transaction Parameter - Pn, P*n, *n

Each transaction in the standard GPSS/360 program has 0 to 100 fullword or halfword parameters whose values are stored as signed integers. The Standard Numerical Attribute Pn references the value of Parameter n of the transaction currently being

processed by the GPSS/360 program. P*n is the value of the parameter whose number m is given by the value of Parameter n. The parameter values of transactions other than the transaction currently being processed cannot be addressed directly in a GPSS/360 model. It is often necessary in GPSS/360 models for one transaction to store its parameter values in savevalue locations so that a second transaction can directly reference the savevalues (Xj) and thereby indirectly reference the parameter values (Pn) of the first transaction.

Parameter values can be modified by ALTER, ASSIGN, COUNT, INDEX, LOOP, MARK, SCAN, SELECT, and SPLIT Blocks.

Transaction Transit Time—M1

Transaction word T6 stores a positive 31 bit mark time. The Standard Numerical Attribute M1 is the transaction transit time of the transaction currently being processed by the GPSS/360 program. This transit time is computed in the following manner:

$$M1 = \text{Transaction Transit Time Absolute Clock Time} - \text{Mark Time (T6) of the transaction currently being processed}$$

When a transaction is created in a GENERATE block the value of the absolute clock is placed in the T6 mark time word. Whenever the transaction enters a MARK block with a blank field A (see "MARK block" later in this chapter) the current absolute clock time is placed in the mark time word.

The Standard Numerical Attribute (M1) therefore, measures the transit time of the transaction currently being processed as either:

1. the time interval from its creation in a GENERATE block until the current absolute clock time, or
2. the time interval from the last time it moved through a MARK block with a blank field A until the current absolute clock time.

The transaction transit time, M1, is the most common distribution table argument, since one of the key performance characteristics in many systems is the transit time of transactions through the system.

The transit time of transactions other than the one being currently processed cannot be addressed directly. Once again, it may be necessary to store these M1 values in intermediate savevalue locations so that they can be referenced by other transactions.

Parameter Transit Time - MPn, MP*n

The MARK block field A can specify a Transaction Parameter number n. Transactions which pass

TABLE 9: S/360 CORE ALLOCATION FOR TRANSACTION ENTITIES

Basic Transaction Words (T1-T4):

T ¹ (4 bytes)	Previous Transaction (number) in Future Events Chain	Next Transaction (number) in Future Events Chain
T ² (4 bytes)	Previous Transaction (number) in Current Event Chain	Next Transaction (number) in Current Event Chain
T ³ (4 bytes)	Address of Common Area where T5 begins	
T ⁴ (4 bytes)	Time Transaction is scheduled to leave Future Event Chain	

Additional Words Obtained from COMMON Area (T5-T15):

T ⁵ (4 bytes)	Next Block for Transaction	Next member of Assembly Set
T ⁶ (4 bytes)	MARK TIME	

T7 (1 byte)	Priority	T8 (1 byte)	Preempt Count	T9 (1 Byte)	0 1 2 3 4 5 6 7 (see below)
----------------	----------	----------------	---------------	----------------	--------------------------------

- T9 = bit 0 - A '1' indicates fullword Parameters
 bit 1 - A '1' indicates Transaction is in Multiple Queues
 bit 2 - Tracing Indicator
 bit 3 - Unused
 bit 4 - A '1' indicates Transaction is a member of a Group
 bit 5 - A '1' indicates Transaction in Future Event Chain
 bit 6 - A '1' indicates Transaction in Current Event Chain
 bit 7 - A '1' indicates Transaction in Interrupt Status

T10 (1 byte)	0 1 2 3 4 5 6 7 (see below)	T11 (1 byte)	Number of Parameters	T12 (1 byte)	(see below)
-----------------	--------------------------------	-----------------	----------------------	-----------------	-------------

- T10 = bit 0 - Scan Status Indicator
 bit 1 - TRANSFER SIM Mode Delay Indicator
 bits 2 & 3 - Selection Factor (2=BOTH,=3 ALL)
 bit 4 - A '1' indicates last block for TRANSFER ALL
 bit 5 - Matching Indicator
 bit 6 - Preempt Status Indicator
 bit 7 - Preempt Flag

T12 Number of bytes obtained for xact words T5-T14 and Parameters.

T13 (2 bytes)	Queue Number (if in a Queue)
------------------	------------------------------

T14 (4 bytes)	Time Transaction Last Entered Queue
------------------	-------------------------------------

T15	Begin Parameters
-----	------------------

through such a MARK block will have the absolute clock time stored in Parameter n. Subsequently, the Standard Numerical Attribute, MPn, is computed as the parameter transit time of the transaction currently being processed by the GPSS/360 program in the following manner:

MPn=Parameter=Current-Value of Parameter
 Transit Absolute n of the Transaction
 Time Clock currently being
 Time processed

It is advisable to use a fullword parameter for this use since the value of the absolute clock might easily exceed the maximum value allowed in a halfword parameter (32, 767).

Transaction Priority — PR

Each GPSS/360 transaction carries in byte T7 its assigned priority level. This value (0-127) is assigned in a GENERATE block and may be modified by a PRIORITY block. When the Standard Numerical Attribute PR is referenced the program obtains the priority of the currently active transaction.

STANDARD LOGICAL ATTRIBUTES OF TRANSACTION ENTITIES

There are two Standard Logical Attributes associated with transactions whose true and false values are used in GATE M and NM blocks to control the flow of transactions.

1. Mj is true if another member of the assembly set of the transaction currently being processed in a GATE M block is in a "matching" condition at block number j. Block j should be a MATCH, ASSEMBLE, or GATHER block.

2. NMj is true if no other members of the assembly set of the transaction currently being processed at a GATE NM block are in a "matching" condition at block number j. Block j should be a MATCH, ASSEMBLE, or GATHER block.

A detailed explanation of these Logical Attributes is given in "GATE M and GATE NM Blocks" later in this chapter.

TRANSACTION PRINTOUT

Each transaction currently active in the system will appear in one of the following five sections of the transaction printout:

CURRENT EVENTS CHAIN
 FUTURE EVENTS CHAIN
 USER CHAINS

INTERRUPT TRANSACTIONS
 TRANSACTIONS IN A MATCHED STATUS

Table 10 shows a sample transaction printout from a GPSS/360 simulation model. Table 10 describes the contents of the various columns which are printed out. Knowledge of the transaction printout is essential for efficient use of GPSS/360, particularly in debugging the inevitable errors which occur in building a simulation model.

Current Events Chain

Each of the transactions in the current events chain is printed out in order of descending priority. This is also the same order in which they are linked in the chain, as outlined in Figure 16. T7 of each transaction contains the Priority Level (0, 1, . . . 127). Word T2 is used to create the actual linkages in the chain. The first two bytes of word T2 contain the number of the transaction which precedes this one on the chain, while the second two bytes of T2 contain the number of the transaction which follows this one on the chain. The overall GPSS/360 scan, described later in this chapter, processes transactions in the order in which they appear through these current events chain linkages.

A priority class table of 128 words is associated with the current events chain (see Figure 16). Each word represents a specific priority class (0, 1, 127) of transactions and contains the following information. The first two bytes contain a count of the number of transactions on the chain in that priority class. The last two bytes of each word contain the number of the last transaction in the particular priority class. Whenever a transaction is added to the current events chain (see Table 11) it becomes the last transaction in its priority class. Each transaction on the current events chain has a 1 in bit 6 of byte T9. This is printed as a 2 in the "CI" column of the transaction printout.

Future Events Chain

The transactions in the future events chain are printed out in order of ascending block departure times from the various ADVANCE and GENERATE blocks in which they are delayed. These times, which are printed in the "BDT" column, are all greater than the current absolute clock time which is printed preceding the transaction printout. The transactions are printed out in the same order as their linkages in the future events chain, as outlined in Figure 17. Word T4 contains the ADVANCE or GENERATE block departure times.

Word T1 is used to create the actual linkages in the future events chain, in the same manner as the current events chain. Each future event chain transaction has a '1' in bit 5 of Byte T9. This is indicated by a "4" in the "CI" column of the transaction printout.

TABLE 10: DESCRIPTION OF TRANSACTION PRINTOUT

CURRENT TRANS	EVENTS BDT	CHAIN BLOCK	PR	SF	NBA	SET	MARK-TIME	P1	P2	P3	P4	SI	TI	DI	CI	MC	PC	PF
1	1	3			4	1	1					1		1	2			
3	2	3			4	3	2	0000	3000	0000	0000	0001		1	2			
4	3	3			4	4	3	0000	0000	0000	0000	0001		1	2			
5	4	3			4	5	4	0000	0000	0000	0000	0001		1	2			
6	5	3			4	6	5	0000	1200	0000	0000	0001		1	2			
7	6	3			4	7	6	0000	0000	0000	0000	0001		1	2			
8	7	3			4	8	7	0000	1800	0000	0000	0001		1	2			
9	8	3			4	9	8	0000	0000	0000	0000	0001		1	2			
10	9	3			4	10	9	0000	2400	0000	0000	0001		1	2			
11	10	3			4	11	10	0000	0000	0000	0000	0001		1	2			
12	11	3			4	12	11	0000	2700	0000	0000	0001		1	2			
13	12	3			4	13	12	0000	0000	0000	0000	0001		1	2			
14	13	3			4	14	13	0000	3000	0000	0000	0001		1	2			
15	14	3			4	15	14	0000	0000	0000	0000	0001		1	2			
16	15	3			4	16	15	0000	3300	0000	0000	0001		1	2			
17	16	3			4	17	16	0000	0000	0000	0000	0001		1	2			
18	17	3			4	18	17	0000	3600	0000	0000	0001		1	2			
19	18	3			4	19	18	0000	0000	0000	0000	0001		1	2			
20	19	3			4	20	19	0000	3900	0000	0000	0001		1	2			
21	20	3			4	21	20	0000	0000	0000	0000	0001		1	2			
								0000	4200	0000	0000	0001		1	2			
								0000	0000	0000	0000	0001		1	2			
								0000	4500	0000	0000	0001		1	2			
								0000	0000	0000	0000	0001		1	2			
								0000	4800	0000	0000	0001		1	2			
								0000	0000	0000	0000	0001		1	2			
								0000	5100	0000	0000	0001		1	2			
								0000	0000	0000	0000	0001		1	2			
								0000	5400	0000	0000	0001		1	2			
								0000	0000	0000	0000	0001		1	2			
								0000	5700	0000	0000	0001		1	2			
								0000	0000	0000	0000	0001		1	2			
								0000	6000	0000	0000	0001		1	2			
								0000	0000	0000	0000	0001		1	2			
								0000	0000	0000	0000	0001		1	2			

TABLE 10: (Continued)

TRANS	Transaction number j.
BDT	Block Departure Time. This represents either the absolute clock time at which the transaction is scheduled to leave an ADVANCE block (Future Event Chain) or the time at which it last left an ADVANCE block (any other chain). BDT is zero for transactions created at a SPLIT block until they enter an ADVANCE block.
BLOCK	The number of the block at which the transaction is currently located. If the transaction is on the Future Events Chain or a User Chain the T1 word is used for transaction linkages and the current block is kept in the last two bytes of word T2 (T2+2). If the transaction is on the Current Events Chain or in an Interrupt or Matching status, the T2 word is used for transaction linkages or the Assembly or Gather Count, thus, the current block for the transaction is kept in the T1 word (T1+1).
PR	Priority Level of Transaction (0, 1, . . . 127). This is stored in byte T7.
SF	Selection Factor. Indicates the type of next block trial to be made. This is stored in bits 2 and 3 of T10. (blank) = indicates only one next block to be tried. B = indicates that the current block (given in the BLOCK column) is a TRANSFER block with a BOTH selection mode. The second block is available only internally in T1. A = indicates that the current block is a TRANSFER block with an ALL selection mode. The NBA column lists the lowest of the n next blocks to be tried. The highest block number is available internally in T1.
NBA	The next block to be entered by the transaction. This is stored in the first 2 bytes of word T5.
SET	Assembly Set Linkage. When the transaction is created, the set number is equal to the transaction number itself. A linkage is formed whenever the transaction enters a SPLIT block. This linkage is updated when any member of an assembly set enters a SPLIT or TERMINATE block. This is stored in the second 2 bytes of T5.
MARK TIME	When the transaction is created the value of the absolute clock is placed in word T6. When the transaction enters a MARK block with a blank field A the current value of the absolute clock is entered in word T6.
P1, . . . P4 P5, . . . P8 . . .	The first line of printout for each Transaction lists the current value of Parameters 1 through 4. The second line lists the values of Parameters 5 through 8 where P5 is in the P1 column, P6 is in the P2 column, etc.
SI	Scan Status Indicator. This is stored in Bit 0 of T10. A one indicates that a next block trial is to be suppressed until the condition causing the delay changes. The bit is reset to zero for all transactions in the Delay Chain when the blocking condition changes.
TI	Tracing Indicator. This is stored in Bit 2 and 3 of T9. A one indicates that the transaction has entered a TRACE block. The bit is reset to zero by an UNTRACE block.
DI	Delay Indicator. This is stored in Bit 1 of T10. A "one" indicates that the transaction has failed to move directly into some next block. It is used with the TRANSFER block in the SIM selection mode which resets the delay indicator. The indicator is also reset when the transaction leaves an ADVANCE block.
CI	Chain Indicator. This is stored in T9. "Zero" indicates transaction is in Matching Status or on a User's Chain. "1" indicates the transaction is in an Interrupt Status.

TABLE 10: (Continued)

- CI (Cont) "2" indicates the transaction is in Current Events Chain.
 "4" indicates the transaction is in Future Events Chain.
- MC Matching Condition. This is stored in Bit 5 of T10. A "four" indicates that the transaction is available for MATCHing, or is at an ASSEMBLE or GATHER block in the process of being assembled or gathered; otherwise the column is blank.
- PC Preempt Count. This is stored in byte T8. This field is incremented by one whenever the transaction is PREEMPTed at one of the facilities which it has SEIZED or PREEMPTed. It is decremented by one whenever the preempt condition is removed.
- PF Preempt Flag. This is stored in Bit 7 of T10. A "one" indicates that the transaction is to be PREEMPTed when it enters the next ADVANCE block that specifies a nonzero time. The transaction will also be PREEMPTed if it is delayed at a MATCH, GATHER, or ASSEMBLE block. The indicator is reset to zero when the transaction is PREEMPTed.

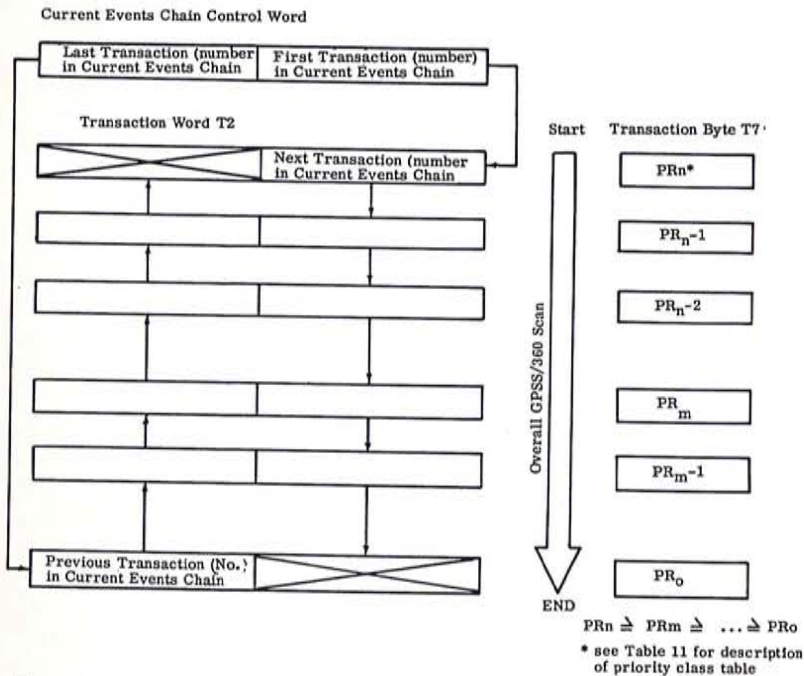
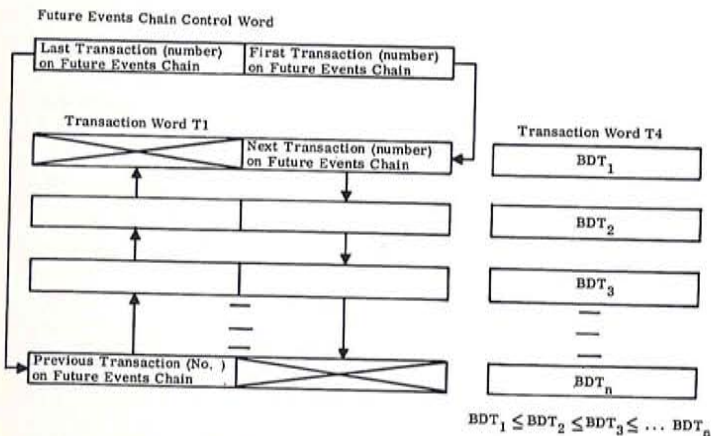


Figure 16. Organization of Current Events Chain



Each j th transaction which enters a positive-time ADVANCE block or is created in a GENERATE block is merged into the Future Events Chain from the remote end so that $BDT_i \leq BDT_j < BDT_{i+1}$. Consequently, if two transactions have the same BDT, the first one merged will precede the other on the Future Event Chain. When the clock is updated to their BDT they will be moved to the Current Events Chain in the same order.

Figure 17. Organization of Future Events Chain

TABLE 11: INFORMATION IN PRIORITY CLASS TABLE ASSOCIATED WITH CURRENT EVENTS CHAIN

Each of the 128 entries (0-127) in the Priority Class Table is made up of two two-byte fields. The first two bytes in each word represent the total number of transactions in the Particular Priority Class. This count is incremented by one when a transaction of that particular priority:

1. is added to the Current Events Chain,
2. has its Priority Level changed in a PRIORITY block to the particular priority,
3. is created at a GENERATE or SPLIT block and put on the Current Events Chain.

This count is decremented by one when a transaction of that particular Priority Level:

1. is switched from the Current Events Chain to the Future Events Chain in a positive-time ADVANCE block,
2. is switched from the Current Events Chain to an Interrupt or Matching status in a MATCH, GATHER, or ASSEMBLE block,
3. has its Priority level changed to another level in a PRIORITY block,
4. is Terminated in a TERMINATE block,
5. is switched from the Current Events Chain to a User's Chain.

The second two bytes in each class are reserved for the number of the last transaction in that particular priority class. This number is changed each time a new transaction is added to a particular priority class or the last transaction is removed from the Current Events Chain. This can occur under the same three conditions described above for incrementing the count.

PRIORITY CLASS TABLE

Priority Level		
127	Total number of Transactions currently in Priority Class 127	Number of last transaction in Priority Class 127
126	Total number of Transactions currently in Priority Class 126	Number of last transaction in Priority Class 126
.	.	.
.	.	.
.	.	.
1	Total number of transactions currently in Priority Class 1	Number of last transaction in Priority Class 1
0	Total number of transactions currently in Priority Class 0	Number of last transaction in Priority Class 0

User Chains

The third part of the transaction printout consists of transactions placed on user chains by the analyst. The chains are printed in numerical order. The transactions in each chain are printed in the order the analyst merged them on the chain. With the exceptions that the NBA is blank and the "CI" column is blank, the transaction has the same information as the current, future, and interrupt Chains.

Interrupt Chain

The fourth part of the transaction printout consists of those transactions which have been PREEMPTed on one or more facilities which they have SEIZED or PREEMPTed. These transactions are printed out in order of their transaction numbers. These transactions also contain the same information as the previous chains except that the "CI" column contains a 1.

Matching Chain

The final transaction printout consists of those transactions which are in a "matching" condition at MATCH, ASSEMBLE, or GATHER blocks. These transactions are identified by their matching indicator (bit 5 of T10) being on. This is indicated in the printout by a 4 in the MC column. This chain is also ordered on transaction number.

ADVANCE BLOCK, IN WHICH TRANSACTIONS ARE DELAYED FOR POSITIVE TIME

2	LOC	7	8	OPERATION	19	A	B
				ADVANCE		Mean Time k, SNAj, SNA*n	Spread k, *n, SNA, SNA*n Function modifier FNj, FN*n

ADVANCE
A,B

ADVANCE blocks never refuse entry to transactions. Transactions move to the next sequential block. The ADVANCE block is the means by which transactions may be delayed in the course of their progress through a GPSS/360 block diagram. The ADVANCE block will compute the interval for which the entering transaction is to remain at the block. This time may have any integral value, including zero. If the time computed is zero, the GPSS/360 simulator will continue processing the entering transaction, and will immediately attempt to move it to the next sequential block. The delay times in the ADVANCE block are specified as described below.

Mean Time (Field A)

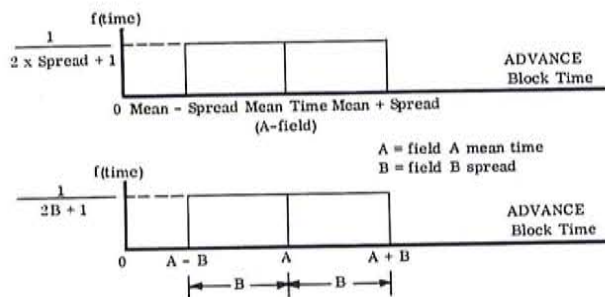
The value of the field A argument (k, *n, SNAj, SNA*n) is taken as the mean (or basic) block time. Constant values cannot exceed 999999. The values of Standard Numerical Attributes (SNAj, SNA*n), however, are restricted only by their own ranges. "Various Ways to Specify Constant Values" in Chapter 2, for example, shows how an INITIAL card can be used to load a large constant into a

savevalue location, whose value X_j can then be referenced as an ADVANCE block mean time.

Spread Time or Function Modifier (Field B)

The basic block time can be modified in one of two ways by a field B argument:

1. A spread constant ($k, *n, SNA_j, SNA*n$) which must not be greater than the computed mean time, will define a uniform or equiprobable distribution of integer delay times (see below). Each of the $(2B+1)$ integer times between $A-B$ and $A+B$ will be chosen with equal probability: $1/(2B+1)$. For example, a spread of 5 used with a mean of 10 will produce block times in the range 10 plus or minus 5 (i.e., from 5 to 15). Only one of the possible integer values would be chosen for each transaction which enters the ADVANCE block. Each integer in the range (5-15) will be computed with equal probability ($1/11$), including the endpoints, 5 and 15. Fractional action times are excluded since the simulator clock only assumes integral values. The spread must not be greater than the mean; otherwise, negative ADVANCE block times may result and Execution Error 530 will occur (spread exceeds mean in time computation).



2. Field B function modifier ($FN_j, FN*n$) will cause the computed value of the SNA specified in field A to be multiplied by the value of the function specified in field B. The function value is not truncated to an integer before the multiplication. Consequently, ADVANCE block mean times can be modified by fractional valued functions such as the exponential function described in Chapter 5.

Transactions That are PREEMPTed While in an ADVANCE Block

When a transaction has been merged into the future events chain, after entering an ADVANCE block, another transaction may enter a PREEMPT block

which references a facility that the transaction in the ADVANCE block has SEIZED. The SEIZING transaction in the ADVANCE block can be in either of the following two states:

1. It is being PREEMPTed on the first facility which it has SEIZED.

2. It has already been PREEMPTed on another facility which it has SEIZED.

If the ADVANCE block transaction has not been PREEMPTed the following steps occur (see Case 3B in "PREEMPT Block" in Chapter 10 for further details):

1. The remaining time which the transaction is to spend in the ADVANCE block is stored in the block departure time word T4, where:

$$\begin{array}{rcl} \text{Remaining} & \text{Block} & \text{Current} \\ \text{ADVANCE} & = \text{Departure} & - \text{Absolute} \\ \text{Block Time} & \text{Time (word T4)} & \text{Clock Time} \end{array}$$

2. The transaction is unlinked from the future events chain.

3. The seventh bit of byte T10 (Preempt Status Indicator) is set to indicate that the transaction is now in an interrupted status.

4. The eighth bit of byte T9 is set to indicate that the transaction is now on the interrupt chain.

5. The preempt count, T8, is set to one.

The section on the "RETURN Block" in Chapter 10 describes how these transactions are removed from a PREEMPTed interrupt status and are merged back into the future events chain. It is also possible for a transaction to be PREEMPTed on facilities which it has previously PREEMPTed. A detailed description of this feature is given in Chapter 10.

An ADVANCE block transaction which has already been PREEMPTed on one facility and placed in an interrupt status, can be PREEMPTed on as many as 255 more facilities which it has also SEIZED. The preempt count (in transaction location T8) is incremented by one each time that a further PREEMPT occurs (Case 3A in "PREEMPT Block" in Chapter 10 gives further details).

Transactions That Enter ADVANCE Blocks

A transaction may be in the current events chain when another transaction enters a PREEMPT block which references a facility that the first transaction has SEIZED. The seizing transaction will not be immediately removed from the current events chain and placed into an interrupt status. Instead, a preemption flag (eighth bit of byte T10) is set to one. The SEIZING transaction will be put into an interrupt status only when it enters a nonzero-time ADVANCE block, or when it enters a MATCH, ASSEMBLE, or GATHER block in which it is placed into a "matching" condition.

When such a SEIZEing and PREEMPTed transaction enters an ADVANCE block the following steps occur (see Case 3D in "PREEMPT Block" in Chapter 10 for further details):

1. The nonzero ADVANCE block time is stored in the T4 block departure time word. The transaction will spend this time in the ADVANCE block only after the preempt count of the transaction has been decremented to zero. At this time the transaction is merged into the future events chain, as described in "RETURN Block" in Chapter 10.
2. The preempt flag (bit 7 of byte T10) is reset to zero.
3. The preempt status bit (bit 6 of byte T10) is set to one.
4. Bit 7 of byte T9 is set to one to indicate that the transaction is on the interrupt chain.
5. The transaction is unlinked from the current events chain.

ADVANCE Block Count Statistics

When a nonzero ADVANCE block time is computed, both the current block count (Wj) and total block count (Nj) are incremented by one. If the block time is zero, only the total block count (Nj) is incremented, unless the transaction cannot enter the next sequential block. The current count (Wj) is then also incremented.

Internal Operation of the ADVANCE Block

Whenever a transaction enters an ADVANCE block and a positive delay is computed, the transaction is unlinked from the current events chain. Bit 5 of T9 is set to one to indicate that the transaction is in the future events chain. The transaction is then merged and linked into the future events chain (see Figure 17) according to the value of its block departure time, which has been stored in transaction word T4. The scan of the future events chain starts at the remote end and continues until a transaction is found whose BDT is less than or equal to the computed departure time of the current transaction.

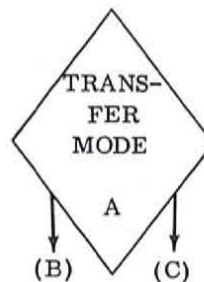
$$BDT_i \leq BDT_k < BDT_{i+1}$$

Whenever the overall GPSS/360 scan can no longer move any more transactions in the current events chain into some next block, the absolute clock time is increased to the block departure time of the first (most imminent) transaction in the future events chain. This transaction and any succeeding transactions with the same block departure time are then linked back into the current events chain.

Each transaction, as it is returned to the current events chain, successively becomes the last transaction in the priority class to which it belongs.

One practical result of the merging criterion ($BDT_i \leq BDT_k < BDT_{i+1}$) is that when two or more transactions have the same block departure time, the first one merged into the future events chain is the first one transferred back into the current event chain. Assuming equal priority, it also becomes the first one processed by the overall GPSS/360 scan after being returned to the current events chain.

TRANSFER BLOCK FROM WHICH GENERALIZED TRANSFERS TO OTHER BLOCKS CAN BE MADE



The TRANSFER block provides the principal means of diverting transactions to a nonsequential next block in a GPSS/360 block diagram.

The TRANSFER Block provides the following nine selection modes by which transactions can move to a nonsequential block:

1. Unconditional (blank)
2. Fractional (.)
3. BOTH
4. ALL
5. PICK
6. Function (FN)
7. Parameter (P)
8. Subroutine (SBR)
9. Simultaneous (SIM)

The mnemonic code specified in field A of the TRANSFER block identifies the selection mode. Fields B and C give various possible next block values. The TRANSFER block in the GPSS/360 assembly program can specify symbolic block locations in fields B and C. When field B of a TRANSFER block is left blank, the assembly program will assign the next sequential value of the block counter to field B of the converted TRANSFER block.

Unconditional (blank) Selection Mode

The entering transaction will proceed unconditionally to the field B next block, and no other block will be tried.

For Example:

TRANSFER , NEXT
TRANSFER , V10

Fractional Selection Mode

The selection mode is interpreted as a three-digit number which indicates the proportion (in parts per thousand) of the entering transactions that is to be diverted to the field C next block. Whichever exit is chosen for a particular transaction, B or C, it will be the only one tried by that transaction. Indirect addressing can be used in field A. *n indicates that the three-digit number is contained in Parameter n of the transaction. The proportion could, therefore, be ASSIGNED to Parameter n before entering a TRANSFER block.

In GPSS/360 the proportion may also be assigned to any Standard Numerical Attribute. Because the selection mode is interpreted as a three-digit number, a computed value of zero or less will result in an unconditional TRANSFER to the field B block and a value of 1,000 or more will result in an unconditional TRANSFER to the field C block.

Example 1: GPSS/360 Card Formats

1	2	LOC	7	8	OPERATION	19	A	B	C
		10			TRANSFER	.709		11	25
		301			TRANSFER	. *1		179	19
		410			TRANSFER	.XH1		25	40

Of the transactions which enter TRANSFER block 10, .709 of these will, on the average, attempt to enter Block 25. Another .291 of the Transactions will attempt to enter Block 11.

The three-digit number which is stored in Parameter 1 of transactions entering TRANSFER Block 301 will represent the probability, in parts per thousand, that the transaction will attempt to enter Block 19. Otherwise the transaction will attempt to enter Block 179. If Halfword Savevalue 1 contained 30 when the transactions enter Block 410, 3% of these will, on the average, attempt to enter Block 40 and 97% will attempt to enter Block 25.

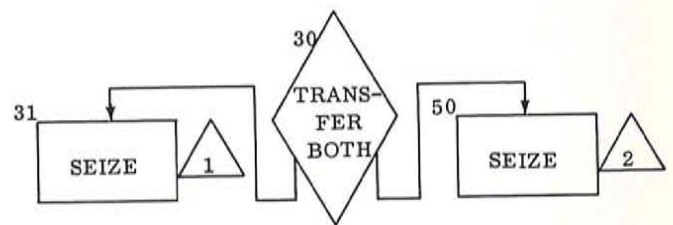
Example 2: Assembly Program Card Formats

1	2	LOC	7	8	OPERATION	19	A,B,C
					TRANSFER	.709,	NEXT 1
					TRANSFER	. *1,	NEXT 2, NEXT 3
					TRANSFER	.368,	NEXT 4, NEXT 5

The first two TRANSFER blocks represent possible equivalents of the GPSS/360 TRANSFER blocks described in Example 1 above. Observe that no absolute block numbers are used. Since 29.1% of the transactions go to the next sequential block after the first TRANSFER block, field B is left blank (indicated by two successive commas, ",, "). The GPSS/360 assembly program will assign the next sequential block number to field B of the TRANSFER block. Blocks 25, 179, and 19 are now assigned the symbolic block locations NEXT1, NEXT2, and NEXT3, respectively as shown in Example 1 above. The third TRANSFER block represents a case where the field B next block is not necessarily the next sequentially numbered block following the TRANSFER block.

BOTH Selection Mode

Each entering transaction will first attempt to exit via the field B next block. If conditions are not met for its entry into this block, the transaction will try the field C next block. If it cannot advance to that block either, it will remain in the TRANSFER block, and will repeat the sequence of trials at each overall GPSS/360 scan until it finds an exit.



In the above example, the transaction will first attempt to enter Block 31. If it is unable to enter that block, it will attempt to enter Block 50. If it fails to enter again, it remains on the current events chain and repeats the same operation at each overall GPSS/360 scan until it finds an exit.

ALL Selection Mode

Each entering transaction will first attempt to exit via the field B next block. If conditions are not met for its advance, the transaction will attempt to exit via blocks B+D, B+2D, B+3D, . . . C where:

D = the value of the field D indexing factor.

C = the field C next block which must exceed B by an exact multiple of the field D indexing factor.

This is checked during input.

If the transaction fails to find an exit, it remains in the TRANSFER block, and repeats the above sequence of trials at each overall GPSS/360 scan until it finds an exit.

Since the field B and C next blocks will normally be entered in symbolic form the analyst must foresee the numbering of subsequent blocks by the GPSS/360 assembly program and insure that the field C next block number will exceed the field B next block by an exact multiple of the field D indexing constant.

Examples:

The first ALL selection mode is legal with the transaction attempting to enter blocks 60, 70, 80, . . . 120 in succession:

The second ALL selection mode is legal only if the number of blocks between block NEXT1 and block NEXT2 is an exact multiple of 5.

```
TRANSFER ALL, 60, 120, 10
TRANSFER ALL, NEXT1, NEXT2, 5
```

The following ALL selection mode is illegal and the card will be rejected during input, because the difference between the field B and C next block is not an even multiple of the field D indexing factor:

```
TRANSFER ALL, 60, 120, 25
```

Only the BOTH and ALL selection modes are conditional. In the other seven modes, an unconditional next block choice is made when the transaction first enters the TRANSFER block. With BOTH and ALL, the next block choice depends on which blocking condition is first removed. Observe that whenever the overall GPSS/360 scan encounters a transaction delayed in a TRANSFER block with a BOTH or ALL selection mode, the GPSS/360 scan will always start with field B next block. Consequently, the BOTH selection mode favors the field B next block in those cases where both the field B and C next blocks can be entered. Similarly, the ALL selection mode favors the lower-numbered next blocks when two or more next blocks can be entered

at the same time. No SNA's or indirect addressing are allowed in fields B, C, and/or D of a TRANSFER block with the ALL selection mode.

PICK Selection Mode

A single next block in the interval from B, B+1, B+2, . . . C, will be unconditionally selected on a random basis for each entering transaction. All values, including the endpoints of the interval, are equally likely with probability $1/(C-B)+1$. No other block will be tried by the transaction. If it fails to exit immediately via the selected exit, it will wait in the TRANSFER block until the blocking condition is removed, and then advance. The field C next block must be greater than or equal to B+1.

```
TRANSFER PICK, 30, 39
```

Transactions entering the above TRANSFER block will attempt to move to one of the 10 blocks (30, 31, . . . 39) with an equal probability of 1/10.

Function Selection Mode (FN)

The value of field B will be computed for that function, and will be truncated if the result is nonintegral. This integer is then added to the field C argument to obtain the next block for the entering: Transaction (field C may be zero if desired). No other block will be tried by the transaction and it will remain in the TRANSFER block until conditions are met for its exit via the selected block.

For example:

```
TRANSFER FN, 3, *3
```

Next Block = Value of Function 3 + Value of Transaction Parameter 3.

Observe that the following TRANSFER block with a blank selection mode does not allow the field C argument to be added to the value of FN3:

```
TRANSFER      A      B      C
                FN3
```

Parameter Selection Mode (P)

The value of the field B argument will be interpreted as parameter number n of the entering transaction. The value of this parameter will be added to the field C argument to obtain the next block selection (field C may be zero if desired). No other block will be tried by the transaction, it will remain in the

TRANSFER block until conditions are met for its advance via the selected block.

For example:

TRANSFER P, 12, FN7

Next Block= Value of
Parameter 12 + Value of Function 7
of Transaction
in Block

Again, observe that a TRANSFER block with a blank selection mode does not allow the value of the field C argument to be added to the value of the field B.

Subroutine Selection Mode (SBR)

The entering transaction will proceed to the field B next block and no other block will be tried. The value of the field C argument is interpreted as a parameter number and the number j of the current TRANSFER block is assigned to that parameter. The field B next block would generally be the start of a block subroutine.

TRANSFER SBR, NEXT, 10

The transaction can now easily return to the block following the original TRANSFER SBR block by making the following TRANSFER block the last block of the subroutine:

	A	B	C
TRANSFER	P	SNAj	1

Next Block = P (SNAj) + 1

The value of SNAj is the same parameter number n specified in field C of the TRANSFER SBR block.

Simultaneous Selection Mode (SIM)

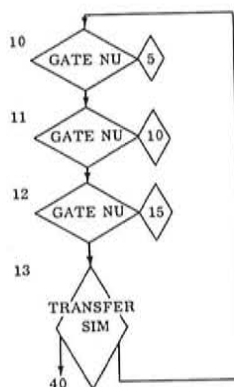
The simultaneous mode of next block selection has been provided so that simultaneous satisfaction of a number of conditions can be represented in a block diagram. There is a SIM delay indicator (bit 1 of byte T10) in each transaction; this indicator records the result of any attempted advance by the transaction. The SIM delay indicator is set to one whenever the transaction finds entry conditions which are not met at its next block selection. If a number of conditions must be

simultaneously satisfied, they may be tested simultaneously (see example below), and if all are satisfactory, the SIM delay indicator of the transaction will remain zero. If any condition causes a delay, the SIM delay indicator will be set to one (where a TRANSFER block selection mode is BOTH and ALL, all attempted exits must fail before the SIM delay indicator is set to one). When the transaction enters a TRANSFER block with a SIM selection mode, the SIM delay indicator is tested. If it is zero, the field B next block is selected for the entering transaction. If the SIM delay indicator is one, the field C next block is selected for the entering transaction, and the SIM delay indicator is reset to zero. In either case, no other block will be tried by the transaction, and it will remain in the TRANSFER block until conditions are met for its advance to the chosen next block. The SIM delay indicator is not retested when the transaction succeeds in leaving the TRANSFER block.

TRANSFER SIM, 40, 10

The status of the SIM delay indicator (bit 1 of T10) is printed out in the DI column of the transaction printout. If the SIM delay indicator has been set, i.e., a delay has occurred, a 1 is printed. A blank is printed if the indicator is reset.

Example:



This example will not allow transactions to proceed to Block 40 unless Facilities 5, 10 and 15 are simultaneously available (not in use). When a transaction enters TRANSFER block 13, its SIM delay indicator is tested. If the transaction was forced to wait because any one of the facilities (5, 10, or 15) was not available the SIM delay indicator will be one. The transaction would then be sent to the field C block 10 to repeat its tests, with the SIM delay indicator reset to zero.

The SIM delay indicator is not set when transactions are delayed in ASSEMBLE, GATHER and MATCH blocks.

Resetting of SIM delay indicator in ADVANCE block

The SIM delay indicator is reset to zero each time the transaction leaves an ADVANCE block in which it spent a nonzero time, i. e., each time the transaction is returned to the current events chain from the future events chain. It is often possible that additional delays may have occurred between an ADVANCE block and a set of simultaneous conditions which the analyst actually desires to test. The SIM delay indicator may, therefore, have been set to one before it ever got into the blocks where simultaneous nondelay conditions are tested. The solution is to precede these blocks with a TRANSFER SIM block whose field B and C next blocks are both the first block in the simultaneous sequence. The TRANSFER SIM block will reset the SIM delay indicator to zero. In the preceding example, the following TRANSFER block could be used:

```
TRANSFER SIM, 10, 10
```

This TRANSFER block assures that the SIM delay indicators of all transactions entering block 10 are reset to zero.

Internal Operation of TRANSFER Block

Whenever a transaction succeeds in entering any next block, the number of this block is stored in the T1 + 2 halfword. When a transaction enters any TRANSFER block, except one in the BOTH or ALL selection mode, the next block to enter is computed, and the transaction will attempt to enter this next block. If a transaction is unable to enter this block, the block number is stored in the first 2 bytes of T5. The next block value will not be recomputed and the GPSS/360 program will continue to try to move the transaction into this next block.

If the selection mode is BOTH or ALL, the field C last block value is stored in the first half of the T1 word. This value will likewise not be recomputed. If the transaction fails to enter any one of the possible next blocks in the BOTH or ALL block range, the GPSS/360 program will continue to try to move the transaction into one of these blocks at each scan of the current events chain.

Byte T10 of all current event transactions indicates the type of next block trial to be made. The following values of byte T10 are printed out in the SF column of the transaction printout, next to the NBA column:

blank -- Indicates that only one next block is to be tried. The transaction could be any block type.

B -- Indicates that a BOTH selection mode is being used. The transaction is in the TRANSFER block whose number is printed in the BLOCK column. The first next block to be tried is printed in the NBA column. The second next block is not printed.

A -- Indicates that an ALL selection mode is being used. Again, the transaction is in the TRANSFER block whose number is printed in the BLOCK column. The lowest number next block to be tried is printed in the NBA column. The field C last block number is not printed.

GENERATE BLOCK TO CREATE TRANSACTIONS

1	2	LOC	OPERATION	IP	A	B	C	D	E	F	G
			GENERATE	Mean Time k, SNA)	Spread k, SNA)	Initial- ization Interval k Modifier FN)	Initial- ization Interval k SNA)	Creation Limit k SNA)	Priority Level k SNA)	Parameter Assignment k, SNA or blank	Parameter Type F, H or blank
GENERATE			A, B, C, D, E, F								

The GENERATE block creates transactions. GENERATE blocks are used to represent the sources of traffic in a system, such as input terminals in a communication network. The creation rate is specified in terms of an intercreation time between successive transactions. The block to which the transactions move is the next sequential block following the GENERATE block. Observe that indirect addressing (*n, SNA*n) is illegal in all fields.

Sequence of GENERATE Block Operations

1. When the GPSS/360 input program encounters a GENERATE block one transaction is immediately created. The set of four basic words for the new transaction is obtained from the internal chain of inactive transactions. The priority class of the new transaction is specified in field E of the GENERATE block. A blank field E implies a 0 Priority Level. The Priority; (0, 1, . . . 127) is then stored in byte T7. The transaction is scheduled to leave the GENERATE block after a time interval that is computed from the field A mean time and the field B modifier in the same manner as an ADVANCE block delay time. However, because the transaction

is created at input time, any function or variable used as a mean or modifier must have been previously defined. Likewise any savevalue must have been previously initialized (see "INITIAL Card", Chapter 8). The value of a function, used as a modifier, will not be truncated; truncation takes place only after multiplication by the mean. At the end of the computed time interval the transaction will leave the GENERATE block. The parameter assignment for the new transaction is specified in field F of the GENERATE block. Since GPSS/360 allows the use of full or halfword parameter values, it is necessary for the user to make such indication in field G. If field G contains an 'F', fullword parameters are assigned. If field G is blank or contains an 'H', halfword parameters are assigned. If field F is blank, field G is not examined and twelve halfword parameters will be assigned. For example:

For ten fullword parameters:
GENERATE 5, FN3, , , , 10, F

For 50 halfword parameters:
GENERATE 10, V3, , , , 50, H

For twelve halfword parameters:
GENERATE X10, FN2

2. Consider the following GENERATE block:

2	LOC	7	8	OPERATION	19
			GENERATE		100

The first transaction from the GENERATE block would be scheduled to leave the GENERATE block at clock time 100. Prior to that time, the "incipient" transaction is located in the future events chain. When its scheduled entry time arrives the "incipient" transaction is first moved from the future events chain to the current events chain. The GPSS/360 scan will eventually cause the transaction to enter the system via the GENERATE block. The new transaction becomes the last transaction in the priority class specified in field E. The current value of the absolute clock is stored in the mark time word (T6) as the creation time.

3. The transaction next attempts to move into the next sequential block following the GENERATE block. If the transaction succeeds in moving into this block, a new transaction is created and a set of words is obtained from the internal chain of inactive transactions. The time interval until the arrival of the new transaction is computed again from the field A mean time and the field B modifier. This

intercreation time interval is added to the current absolute clock time to obtain the time at which the transaction is to enter the system. The new "incipient" transaction is then merged into the future events chain scheduled to enter the GENERATE block. In the above example, this step would first be performed at clock time 100. Since the action time of the GENERATE block is the constant 100, the entry time of the new transaction would be 200. At that time the new transaction would also receive the same processing as described in steps 2-4. The flow of transactions from each GENERATE block is thus propagated by each preceding member of the stream.

4. Assume now that the transaction entering the system via the GENERATE block has been moved from the future events chain to the current events chain but fails to enter the next sequential block, e.g., a GATE, TEST, SEIZE, PREEMPT, or ENTER block. In this case the successor "incipient" transaction will not be created. The successor transaction will be created only when the first transaction finally succeeds in entering the next sequential block. The intercreation time will be computed then and added to the absolute clock time to determine the entry time of the successor transaction. This will, of course, lead to interarrival times greater than those specified by fields A and B.

5. If the first intercreation interval is computed to be zero (that is, when the GENERATE block definition card is first encountered), it will always be taken to be one. Observe that a zero intercreation time is permissible (both fields A and B are blank). However, a block type which will eventually cause a blocking condition must follow the GENERATE block (GATE, TEST, SEIZE, PREEMPT, ENTER). Otherwise, the GENERATE block will continue to create transactions at an infinite rate until all available transactions in the GPSS/360 program are used. Execution Error 468 will then occur.

6. The total block count (Nj) is incremented by one when each transaction leaves a GENERATE block. Since the current count is incremented only when a transaction fails to enter the next block, the current count (Wj) will never exceed one.

7. Indirect addressing is not possible in the GENERATE block because all the parameters of GENERATE block transactions have zero values.

Initialization Interval and Creation Limit

Field C of the GENERATE block definition card can define an initialization interval, which is taken as the time at which the first transaction is scheduled

to leave the GENERATE block. This interval may be less than, equal to, or greater than field A mean time.

Field D of the GENERATE block can define a creation limit, which is the maximum number of transactions that will be created by the GENERATE block. When the specified number of transactions has been created, an "incipient" successor transaction is no longer created and merged into the future events chain. If field D is blank the GENERATE block will continue to create transactions indefinitely.

Redefinition of a GENERATE Block

Whenever a GENERATE block is redefined by a new GENERATE block definition card, the GPSS/360 program will examine all the transactions which are currently in the simulation model to find the unique transaction associated with the GENERATE block (there may be no transaction if a field D creation limit has been decremented to zero so that the GENERATE block has been deactivated). The old transaction, if one exists, is immediately destroyed. In either case, a new transaction is then created just as if the GENERATE block was being defined for the first time.

Error Conditions

A GENERATE block may never be entered by a transaction moving in the simulation model. Execution Error 413 will result if this is attempted. Just as with ADVANCE blocks, the field B spread time of a GENERATE block must not exceed the field A mean time; otherwise, Execution Error 530 will result. If the mean or modifier is specified by a variable, the variable statement must have been previously defined or Input Error 260 will occur. If a function modifier has not been defined by the time the GENERATE definition card is encountered, Input Error 264 results (undefined function). In general, FUNCTION and VARIABLE definition cards should be placed first in the input deck to avoid this GENERATE block error.

Internal Operation of GENERATE Block

The field A mean, the field B modifier, the field C offset, the field D creation limit, the field E priority level and the field F number of parameters are stored, in this order, in the six words (24 bytes) obtained from the GPSS/360 COMMON area. The field C initialization interval is recognized only at input time when the creation time interval of the first transaction is computed. The field E priority is also stored in byte T7 of the first transaction.

The field SNA denoting priority is evaluated each time a transaction is created at the GENERATE block.

If a creation limit is specified in field D its value is also placed in the mark time word (T6) of the first transaction. When this transaction leaves the GENERATE block, the T6 value is decremented by one. If it has been decreased to zero, a new "incipient" successor is not created, and the GENERATE block is thereby deactivated. If word T6 is still nonzero, a successor transaction is created and merged into the future events chain.

This decrementing operation on mark time word (T6) is performed on all GENERATE block transactions. Consequently, if the GENERATE block does not specify a field D creation limit, a sequence of negative values (-1, -2, -3, . . .) will appear in the mark time word (T6) and will be printed in the transaction printout. At any time, -n in the MARK column indicates that n transactions have been created at the particular GENERATE block at which the Transaction is located.

Recreating GENERATE Block Transactions after a CLEAR Card

The CLEAR card removes all transactions from a GPSS/360 model, including the transactions which are associated with GENERATE blocks (see "Clear Card" in Chapter 15). After completing all the various CLEARing operations, the GPSS/360 program recreates a transaction at each GENERATE block in the model, just as if the GENERATE block definition card had just been read. If a field C offset was specified in the original GENERATE block definition card, the SNA is evaluated to obtain the arrival time of the first transaction. Otherwise, the arrival time of the first transaction at the GENERATE block is computed from the field A mean time and field B modifier values. If there was originally a field D creation limit, the SNA is reevaluated and stored in the mark time word (T6). If a priority level was specified in field E, the SNA is evaluated and the result is moved to byte T7 of transaction words. Each new GENERATE block transaction is then merged into the future events chain on the basis of its computed arrival time.

The GPSS/360 program searches the blocks in ascending order (e.g., 1, 2, . . .) so that the recreated transactions are added to the current events chain in the order of increasing GENERATE block number.

Reactivation of GENERATE Blocks

If a creation limit has been specified in field D, and has been decremented to zero (i.e., there is

no longer a successor transaction at the GENERATE block), the GENERATE block will be reactivated in the simulation only if one of the following two events occur:


1. A CLEAR card is read in and a new transaction is created as described above.
2. The GENERATE block is redefined with a new GENERATE block definition card, as described earlier in this chapter.

Redefinition or Changing to a GENERATE Block

"CHANGE Block" in Chapter 6 warns that a currently defined block, which is not a GENERATE block and which contains one or more transactions, cannot be redefined or CHANGED to a GENERATE block. Otherwise Error 698 occurs.

TERMINATE BLOCK TO DESTROY TRANSACTIONS

1	2	LOC	7	8	OPERATION	19	A
					TERMINATE	k, *n, SNAJ, SNA*n Termination Count	



The TERMINATE block removes individual transactions from the block diagram. It is used to represent the completion of a path of flow in a system, such as the arrival of a message at a receiving terminal in a communication network. Field A of the TERMINATE block specifies the number of units which this block contributes to the total TERMINATE count. If field A is blank, the block does not contribute to the count. Since the transactions are removed immediately upon entering a TERMINATE block, no further functions are performed by the block. The transaction is unlinked from the current events chain and the set of four basic transaction words are returned to the internal chain of inactive transactions. The additional transaction words in COMMON are freed for other use. The contents of all words, including parameter words, are set to zero.

A TERMINATE block will never refuse entry to a transaction. Each time a transaction enters the block the count of the total number of transactions that have entered the TERMINATE block (N_j) is incremented by one. The number of transactions currently at the TERMINATE block is always zero, i.e., $W_j = 0$.

Control of Simulation Run Length

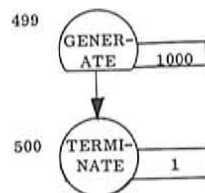
When the user prepares a GPSS/360 simulation run he specifies the length of the run in a START card in terms of a number of terminations to be performed.

Since all paths of the block diagram do not have equal significance, the TERMINATE blocks are permitted to specify whether or not they contribute to the run termination count. The value of the TERMINATE block field A argument will be counted toward the run termination count. Field A of the START card, in turn, will specify the run termination count ("START Card" in Chapter 15). If desired, several TERMINATE blocks may contribute to the run.

Example of Simulation Run Timer

1	2	LOC	7	8	OPERATION	19	A
*					RUNTIMER		
	499				GENERATE	1000	
	500				TERMINATE	1	
					START	5	
					START	20	

RUN FOR 5000 CLOCK UNITS
NOW RUN FOR 20000 CLOCK UNITS



The pair of blocks in this example may be used to control the simulation run. Each transaction which enters the TERMINATE block will decrement the current run termination count by one. Assume that all other TERMINATE blocks in the block diagram have blank fields A. The program will run until clock time 5000 is reached, since the first START card specifies that the termination count for the first run is 5. Since the GENERATE block creates one transaction every 1000 time units, the fifth transaction will enter the TERMINATE block at clock time 5×1000 or 5000. The second START card specifies 20 terminations, so the second run will last 20,000 clock units. The GPSS/360 program begins executing the simulation run when it encounters a START card. It automatically prints the statistics produced by the run after the termination count has been reduced to zero or less. The GPSS/360 program then reads input cards until another START card is encountered.

BLOCK TYPES THAT MANIPULATE USER CHAINS

An additional type of transaction chain is available which, in conjunction with LINK and UNLINK blocks, enables the analyst to remove transactions from the current events chain, put them in a temporarily inactive state on the transaction chain (user chain) and then, at some later time, place them back on the current events chain. This enables the analyst to structure his own chains and completely bypass the predetermined operation of the current events chain. These user chains can be utilized to model any

queuing system and also to reduce computer running time.

In the standard GPSS/360 program, there are 20 to 100 user chains. Standard Numerical Attributes associated with user chains are as follows:

CAj = Average number of transactions on User Chain j
 = $\frac{\text{Cumulative Time Integral (U7 Doubleword)}}{\text{Relative clock time since last RESET or CLEAR card}}$

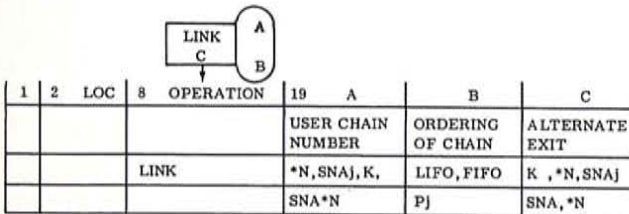
CHj = Current number of transactions on User Chain j (U3 Halfword)

CMj = Maximum number of transactions on User Chain j (U4 Halfword)

CCj = Total number of entries on User Chain j (U5 Fullword)

LINK BLOCK

The LINK block is used to remove a transaction from the current events chain and place it on a user chain. The symbol for the LINK block is:



Field A specifies the user chain to which the entering transaction will be "linked", and may be any SNA. For example, the constant 1 would indicate User Chain 1, X19 would indicate the user chain specified by the contents of Savevalue 19, *1 or P1 would indicate the user chain specified by the contents of Parameter 1 associated with the transaction entering the LINK block, etc.

NOTE 1: If the Standard Numerical Attribute CHj were used for the A argument, the number of transactions on the User Chain j would be used to determine the user chain to which the entering transaction would be linked.

NOTE 2: If the value of any specified SNA is zero, an error results.

Field B specifies the ordering to be followed on the user chain.

FIFO The transaction is placed on the end of the user chain.

LIFO The transaction is placed at the beginning of the user chain.

P(j) The transaction is merged into the user chain according to the value of P(j). The sequence is ascending when ordering by a parameter value. That is, the transaction with the smallest value of P(j) would be at the beginning of the user chain, and the transaction with the largest value of P(j) is at the end. If an entering transaction has a P(j) value equal to the P(j) value of a transaction(s) already on the user chain, the entering transaction will be merged in back of all transactions on the user chain whose P(j) value equals the P(j) of the entering transaction.

TABLE 12: CORE ALLOCATION FOR USER CHAINS

U1	First Transaction On User Chain	U2	Last Transaction On User Chain
(2 bytes)		(2 bytes)	
U3	Number of Transactions On User Chain	U4	see below
(2 bytes)		(2 bytes)	Bit 0 = Link Indicator
			Bits 1 - 15 = Maximum number on User Chain
U5	Total Entry Count		
(4 bytes)			
U6	Clock Time of Last Status Change		
(4 bytes)			
U7	Cumulative Time Integral		
(8 bytes)			

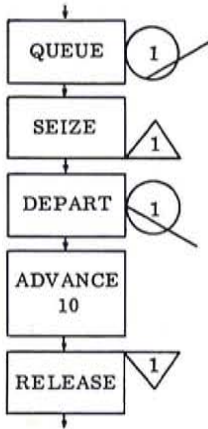
TABLE 13: EFFECT OF RESET AND CLEAR CARDS ON USER CHAIN ATTRIBUTES

Word	Length	RESET Card Attribute Value Before RESET Card	Result of RESET Card On Attribute Value
U1	2 Bytes	First Transaction On User Chain	Unchanged
U2	2 Bytes	Last Transaction On User Chain	Unchanged
U3	2 Bytes	Number of Transactions On User Chain	Unchanged
U4	2 Bytes	Maximum Number of Transactions On User Chain	Set equal to current number on User Chain
U5	4 Bytes	Total Number of Transactions On User Chain	Set equal to current number on User Chain
U6	4 Bytes	Clock Time of Last Status Change	Set to current value of the absolute clock
U7	8 Bytes	Cumulative Time Interval	Set to zero

CLEAR Card

All 24 bytes (U1-U7) are set to zero.

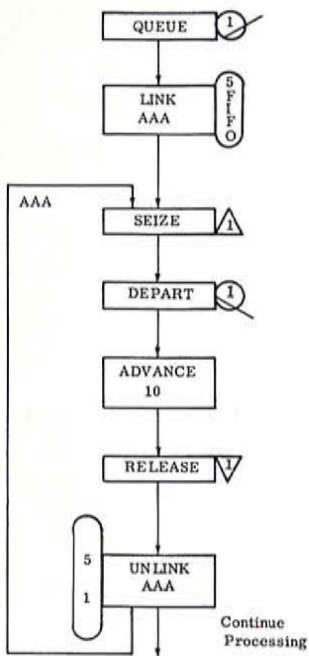
The field C alternate exit is used when representing various queuing situations. The use of field C and the link indicator associated with it can be understood more easily when explained in conjunction with the type of problem it will be used to solve.



In the above block sequence, if a transaction attempts to seize Facility 1 and finds the facility busy, the transaction will be removed from the current events chain and placed on a delay chain associated with the facility. When the facility is finally released, all those transactions which were

placed on the delay chain waiting for the facility to be released are removed from this delay chain and put back on the current events chain. The transactions then attempt to seize the facility, but only the first transaction is able to; the following transactions are then placed back on the delay chain when they attempt to seize the facility. This procedure can be quite time-consuming when the number of transactions waiting to seize the facility becomes relatively large. Instead of the standard delay chain which normally contains those transactions waiting to seize the facility, in the following example the transaction is placed on a user chain if it cannot seize the facility and only one transaction is removed from this user chain when the facility is released.

The incoming transaction is to be placed on the user chain if the facility is in use, so there is a link indicator associated with each user chain to display this fact. The indicator is originally in an off position. It is to be in an on position whenever the facility is in use or when a transaction is already waiting to seize the facility, and off when the facility is free so that it can be determined whether the transaction should try to seize the facility immediately or be placed on the user chain.



The above sequence of blocks accomplishes the purpose of allowing only one transaction to attempt to seize the facility at any one time. The first transaction to enter the LINK block finds the link indicator off and is not placed on the user chain but takes the field C alternate exit, AAA, and turns the link indicator on. The transaction then seizes Facility 1, leaves the queue, and proceeds to the ADVANCE block.

Assume that before this transaction releases the facility, another transaction enters the LINK block; this latter transaction finds the link indicator on and therefore is placed on User Chain 5. When the former transaction enters the UNLINK block, it finds that there is a transaction on User Chain 5; this transaction is removed from Chain 5 and sent to block AAA to seize the facility. The link indicator remains on, and the transaction which enters the UNLINK block continues to the next sequential block. If a transaction enters the UNLINK block and finds no transaction on the chain, it will turn the link indicator off so that any following transaction will be able to seize Facility 1 immediately.

The sequence of operations associated with the LINK block are as follows:

1. The "A" argument is evaluated to determine the user chain.
2. If the "C" argument is blank the link indicator associated with the specified user chain will be set to on and the entering transaction will be linked

unconditionally to the user chain on the basis of the ordering specified by the "B" argument.

3. If the "C" argument is not blank, the sequence is as follows:

The user chain link indicator is tested. If the link indicator is on, the entering transaction will be linked on the user chain on the basis of the ordering specified by the "B" argument. If the link indicator is off, it will be set to on and the entering transaction will proceed to the block specified by the "C" argument.

UNLINK BLOCK

The UNLINK block is used to remove transactions from a user chain. The symbol for the UNLINK block is:

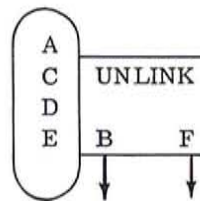


Figure 18 illustrates the format of the UNLINK block.

Block Arguments:

Field A User Chain number from which transaction will be UNLINKed.

Field B Number of the next block for the UNLINKed transaction(s).

Field C Transaction UNLINK count.
ALL-- all transactions will be UNLINKed
or

the value of the argument specified will be the number of transactions UNLINKed.

Field D Can contain one of the following:

1. Parameter number
 - a. If field E is blank, the value of the specified parameter of the entering transaction will be matched with the value of the associated parameter of the transactions on the user chain.
 - b. If field E is not blank, the value of the specified parameter of the transactions on the user chain will be matched with the value of the field E argument.

In both cases, those transactions which satisfy the matching condition will be UNLINKed and sent to the field B block.

2. BACK

The transactions will be removed from the user chain starting at the back of the chain according to the count specified in Field C. Field E must be blank.

3. Boolean Variable

The transactions will be removed from the user chain according to the count specified in field C if and only if the value of BVj is 1. If BVj=0, the entering transaction will attempt to enter the block specified in field F. If field F is blank, the transaction will attempt to enter the next sequential block. Whenever a Boolean variable is specified in field D, field E must be blank.

Field E The argument whose value will be matched with the value of the parameter specified in field D of the transaction on the user chain.

Field F Block number of the next block for the entering transaction when the transaction count of the specified user chain is zero, the match option is used and the condition is not met, or the value of a specified Boolean variable is zero.

The sequence of operations associated with the UNLINK block is as follows:

Case 1: A, B, C arguments have entries; D, E, F arguments are blank.

1. The "A" argument is evaluated to determine the user chain.

2. The user chain is tested to determine whether there are any transactions on it. If the user chain is empty (no transactions), the associated user chain link indicator is set to off and the entering transaction proceeds to the next sequential block.

If the user chain is not empty, the "C" argument is evaluated to determine the number of transactions which should be removed from the user chain.

3. Transactions are removed from the user chain starting at the beginning of the chain and continuing until the count has been decremented to zero or until no transactions remain on the chain. The transactions which are removed from the user chain will be placed on the current events chain scheduled to enter the block specified by the "B" argument of the UNLINK block.

4. The transaction which entered the UNLINK block will proceed to the next sequential block.

Case 2: A, B, C, F arguments have entries; D, E arguments are blank.

This is similar to Case 1 except when the specified user chain is empty.

In this situation the link indicator associated with the user chain is turned off and the transaction which entered the UNLINK block proceeds to the block specified by the "F" argument of the UNLINK block and not to the next sequential block.

Case 3: A, B, C, D arguments have entries; E and F arguments are blank.

This is similar to Case 1. However, only transactions on the user chain whose value of Parameter (j) (which is specified in field D) equals the value of Parameter (j) of the transaction which entered the UNLINK block will be removed from the User Chain. Transactions are removed from the chain until the count specified in field C has been reduced to zero or until all the transactions on the chain have been examined. All unlinked transactions are sent to the block specified by the "B" argument of the UNLINK block. The transaction which entered the UNLINK block proceeds to the next sequential block.

A, B, C, D arguments have entries; E and F arguments are blank; D argument specifies Back.

This is similar to Case 1, except that transactions will be removed from the back (or end) of the specified user chain.

A, B, C, D arguments have entries; E and F arguments are blank; D argument specifies BVj.

1	2	Loc	8	Operation	19	A	B	C	D	E	F
						User Chain Number	NBA	Transaction Unlink Count	Parameter Number	Match Argument	NBB
				UNLINK		K, *N, SNAj, SNA*N	K, *N SNAj, SNA*N	All or K, *N, SNAj, SNA*N	K, *N SNAj, SNA*N BACK	K, *N, SNAj, SNA*N	K, *N SNAj, SNA*N

Figure 18. General Format of the UNLINK Block

This is similar to Case 3 except that BVj is computed for each transaction on the user chain and only transactions for which BVj=1 are removed from the user chain.

Case 4: A, B, C, D, F have entries; E argument is blank.

This is similar to Case 3 except when the specified user chain is empty (CHj=0) or there is no matching parameter value or BVj=0 for all transactions on the user chain. In this situation the transaction which entered the UNLINK block proceeds to the block specified by the "F" argument of the UNLINK block and not the next sequential block. The link indicator is turned off only if the chain is empty.

NOTE: Field E must be blank when the "D" argument of the UNLINK block specifies BACK, or BVj, or else an appropriate error message will be printed out.

Case 5: A, B, C, D, E arguments have entries; F argument is blank.

1. The user chain is determined by the "A" argument.
2. The number of transactions to be removed is determined by the "C" argument.
3. The block to which the unlinked transactions will be sent is determined by the "B" argument.
4. The transaction(s) on the user chain whose Pj (specified in field D) value equals the SNA (specified in field E) is removed from the user chain.
5. Transactions are examined and removed (if possible) starting at the beginning of the user chain and continuing until the count has been decremented to zero or all transactions on the user chain have been examined.
6. The transaction which entered the UNLINK block continues to the next sequential block.

Case 6: A, B, C, D, E, F arguments have entries.

This is similar to Case 5 except when the specified user chain is empty (i. e., CH(j)=0) or no match is found. The transaction which entered the UNLINK block proceeds to the block specified by the "F" argument of the UNLINK block and not to the next sequential block. The link indicator is set to off only if the chain is empty.

LINK/UNLINK EXAMPLES

The block diagrams of the models shown in Figures 19, 20, and 21 illustrate how various situations may be represented making use of the

LINK, UNLINK blocks and the user chain entity of the GPSS/360 program.

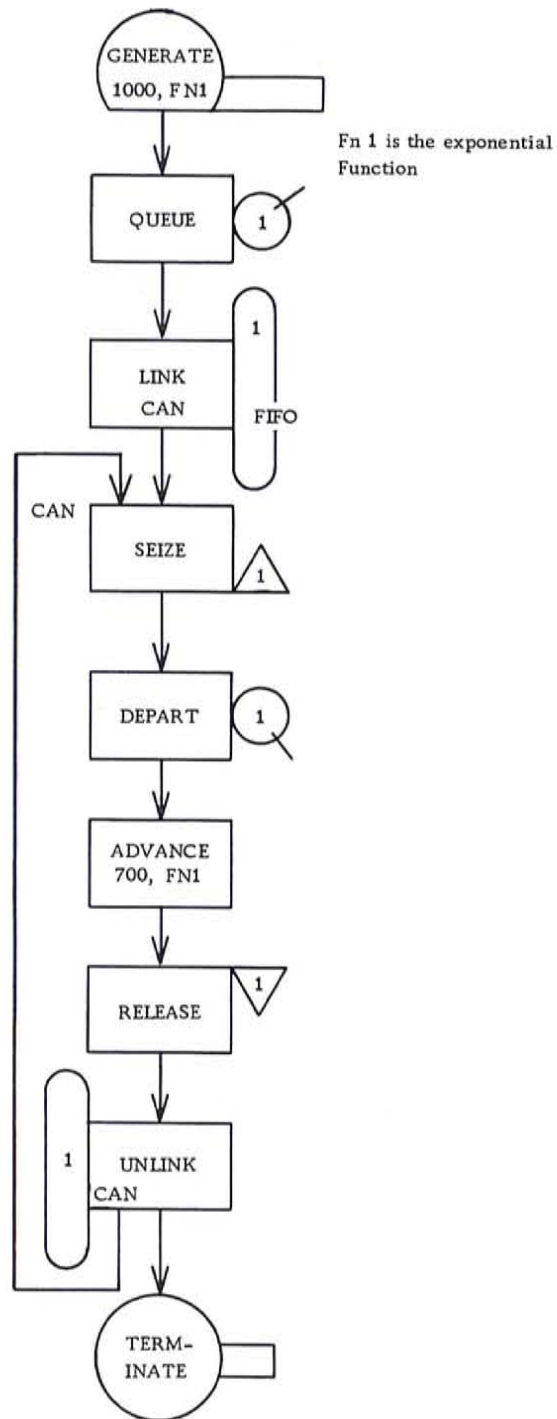


Figure 19. A FIFO Queue Situation
Example 1

Figure 19 illustrates a FIFO queue situation. This is the same model used to indicate the use of the link indicator and is explained in more detail

here. The reason for using the LINK/UNLINK combination here is to save computer time, as explained previously. Transactions leaving the GENERATE block enter the QUEUE block. After updating the associated queue statistics, the transaction enters the LINK block. Since this LINK block has an alternate exit, the associated user chain "link indicator" is tested. If the indicator is off, the transaction immediately sets the user chain link indicator to "on" and proceeds to the alternate block, which in this model is the SEIZE block. If the user chain indicator is on, the transaction entering the LINK block is unconditionally linked to the user chain specified by the A argument of the LINK block. Since the B argument specifies FIFO, the transaction is placed at the end of the user chain.

Note that transactions which are linked to user chains do not proceed for further processing until they are subsequently removed from the user chain by a transaction entering an UNLINK block. When removed from the user chain the unlinked transaction(s) proceeds to the block specified by the B argument of the UNLINK block.

When the transaction enters the SEIZE block it updates statistics associated with the specified facility. It then proceeds to the DEPART block, where the queue statistics are updated. The transaction then proceeds to the ADVANCE block, where it remains the amount of time specified by the mean and modifier of the ADVANCE block.

Upon leaving the ADVANCE block, the transaction enters the RELEASE block. The specified facility is released and the associated facility statistics are updated. The transaction then proceeds to the UNLINK block.

When the transaction enters the UNLINK block, the user chain specified by the A argument is examined. If the specified user chain is empty, i.e., $CH_j=0$, the associated user chain link indicator is set to "off" and the transaction proceeds to the next sequential block. In this model this is a TERMINATE block where the transaction would be removed from the system.

If the user chain was not empty, i.e., $CH_j \neq 0$, the first transaction on the user chain would be removed and placed on the current events chain scheduled to enter the block specified by the B argument of the UNLINK block. In this model this would be the SEIZE block whose symbolic name is CAN. The entering transaction would then proceed to the next sequential block. Although this is a relatively straightforward model it illustrates several important points:

1. The only transactions which are active in the system— that is, on the current, future,

interrupt, or delayed chains—are the transactions coming from the GENERATE block and the one transaction which has currently seized the facility. All other transactions, if any, would be on User Chain 1.

2. Since all delayed transactions—that is, transactions queued up for facility 1—would be on User Chain 1, the simulation program will not waste computer time resetting and setting delay indicators for these transactions every time the status of the facility changes. The amount of time saved is dependent on the length of the queue. The longer the queue the more time will be saved by using the LINK/UNLINK combination to control the queue for various entities.

3. The user has the ability to dynamically form his own chains, and is no longer restricted to the future, current, interrupt, and delayed chains associated with the internal operation of the GPSS/360 program.

Example 2

The purpose of this example is to show that any queuing situation or any queue service technique can be represented by LINK/UNLINK. Figure 20 illustrates a situation where the transaction to be serviced is randomly selected from a user chain. The operation of this model is similar to the model illustrated in Figure 19, up to and including the point where the transaction leaves the RELEASE block.

When the transaction leaves the RELEASE block it enters the ASSIGN block. At the ASSIGN block the value of V1 is computed and assigned to transaction Parameter 1, where $V1 = RN1@CH3$.

<u>Values of CH3</u>	<u>Possible values of V1</u>
0	0
1	0
2	0 through 1
3	0 through 2
4	0 through 3
-	-
-	-
N	0 through (N-1)

After leaving the ASSIGN block, the transaction will enter the TEST block.

P1 = 0

The transaction proceeds to UNLINK block B. When the transaction enters UNLINK block B, it examines User Chain 3. If there are no transactions on User Chain 3, it sets the associated user chain link

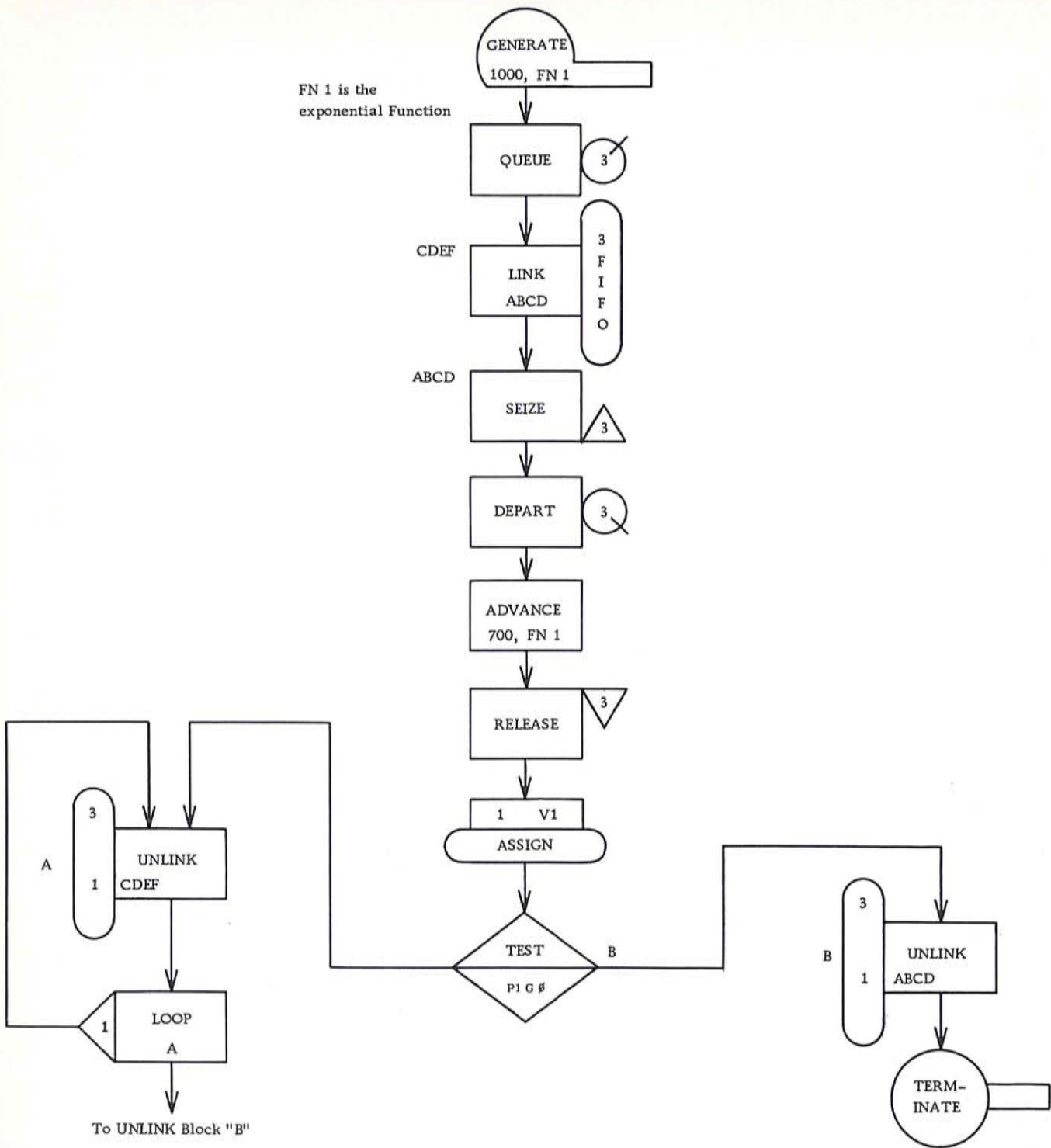


Figure 20. Random Selection of Transaction from a User Chain

indicator to "off" and proceeds to the next sequential block. In this model this is a TERMINATE block where the transaction would be removed from the system.

If the user chain was not empty, that is $CH3 \neq 0$, the first transaction on User Chain 3 would be

removed and placed on the current events chain scheduled to enter the block specified by the B argument of the UNLINK block. In this model this would be the SEIZE block whose symbolic name is ABCD. The entering transaction would then proceed to the next sequential block.

P1 ≠ 0

The transaction would proceed to UNLINK block A. The transaction entering the UNLINK block A would cause the first transaction on User Chain 3 to be removed and placed on the current events chain scheduled to enter the block specified by the B argument of the UNLINK block. In this model this would be LINK block CDEF. At LINK block CDEF the unlinked transaction(s) would again be linked to User Chain 3.

The transaction would leave the UNLINK block and enter the LOOP block. In the LOOP block, P1 would be decremented by one and tested. If P1 were now zero, the transaction would go to UNLINK block B, whose operation would be as previously described. If P1 ≠ 0, the transaction would go to UNLINK block A. This action would "cycle" the transactions on User Chain 3 until the randomly selected transaction calculated by V1 was at the beginning of the user chain. At this point it would be removed and serviced by UNLINK block B.

Example 3

Figure 21 illustrates how the analyst can bypass the predetermined operation of the current events chain and completely structure his own order of occurrences. This model depicts a vehicle rental agency where the transactions representing vehicles have their P1 equal to 1, 2, 3, depending on vehicle type, and P3 equal to their year. As the vehicles are brought back to the agency they are linked onto one of three chains depending on the type of vehicle. There is one control transaction which represents an agent renting the vehicles.

The control transaction enters the SAVEVALUE block where the year of the vehicle the customer desires is placed in SAVEVALUE 10, the type of vehicle desired is then placed in Parameter 6, and then the transaction enters the UNLINK block. An attempt is then made to remove one transaction from the proper chain of vehicles (this chain number is given by P6). A search is made of the proper chain, and the first transaction (vehicle) whose year (P3) matches the year of the vehicle desired (X10) is removed from the chain and sent to its proper service routine as directed by FN2. If a vehicle is found, the control transaction drops through to spend the proper amount of time to fill out rental forms. If a vehicle is not found which matches the specified condition the control transaction goes through a string of blocks to see whether there is an alternate vehicle to satisfy the customer.

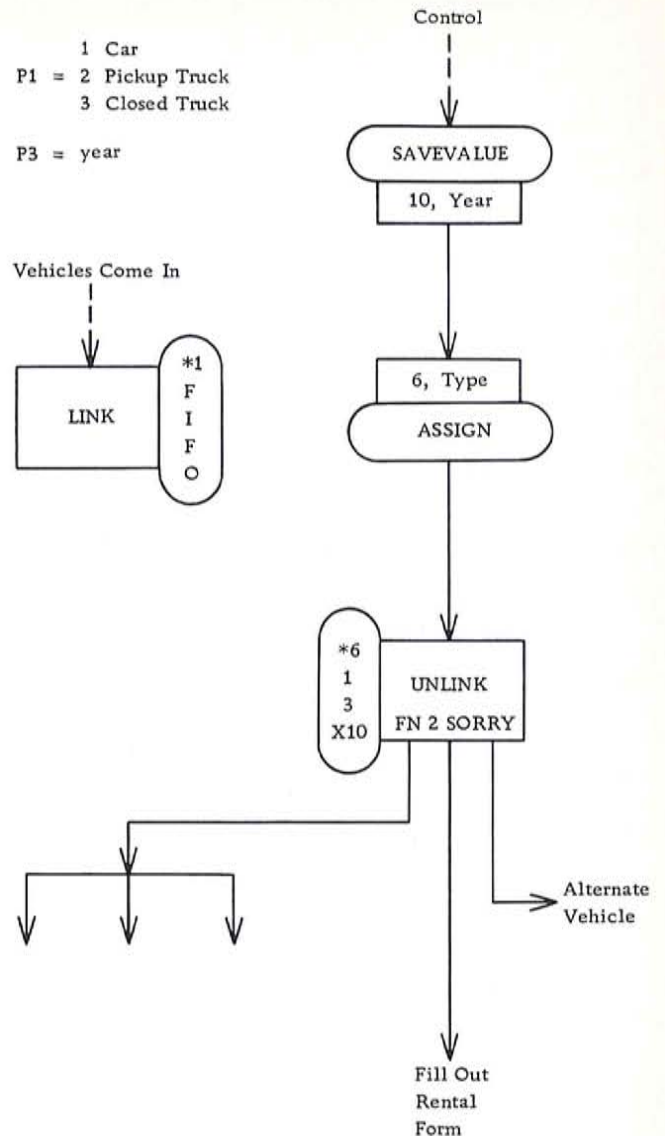


Figure 21. User Chain Structured Based on Parameter Value

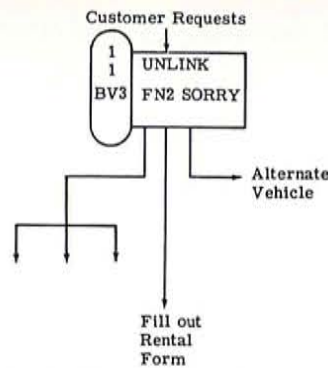
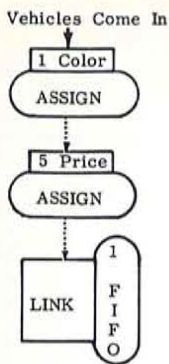
Example 4

Figure 22 shows the use of a Boolean variable in an UNLINK block. In this mode, five different attributes are assigned to describe each vehicle:

- 0 Red
- P1 = 1 Green
- 2 Yellow
- 3 Blue

P2 = Weight

- P3 = 10 Automatic
- 20 Standard



BVARIABLE 3 = P1'E'3*P2'LE'K1000*P3'E'10*(P4'E'1+P4'E'3)*P5'LE'K250

Figure 22. UNLINKing User Chain Transactions that Satisfy Boolean Variable.

- 0 Ford
- P4 = 1 Cadillac
- 2 Plymouth
- 3 Chevy

P5 = Leasing Price

USER CHAIN STATISTICS

In GPSS/360, the following output statistics will be provided for each user chain:

1. Maximum number of transactions on the user chain (U4 Halfword)
2. Average number of transactions on the user chain

$$\frac{U7 \text{ Doubleword}}{C1} = \frac{\text{Cumulative Time Integral}}{\text{Relative Clock Time}}$$

3. Total number of transactions which were placed on the user chain (U5 Fullword)
4. Average time a transaction was on the user chain

$$\frac{U7 \text{ Doubleword}}{U5 \text{ Fullword}} = \frac{\text{Cumulative Time Integral}}{\text{Total entry count}}$$

The interpretation and meaning of these statistics are dependent on the model and the manner in which the user chain entity is being used.

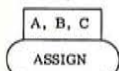
For example, if the user chains were being used to represent an inventory situation, the above statistics would represent:

- maximum number of items in inventory,
- average number of items in inventory,
- total number of items in inventory over a period of time,
- average time an item was in inventory.

BLOCK TYPES THAT MODIFY TRANSACTION ATTRIBUTES

ASSIGN Block

1	2	LOC	7	8	OPERATION	19	A	B	C
					ASSIGN	Parameter Number n	k, *n, SNA}(#)	SNA to be ASSIGNED	No. of Function Modifier
						SNA*n	k, *n, SNA}, SNA*n	k, *n, SNA}, SNA*n	



The ASSIGN block is the principal means of entering numerical values into transaction parameters. The ASSIGN block will never refuse entry to a transaction. Transactions will move to the next sequential block following the ASSIGN block. The value of the field A argument is the number n of the parameter to which a value is to be ASSIGNED. In GPSS/360, each transaction has twelve parameters unless otherwise specified by the analyst. Therefore, the legal field A values are 1, 2, . . . , 12. The parameter values are stored as signed numbers in the words associated with the transaction, beginning at T15. The program enables the user to designate that zero to 100 parameters be associated with each transaction. He also has the ability to specify fullword or halfword parameters, both of which are signed integers. The SNA for all parameters remains P; the interpreting of halfword or fullword parameters is an internal operation. For example, if the user desires to have 20 fullword parameters associated with each transaction, the following GENERATE block could be specified:

GENERATE , , , , 20, F

If fields F and G of the GENERATE block are blank, twelve halfword parameters are provided. With the use of fullword parameters, GPSS/360 parameters have a maximum range of -2^{31} to $+2^{31}-1$. Halfword parameters are restricted to -2^{15} to $+2^{15}-1$.

Replacement, Subtraction and Addition Modes

The character (blank, + or -) which immediately follows the field A parameter number indicates how the computed ASSIGN value is to be used:

- (blank) the value is to replace the current value of the specified parameter
- (+) the value is to be added to the current value of the specified parameter
- (-) the value is to be subtracted from the current value of the specified parameter

For example:

ASSIGN 3, X10 Replace entering value of Parameter 3
 ASSIGN *n+, X10 Add to entering value of Parameter *n
 ASSIGN FN3-, X10 Subtract from entering value of Parameter FN3

ASSIGNed Value

Field B of the ASSIGN block specifies a Standard Numerical Attribute whose value is to be assigned to the field A parameter. The index number (j) of a function modifier may be specified in field C. In this case, the value assigned is the product of the field B Standard Numerical Attribute and the untruncated value of the field C function modifier (FNj). The function modifier values (FNj) can, therefore, be noninteger values, similar to the field B function modifier values in ADVANCE and GENERATE blocks. These are the only three cases in the GPSS/360 program where noninteger values occur.

If any overflow occurs during the computation of the integer to be assigned, only the low-order 15 bits are retained for halfword parameters and the following warning message is also printed:

WARNING EXECUTION ERROR 850. BLOCK NUMBER XXXX. CLOCK YYYY. SIMULATION CONTINUES.

Where: XXXX = ASSIGN block at which the error occurred.

YYYY = Clock time at which the error occurred.

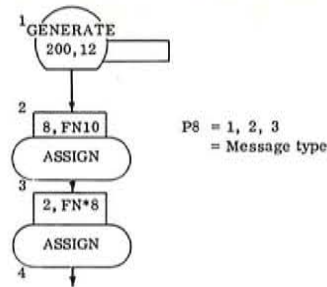
This message is only printed the first time an overflow occurs as a result of an ASSIGN operation and the simulation continues.

Example 1:

The following example illustrates how ASSIGN blocks may be used to introduce system data into a block diagram. Let us suppose that a communication system receives messages of three types, each of which follows a different character length distribution. In each transaction, Parameter 8 will contain the message type and Parameter 2 will contain the message length.

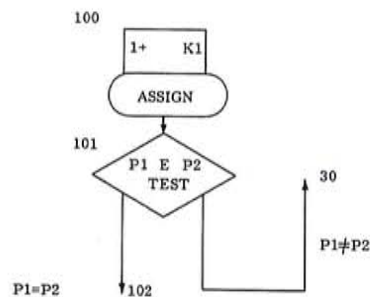
Example 2:

ASSIGN blocks may also be used to control the logic of the block diagram. Let us suppose that the system processes messages character by character. The block diagram will thus represent the processing steps that are required in the examination of one character. Each transaction represents one message and must thus repeat the processing loop a number of times that is equal to



* 1	LOC 2	7	OPERATION 8	13	A 19	B	C						
	1		GENERATE		200	12							
	2		ASSIGN		8	FN10							
	3		ASSIGN		2	FN*8							
*	CHARACTER LENGTH DISTRIBUTIONS						MESSAGE TYPE						
	10		FUNCTION		RN1	D3							
	.075	1	.95	2	1.0	3							
	1		FUNCTION		RN1	C5		TYPE 1	CHARACTER LENGTH				
	0	32	.38	55	.492	68	.705	90	1.0	122			
	2		FUNCTION		RN1	C3		TYPE 2	CHARACTER LENGTH				
	0	20	.63	31	.99	57							
	3		FUNCTION		RN1	D2		TYPE 3	CHARACTER LENGTH				
	.0822	42	1.0	52									

its character length. Transaction Parameter 2 contains the message character length. The following pair of blocks can be used to control the loop for each transaction. The ASSIGN block adds one to the count of the number of processing loops the transaction has executed (Parameter 1). The TEST block admits all transactions. It diverts



them back to the start of the processing loop so long as the relational condition (P1 equal to P2) is false. When P1 finally equals P2, all the characters have been processed and the transaction proceeds to the next sequential block after the TEST block.

Example 3:

The importance of not truncating the value of the ASSIGN block field C modifier can be illustrated by the following example. Assume that there are numerous message types in a simulation model and each of these has a different mean character length. The message type numbers and character lengths are ASSIGNED to Transaction Parameters 4 and 7, respectively. Assume that the character lengths of each message type are exponentially distributed. Assume that the discrete numerical valued (Dn) Function 2, which uses P4 (message type) as an argument, gives the mean character length of each message type as the function value, FN2. Assume, finally, that Function 1 is the same exponential function as that described in the "Exponential Distribution" example in Chapter 5.

The appropriate randomly distributed character lengths can be assigned to each transaction by passing all transactions through the following ASSIGN block:

ASSIGN 7, FN2, 1

Each FN 2 mean character length value would be multiplied by the exponentially distributed untruncated fractional values of Function 1.

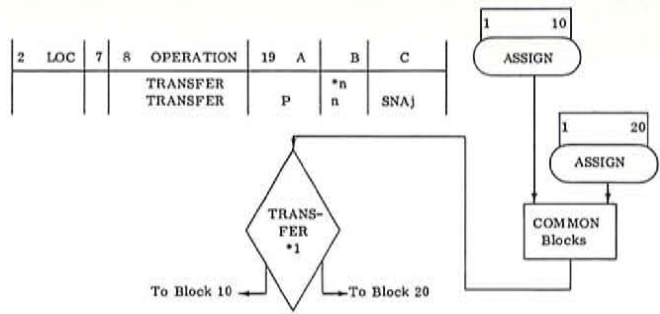
Consider what would happen if the value of the field C function modifier (FN1) was truncated before multiplying the value of FN2, the mean character length. The only possible values of FN1 would be the integer values 0, 1, 2, . . . 8. The most reasonable solution to this problem would be to modify Function 1 by dividing each Y_i function value by 1000. The following ASSIGN block could then be used to assign exponentially distributed character lengths to each transaction.

ASSIGN 7, V10
10 VARIABLE FN2 * FN1/K1000

The combination of FN1/K1000 generates the same exponentially distributed values as the original fractional valued FN1. The simulation model would run more slowly, however, when computing the value of Arithmetic Variable 10.

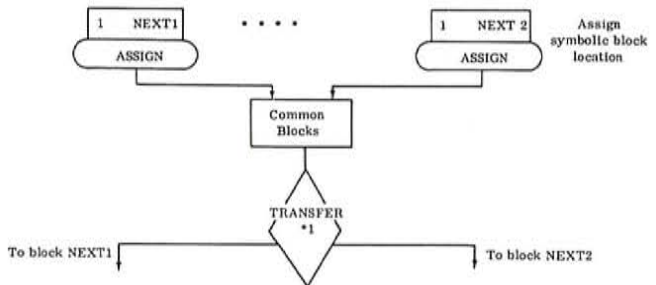
Symbolic Assembly Program Block Locations as Field B Arguments.

A very common use of ASSIGN blocks is to assign different block numbers to some Transaction Parameter n, depending on the type of transaction. These different transaction types can all move through a common group of blocks, which end in one of the following two types of TRANSFER blocks:



Each different transaction type will now TRANSFER to the block numbers originally specified as the field B arguments of the entering ASSIGN blocks.

The symbolic assembly program is based on the use of symbolic block locations. Consequently, symbolic block locations will very often be coded as the field B arguments of ASSIGN blocks, so that the above TRANSFER block operations can be performed.



INDEX Block

2	LOC	7	8	OPERATION	19	A	B
				INDEX	Parameter no. n		
					k, *n, SNAj,	Increment	
					SNA*n	k, *n, SNAj,	
						SNA*n	

The INDEX block is used to change the values of parameters of transactions which enter the INDEX block. The value of the SNA specified in field B is added to the current value of the parameter whose number n is specified by the field A SNA. This sum is then placed in Transaction Parameter 1. The original field A parameter is unchanged, unless, of course, Parameter 1 is specified in field A. Therefore:

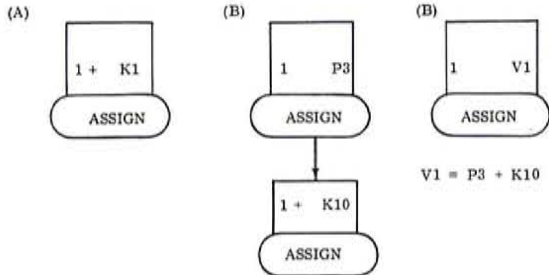
$$P1 = \text{Value of parameter } (SNA_A) + SNA_B$$

The INDEX block will never refuse entry to a transaction. Transactions proceed to the next sequential block following the INDEX block.

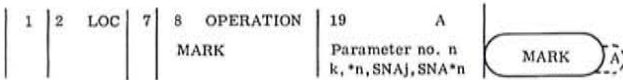
Examples:

- (A) INDEX 1, 1 Increment P1 by 1
- (B) INDEX 3, 10 Replace P1 with P3 + 10

The above INDEX block operations could be performed more slowly by the following block combinations.



MARK Block



Each transaction can have two kinds of transit time Standard Numerical Attributes. These were described earlier in this chapter.

- M1 = Transaction Transit time = Current Absolute Clock Time - Mark Time word (T5) of the Transaction currently being processed
- MPn = Parameter Transit Time = Current Absolute Clock Time - Value of Parameter n of the Transaction currently being processed

The absolute clock time when each transaction enters a simulation model via GENERATE blocks is stored in the Mark Time word (T6) of each transaction.

This original creation time can be updated and replaced by the current absolute clock time by passing the transaction through a MARK block. The MARK block never refuses entry to a transaction. Transactions proceed to the next sequential block. The absolute clock time when the transaction enters the MARK block is placed in the Mark Time word (T6). Field A can contain the number n of a transaction parameter. Transactions which enter such a MARK block will have the low-order 15 bits (modulo 32, 768) of the absolute clock time stored in the specified halfword parameter. If a fullword parameter is being used, the absolute clock time is stored as a 31-bit integer. This permits use of the Parameter

Transit Time Attribute (MPn or MP*n) to measure intermediate transit times through a model. If the value of the field A argument is zero, it will be interpreted as if it were blank; i. e., the absolute clock time will be placed in the Mark Time word (T6).

COUNT Block

2	LOC	7	8	OPERATION	14	X	19A	B	C	D	E	
				COUNT		Conditional operator or logical operator	Parameter in which to place count	Lower limit of entity class to be examined	Upper limit of entity class to be examined	Comparison value if conditional operator specified in column 14	Entity attribute to be counted	A, B, C, E
							k, *n, SNAj, SNA*n	k, *n, SNAj, SNA*n	k, *n, SNAj, SNA*n	k, *n, SNAj, SNA*n	any SNA except MATRIX SAVEVALUES	

The COUNT block enables the user to determine the number of items which meet a specific condition by passing a transaction through a single block -- namely, the COUNT block. For example, the user might be interested in determining the number of facilities within a given range which are not in use, or the number of storages with an average utilization less than 500 (in parts per thousand), etc.

Field A of the COUNT block specifies a parameter number of the entering transaction in which the COUNT will be placed. Field A may be any System Numerical Attribute (SNA) and may be indirectly specified (*n). However, if the user specifies *n, the parameter in which the COUNT will be placed is not parameter n but parameter j, where j is the value of parameter n.

Fields B and C of the COUNT block specify the lower and upper limits, respectively, of the range of the specified entity to be tested.

Field D of the COUNT block is used in conjunction with the conditional operators (E, NE, G, GE, L, LE) specified in column 14. The SNA specified in field D is evaluated and compared against the entity attribute specified by field E. It is not necessary to specify field D if other than conditional operators are specified in column 14 of the COUNT block.

Field E of the COUNT block is also used in conjunction with the field D entry and with the conditional operators (E, NE, G, GE, L, LE) specified in column 14. Field E specifies the entity attribute to be counted and may be any GPSS/360 SNA mnemonic except MATRIX SAVEVALUES. It is necessary only to specify the SNA mnemonic in the E fields, since the range of a

given entity class to be included in the test is specified in fields B and C.

The operator entry beginning in column 14 may be a logical or a conditional operator. The logical operators specify explicit conditions which are to be examined -- count of facilities not in use, count of logic switches reset, etc. The logical operators are:

Facilities

- NU Facility not in use (available)
- U Facility in use (any regular or preempt usage)
- NI Facility not interrupted (no PREEMPT usage, but either available, or in regular usage)
- I Facility interrupted (any PREEMPT usage)

Storages

- SE Storage empty (zero contents)
- SNE Storage not empty (nonzero contents)
- SF Storage full (zero space)
- SNF Storage not full (nonzero space)

Logic Switches

- LR Tests for logic switch reset
- LS Tests for logic switch set

When logical operators are used, fields D and E of the COUNT block should be blank.

As previously mentioned, six conditional operators may also be used for entity attributes which may have a wide range of values. The conditional operators are:

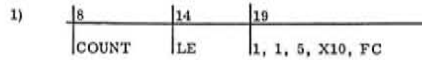
- L Less than. If the value of the SNA specified in field E is less than the value of the SNA specified in field D, the relation is satisfied.
- LE Less than or equal to. If the value of the SNA specified in field E is less than or equal to the value of the SNA specified in field D, the relation is satisfied.
- E Equal to. If the value of the SNA specified in field E is equal to the value of the SNA specified in field D, the relation is satisfied.
- NE Not equal. If the value of the SNA specified in field E is not equal to the

SNA specified in field D, the relation is satisfied.

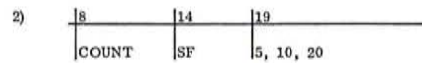
- G Greater than. If the value of the SNA specified in field E is greater than the value of the SNA specified in field D, the relation is satisfied.
- GE Greater than or equal to. If the value of the SNA specified in field E is greater than or equal to the value of the SNA specified in field D, the relation is satisfied.

When conditional operators are used, the COUNT block must have entries in fields D and E.

The operation of the COUNT block can best be explained by examples.

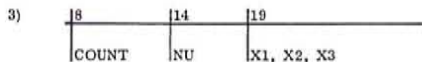


This example would count the number of facilities between 1 and 5, inclusive, which have an entry count (FC) less than or equal to the current value in fullword savevalue 10. The result of the count (0-5) will be placed in parameter 1 of the entering transaction.



This example would count the number of storages between 10 and 20, inclusive, which are full (SF). The resultant count would be placed in parameter 5 of the entering transaction.

NOTE: No D or E fields are required.



This example would count the number of facilities between i (the value of X2) and j (the value of X3), inclusive, which are not in use. The resultant count would be placed in parameter n (value of X1) of the entering transaction.

SELECT Block

2	LOC	7	8	OPERATION	14	X	19	A	B	C	D	E	F
				SELECT									
					Conditional operator or logical operator or (MAX, MIN)			Parameter in which to place entity # that meets condition	Lower limit of entity class to be examined	Upper limit of entity class to be examined	Com-parison value if conditional operator specified in column 14	Entity attribute to be examined	Alternate exit, if no entity meets specified condition
					k, *n SNAJ, SNA*n			k, *n, SNAJ, SNA*n	k, *n, SNAJ, SNA*n	k, *n, SNAJ, SNA*n		any SNA except MATRIX SAVE-VALUES	k, *n, SNAJ, SNA*n

The operation of the SELECT block is similar to that of the COUNT block except that, instead of COUNTing the number of entities that meet a specified condition, the SELECT block "selects" the first entity of the specified range which meets the prescribed condition. The number of this entity is then placed in a parameter of the entering transaction which is specified by field A of the SELECT block.

The contents of fields A through E of the SELECT block are identical to those of the COUNT block. In addition, Field F is used to specify an alternate block for the entering transaction if no entity in the given range meets the specified condition. If no field F is specified, the entering transaction always proceeds to the next sequential block.

Column 14 of the SELECT block may contain any of the 16 mnemonics mentioned previously for the COUNT block. In addition, two others are provided for the SELECT block only: MAX and MIN. If MAX or MIN is specified in column 14 of the SELECT block, no Field D constant is necessary.

The operation of the SELECT block is illustrated in the following examples.

1)

8	14	19
SELECT	MAX	1, 5, 10, , FR

This example selects the facility with maximum utilization (FR) between facilities 5 and 10 inclusive. The facility number is placed in parameter 1 of the entering transaction. NOTE: The field D is blank when MAX or MIN is used for the conditional operator beginning in column 14.

2)

8	14	19
SELECT	GE	10, 10, 20, X5, Q

This example selects the first queue, between queues 10 and 20 inclusive, with a current contents (Q) greater than or equal to the value in fullword savevalue 5 (X5). The queue number is placed in parameter 10 of the entering transaction.

BLOCKS THAT MODIFY THE SEQUENTIAL BLOCK FLOW OF TRANSACTIONS

The GPSS/360 program generally attempts to move transactions only to the next sequentially numbered block. The TRANSFER block, discussed earlier in this chapter, permits a wide variety of transfers to nonsequential next blocks. Three other block types can also conditionally transfer transactions to non-sequential blocks:

1. LOOP block
2. TEST block
3. GATE block

LOOP Block

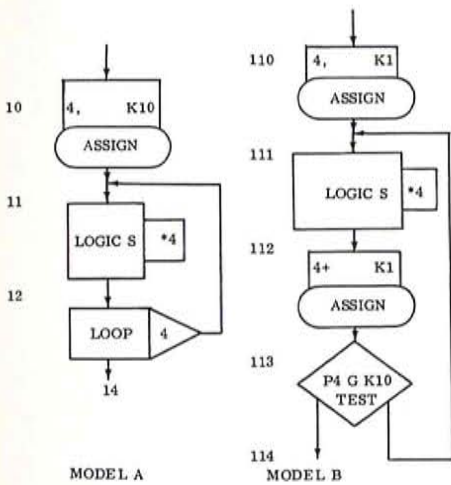
2	LOC	7	8	OPERATION	19	A	B
				LOOP	Parameter no. n k, *n, SNAJ, SNA*n		Next Block if Pn ≠ 0 k, *n, SNAJ, SNA*n

The LOOP block serves to control the number of times that a transaction will pass through a section of the block diagram. Any one of the transaction parameters of a transaction may be used to count the number of loops which have been executed. A transaction is never refused entry to a LOOP block. The GPSS/360 program obtains the contents of the transaction parameter which is specified in field A of the LOOP block. This integer is reduced by one, and the result is placed back in the parameter.

A next block is then selected for the transaction in the following manner. If the parameter that was decremented is not yet zero, the transaction will be sent to the block specified in field B of the LOOP block. If the parameter has been reduced to zero, the transaction will be sent to the next sequential block following the LOOP block. The field B next block will customarily specify the first block number in the block diagram loop. Any number of transactions may simultaneously execute the same loop. If a transaction enters the LOOP block with a parameter value of n, and the parameter is not altered elsewhere in the loop, it enters the LOOP block n times and leaves, nonsequentially, n-1 times. If the specified parameter field is zero or negative at the time of entering a LOOP block, running Execution Error 429 occurs (LOOP counter parameter initially zero or negative).

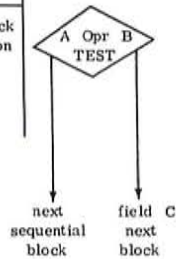
Example:

Logic Switches 1 through 10 are to be put in a set condition. The following figures illustrate two procedures for creating the required block diagram loop. In each case transactions will execute the loop ten times. The LOOP block in Model A will be entered ten times by each transaction. The first nine entries will move back to Block 11. The tenth entry will cause Parameter 4 to be reduced to zero, and Block 14 will be chosen as the next block. In order to execute a loop n times using the LOOP block, the number n should be in a parameter field at the start of the loop. Model A runs much faster since the LOOP block compactly performs the work of the ASSIGN and TEST blocks in Model B.



TEST Block

LOC	7	8	OPERATION	13	A	B	C
			TEST	E	First SNA	Second SNA	Next block if relation is false
			TEST	NE	k, *n	k, *n	k, *n,
			TEST	GE	SNAj,	SNAj,	SNAj,
			TEST	G	SNA*n	SNA*n	SNA*n
			TEST	LE			
			TEST	L			



The TEST block, unlike the LOOP block, does not alter any transaction attributes other than the next block to which the transaction proceeds. The TEST block controls the flow of transactions through the medium of an algebraic relation between the integral values of two Standard Numerical Attributes specified in fields A and B of the TEST block. The following six mnemonics specify algebraic

relations which may appear in the auxiliary operation field of the TEST block:

- L - less than -- If the field A Standard Numerical Attribute is less than the field B Attribute, the relation is true.
- LE - less than or equal to -- The relation is true if the field A Standard Numerical Attribute is less than or equal to the field B Attribute.
- E - equal to -- The relation is true only if the two Standard Numerical Attributes are equal.
- NE - not equal -- The relation is true unless the two Standard Numerical Attributes are equal.
- G - greater than -- If the field A Standard Numerical Attribute is greater than the field B Attribute, the relation is true.
- GE - greater than or equal to -- The relation is true unless the field A Standard Numerical Attribute is less than the field B Attribute.

Unconditional and Conditional Entry into TEST Blocks

TEST blocks can operate in two possible ways:

1. Unconditional Entry Mode. If a next block is specified as a field C argument (k, *n, SNAj, SNA*n), transactions will never be refused entry to the TEST block. If the relational statement defined by the auxiliary relational operator and the field A and B arguments is true, the transaction will attempt to move to the next sequential block. If the relational statement is false, the transaction will attempt to move to the next block specified by the value of the field C argument. The next block choice is made only once; this occurs when the transaction first enters the TEST block and is not changed subsequently. When the symbolic assembly program is used, the field C argument of the TEST block can be a symbolic block location.

2. Conditional Entry Mode. If field C of the TEST Block is blank (i.e., there is no alternate next block specified), transactions are denied entry into the TEST block until the relational statement defined by the auxiliary relational operator and the field A and B arguments are true. The TEST block algebraic relation is tested each time the overall GPSS/360 scan processes a transaction which has been blocked by a TEST

block operating in the conditional entry mode. Delayed transactions are usually chained in push-down delay chains; this deactivates their processing by the overall GPSS/360 scan until the blocking condition is removed. This speeding up of the GPSS/360 program does not occur, however, with transactions which are delayed by conditional entry TEST blocks. The indiscriminate use of such TEST blocks can, therefore, materially increase the running time of the GPSS/360 model.

Examples:

1.	8	13	A	B	C
	TEST	L	C1	K500	SNA

Until the simulator clock reaches 500, transactions will proceed from the TEST block to the next sequential block. At or after clock time 500, transactions will proceed to the block specified by the value of the field C argument.

2.	8	13	A	B
	TEST	GE	N12	N50

Whenever the total block count at block number 12 (N12) is greater than or equal to the total block count at block number 50 (N50), transactions will succeed in entering the TEST block and moving to the next sequential block. Whenever the count at block 12 is less than that at block 50, Transactions will not succeed in entering the TEST block.

3.	8	13	A	B	C
	TEST	E	V6	K0	SNA

Whenever the computed value of Arithmetic Variable Statement 6 is equal to zero, transactions will proceed sequentially. Whenever the value is not equal to zero, transactions will proceed to the block specified by the value of the field C argument.

4.	8	13	A	B
	TEST	G	P7	R20

The transaction will succeed in entering the TEST block only if the value of Transaction Parameter 7 is greater than the remaining capacity (R20) in Storage 20. Otherwise, the transaction will not succeed in entering the TEST block.

5. The following sequence of blocks may be used to divert every twentieth transaction to a special path starting at block SPEC. After entering the ADVANCE block, each transaction succeeds in

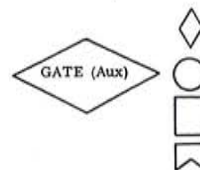
unconditionally entering the TEST block. The program computes the value of Arithmetic Variable Statement 10, which is equal to the remainder from the division of the total block count at block ABC (N\$ABC) by the constant 20. This will be zero whenever the block count is an exact multiple of 20. The first 19 transactions which enter the TEST block will find that the relational statement V10 NE 0 is true, and will proceed to the next sequential block. The twentieth transaction will enter the TEST block and proceed to block SPEC for special processing. The cycle will then repeat, with the fortieth transaction moving to block SPEC, etc.

```

ABC      ADVANCE
          TEST NE      V10, 0, SPEC
          VARIABLE     N$ABC@20
10
  
```

GATE Block

2	LOC	7	8	OPERATION	13	19	A	B
				GATE	Attribute Mnemonic SNA		Entity Index j k, *n, SNAj, SNA	Next block no. if Logical Attribute is false. k, *n, SNAj, SNA*n



The GATE block, similar to the TEST block, does not alter any transaction attributes other than the next block to which the transaction proceeds. The GATE block controls the flow of transactions as a function of the true or false condition of twelve Standard Logical Attributes. The mnemonic code for these attributes is placed in the auxiliary field (columns 13-18) of the GATE block. The entity index (j) is specified by the value of the field A argument.

Facility Logical Attributes

1. NU Field A Facility j is not in use (not SEIZED or PREEMPTed)
2. U Field A Facility j is in use (either SEIZED or PREEMPTed)
3. NI Field A Facility j is not being PREEMPTed
4. I Field A Facility j is currently being PREEMPTed by some transaction

Storage Logical Attributes

5. SE Field A Storage j is empty, i.e., S_j = 0

- 6. SNE Field A Storage j is not empty, i.e.,
Sj > 0
- 7. SF Field A Storage j is full, i.e.,
Rj = 0
- 8. SNF Field A Storage j is not full, i.e.,
Rj > 0

Logic Switch Logical Attributes

- 9. LS Field A Logic Switch j is in a Set condition
- 10. LR Field A Logic Switch j is in a Reset condition

Transaction Logical Attributes

- 11. M A transaction (which belongs to the same assembly set as the transaction currently trying to enter, or within, the GATE block) is in a matching condition at the field A block j.
- 12. NM No transaction (which belongs to the same assembly set as the transaction currently trying to enter, or is within, the GATE block) is in a matching condition at the field A block j.

Unconditional and Conditional Entry into GATE Blocks

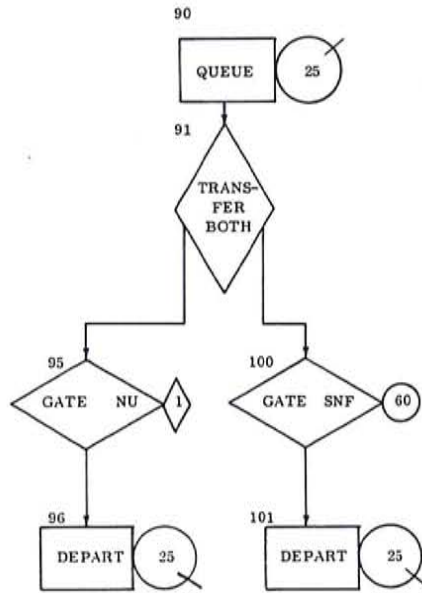
GATE blocks, just like TEST blocks, can operate in two possible ways:

1. Unconditional Entry Mode. If a next block is specified as a field B argument (k, *n, SNAj, SNA*n), transactions will never be refused entry to the GATE block. If the specified logical attribute is true, the transaction will attempt to move to the next sequential block. If the logical attribute is false, the transaction will attempt to move to the next block specified by the value of the field B argument. The next block choice is made only once, when the transaction first enters the GATE block and is not subsequently changed.
2. Conditional Entry Mode. If field B of the GATE block is blank (i.e., there is no alternate next block specified), transactions are denied entry into the GATE block until the specified Standard Logical Attribute is true. The true or false value of the Standard Logical Attribute is not tested (except for the M and NM attributes) each time that the overall GPSS/360 scan processes a transaction which has been blocked by a GATE block which is operating in the conditional entry mode. Delayed transaction are instead chained in pushdown delay chains, which thereby deactivates their processing by the overall GPSS/360 scan until the value of the Standard Logical Attribute becomes true. These

delay chains are described in Chapters 9, 10, and 11, which discuss the logic switch, facility, and storage entities.

Example:

In the following block diagram, transactions will contribute to the contents of Queue 25 until either Facility 1 is available (not in use), or Storage 60 is not full. See Diagram.



Transactions which attempt to enter one or more conditional entry GATE blocks from a TRANSFER block with a BOTH or ALL selection mode will not be placed in a pushdown delay chain. The overall GPSS/360 scan will therefore always attempt to move them into the set of next sequential blocks when the scan encounters them in the current events chain. Therefore the use of such blocks should be made with care to avoid running time inefficiency.

BLOCKS THAT CREATE AND PROCESS MEMBERS OF TRANSACTION ASSEMBLY SETS

GENERATE blocks are the primary means by which transactions are created and enter a simulation model. Transactions can never enter a GENERATE block. The SPLIT block, on the other hand, permits transactions to enter it, and specifies in field A how many offspring of the entering parent transaction are to be created. These copies, along with the parent from which they were created, become members of a unique transaction assembly set.

The TERMINATE block is the primary means by which transactions are destroyed and removed from

a simulation model. The ASSEMBLE block will also destroy transactions of an assembly set and allow only the first of n transactions to proceed to a next sequential block after n-1 others have been destroyed.

The MATCH and GATHER blocks are provided to control the flow of members of transaction assembly sets through the system. Further control is provided by the GATE M and GATE NM blocks.

SPLIT Block

2	LOC	7	8	OPERATION	18	A	B	C	D	
				SPLIT		Nr. of copies k, *n, BNA), BNA*n	Next block for copies k, *n, BNA), BNA*n	Parameter for serial numbering k	Nr. of parameter k, *n, BNA), BNA*n	

The SPLIT block never refuses entry to a transaction. The SPLIT block serves the function of creating offspring of the original, or parent, transaction which enters it. The number of copies to be created is specified by the field A argument. If this argument has a computed value of zero, then the SPLIT block performs no operation and the entering transaction simply proceeds to the next sequential block. All copies are created as soon as the original transaction enters the SPLIT block. The original transaction will attempt to move to the next sequential block after creating the copies. All copy transactions move to the block specified by the value of the field B argument, which is computed separately for each copy transaction.

Field C can optionally designate a transaction parameter (n) so that a serial number may be associated with the new transaction. If, for example, Parameter number k is specified, the following values will be assigned to the kth parameter of the parent transaction and the N copies which are created:

$$X = \text{entering value of Parameter } k$$

<u>Transaction</u>	<u>Value of Parameter k</u>
Original	$P_k = X + 1$
1st Copy	$P_k = X + 2$
2nd Copy	$P_k = X + 3$
.	.
.	.
.	.
Nth Copy	$P_k = X + (N+1)$

Field D of the SPLIT block specifies the number of parameters to be assigned to each copy transaction. If field D is blank, the number of parameters assigned will be the same as that of the original transaction and all values of the original's parameters will be placed in the corresponding parameters of each copy transaction. If a value is specified in field D each parameter (n) of the original which has a corresponding parameter (n) in the copy will be copied. If the copy transaction has more parameters than the original all additional parameters in the copy are set to zero. The length attribute of the parameters of the original transaction is transferred to all copies, i.e., if the parent transaction has halfword parameters, all copies will have halfword parameters; if the parent has fullword parameters, all copies will have fullword parameters.

All the parameter values specified above, the priority level, and the mark time (but not the block departure time) of the original transaction are duplicated in each SPLIT offspring. Each new transaction successively becomes the last transaction in the same priority class in the current events chain. The block departure time of the SPLIT transactions remains zero until they enter an ADVANCE block.

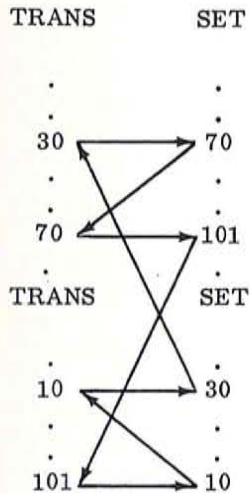
The Total Block Count (Nj) and Current Block Count (Wj) of the SPLIT block are incremented by one for each original and copy transaction. The current count is reduced by one as each transaction succeeds in leaving the SPLIT block.

Each new SPLIT transaction becomes a member of a unique transaction assembly set which evolves from the original transaction that was created in a GENERATE block. The assembly set linkages between transactions of the same assembly set is carried in the second two bytes of transaction word T5, (T5+2). When a transaction is created in a GENERATE block, T5+2 will contain the number of the transaction itself. Hence, each transaction will always belong to an assembly set which consists minimally of itself.

Assume that such a transaction, e.g., number 10, enters a SPLIT block with the Constant 1 specified in field A. The program will obtain the next available set of Transactions words (T1-T4) from the internal chain of inactive transactions. Assume these four words (T1-T4) are associated with Transaction number 30. At this point, the program would chain these two transactions together in a closed one-way cyclic chain by placing the number 10 in T5+2 of Transaction 30 and the number 30 in T5+2 of Transaction 10.

The contents of T5+2 is printed in the SET column of the transaction printout (see Table 10). Consider the following as a portion of such a printout containing Transactions 10-30-70-101, all members of the same assembly set.

CURRENT EVENTS CHAIN



The assembly set linkages cannot reveal directly which transaction was originally created in a GENERATE block. If a copy transaction is itself SPLIT the second copy becomes a member of the same assembly set as the first copy. A transaction is thus a member of one and only one assembly set. Any number of transactions may be contained in a single assembly set. When a transaction is terminated, the program automatically removes it from its assembly set. An assembly set thus continues to exist until the last transaction in the set is terminated. The assembly set linkages are revised after each termination to maintain the closed cyclic chain.

Any number of assembly sets may simultaneously exist in the block diagram without conflict. Each transaction that is created at a GENERATE block is considered an independent assembly set, so the number of sets in the block diagram will be a fluctuating number.

ASSEMBLE Block

2	LOC	7	8	LOCATION	19	A
			ASSEMBLE		Nr. to be assembled k, *n, SNAj, SNA*n	



An ASSEMBLE block joins a specified number of transactions from an assembly set into a single transaction. The ASSEMBLE block will never refuse entry to a transaction. Transactions move to the next sequential block following the ASSEMBLE block. When a transaction enters an ASSEMBLE block, the program searches through that transaction's assembly set linkages to find if another transaction of the same set is at the ASSEMBLE block. There are two possibilities:

1. No other transaction is at the ASSEMBLE block in the process of being assembled (see below).
2. A transaction is already at the ASSEMBLE block being assembled (see below).

Initial Transaction in Assembly Set to Arrive at an ASSEMBLE Block.

The field A assembly count is reduced by one. If the result is zero, i.e., only one transaction is to be ASSEMBLED, the transaction proceeds immediately to the next sequential block. If the result is negative, i.e., the computed count was zero or negative, Execution Error 607 occurs (assembly count zero or negative at ASSEMBLE block). Therefore, the field A value must always be one or more.

Usually, the field A assembly count is greater than one, so that the decrementing operation for the initial transaction leaves a positive balance. The assembly count balance is placed in word T2 of the initial transaction (see Table 9) and the transaction itself is unlinked from the current events chain and placed in an interrupt status (not to be confused with the interrupt chain associated with facilities). It will not be returned to the current events chain until a sufficient number of other transactions have entered the ASSEMBLE block to decrement the assembly count to zero. The Total Block Count (Nj) and Current Block Count (Wj) are both incremented by one. The overall GPSS/360 scan then proceeds to the next sequential transaction in the current events chain.

Internally, the chain indicators in T9 are set to zero to indicate an interrupt status. This zero will be printed in the CI column of the transaction printout as a blank indicating no chain linkages. The Matching Indicator (Bit 5 of byte T10) is set to one to indicate the transaction is in the process of assembly. This indicator is printed in the MC column as a 4.

Succeeding Transactions to Arrive at an ASSEMBLE Block

These transactions will discover that another transaction of their assembly set is currently in the process of being assembled at the ASSEMBLE block. This transaction has been processed as described in the previous section. Its current block number (which is stored in T1) is the ASSEMBLE block number. In addition, its matching indicator has been set to one which indicates it is in the process of being assembled.

The assembly count balance (in T2 of the initial transaction) is now reduced by one. The newly arrived transaction is immediately destroyed. If the assembly count still exceeds zero the overall GPSS/360 scan proceeds to the next transaction on the current events chain.

However, if the count has been reduced to zero, the initial transaction is returned to the current events chain as the last transaction in its priority class. Consequently, it may not be the next transaction to be processed by the overall GPSS/360 scan.

Internally, the chain indicator (T9) is set to indicate the transaction is on the current events chain and the matching indicator is reset to zero.

The completion of the assembly is a GPSS/360 scan status change; i. e. , the status change flag is set. After the initial transaction has been replaced on the current events chain, the overall GPSS/360 scan returns to the start of the current events chain (see "Overall GPSS/360 Scan" later in this chapter). This assures that the overall GPSS/360 scan will process the initial transaction at the same clock time that it completes the assembly.

Even if the initial transaction cannot subsequently move to the next sequential block after it has been restored to the current events chain, it will not be considered as being in the process of assembly. This is because its matching indicator has been set to zero. Consequently, if another transaction in the same assembly set should now arrive at the ASSEMBLE block it would become the initial transaction of the next assembly, even though the previously assembled transaction is still at the block. The Current Block Count (Wj) is reduced by one as each assembled transaction succeeds in leaving the ASSEMBLE block.

Summary of ASSEMBLE Block Operations

The effect of the ASSEMBLE block is to permit one transaction from an assembly set to advance after

N-1 other transactions of that set have arrived at the same ASSEMBLE block. Therefore, only the attributes of the initial transaction are retained. It should be obvious that only one subset of any assembly set may be in the assembly process at one ASSEMBLE block at any moment in time. It is permissible, however, for several subsets of the same set to be simultaneously in the process of assembly at several different ASSEMBLE blocks. It is also permissible for several subsets of the same set to be assembled at the same ASSEMBLE block during successive intervals in time. Lastly, it is possible for several different assembly sets to be simultaneously in the process of assembly at the same ASSEMBLE block. Obviously, the specified number of transactions must enter the ASSEMBLE block. Otherwise, the initial transaction will remain permanently in the ASSEMBLE block in an interrupt status. Also, if the initial transaction is the sole member of its assembly set in the system, Execution Error 609 occurs.

Transactions That are PREEMPTed While in an ASSEMBLE Block

After the initial transaction which is to enter an ASSEMBLE block has been placed in an interrupt "matching" condition, another transaction may enter a PREEMPT block which references a facility that the ASSEMBLE block transaction has SEIZED. The SEIZING transaction in the ASSEMBLE block has either already been PREEMPTed on another facility which it has SEIZED, or this is the first facility on which it is being PREEMPTed.

If the ASSEMBLE block transaction has not been previously PREEMPTed, the following steps occur:

1. The Preempt Status Indicator (Bit 6 of T10) is set to one to indicate that the transaction is being PREEMPTed on a facility.
2. The Preempt Count (T8) is incremented by one.
3. The chain indicator is set to one (Bit 7 of T9) to indicate the transaction is on the interrupt chain for some facility.

Transactions in an ASSEMBLE block, which are both being PREEMPTed and are in a matching condition, will be returned to the current events chain only after both conditions are met.

An ASSEMBLE block transaction, which has already been PREEMPTed on one facility and has been placed in a preempt status, can be PREEMPTed on as many as 126 more facilities which it has

SEIZED. The preempt count is incremented by one each time that a further PREEMPT occurs. Error 474 will occur if this count exceeds 127.

PREEMPTed Transactions That Enter ASSEMBLE Blocks

A transaction may be in the current events chain when another transaction enters a PREEMPT block which references a facility that the first transaction has SEIZED. The SEIZING transaction will not be immediately removed from the current events chain and placed into an interrupt status. Instead, its preempt flag (bit 6 of T10) is set to one. The SEIZING transaction will be placed in an interrupt status only when it enters a nonzero-time ADVANCE block, or when it enters a MATCH, ASSEMBLE, or GATHER block in which it is placed into a matching condition.

When such a PREEMPTed transaction enters an ASSEMBLE block as the initial transaction, the following steps occur:

1. The matching condition bit is set to one according to the procedures described earlier in this chapter for the initial transaction.
2. The preempt flag is reset to zero.
3. The preempt status bit is set to one.
4. The interrupt status bit is set to one to indicate that the transaction is in an interrupt status.
5. The transaction is unlinked from the current events chain.

It should be noted that a transaction may be PREEMPTed on a facility which the transaction itself is PREEMPTing if the PREEMPT block is operating in the priority mode (see "PREEMPT Block," Chapter 10). In such case the same steps

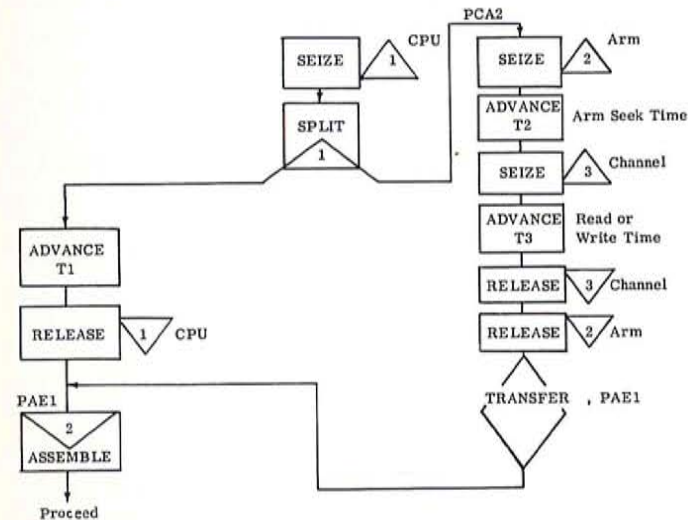
occur as noted above for SEIZING transactions which are preempted.

Examples of SPLIT and ASSEMBLE Blocks

1. Parallel I/O operations. A transaction SEIZES Facility 1, which represents the CPU, and SPLITs off a copy transaction to simulate an I/O operation. The original transaction (left-hand flow) then simulates processing by spending time in an ADVANCE block, RELEASEs the CPU, and goes into an ASSEMBLE block to wait for the completion of the I/O operation. Meanwhile, the copy transaction (right-hand flow) simulates the I/O operation by SEIZING an access arm (Facility 2), spending seek time in an ADVANCE block, SEIZING the channel (Facility 3), spending time in an ADVANCE block to simulate the actual read or write operation, RELEASEing the channel, RELEASEing the access arm, and entering the ASSEMBLE block to allow the original transaction to proceed. The coding and the block diagram for this model is shown below.

Assembly Program coding is used below:

1	2	LOC	7	8	OPERATION	19	A, B, C
					SEIZE		1
					SPLIT		1, PCA2
					ADVANCE		T1
					RELEASE		1
		PAE1			ASSEMBLE		2
					SEIZE		2
					ADVANCE		T2
					SEIZE		3
					ADVANCE		T3
					RELEASE		3
					RELEASE		2
					TRANSFER		, PAE1

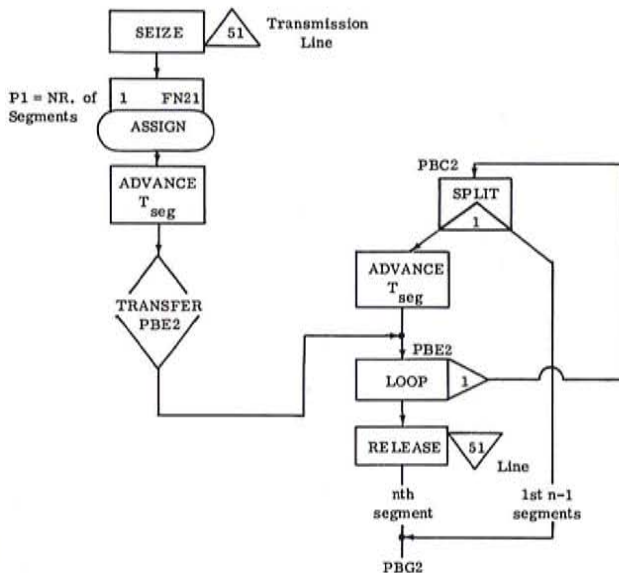


2. Transmission of message segments.

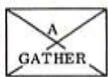
A transaction which is simulating a message SEIZES a line (Facility 51). The number of message segments (n) in this message is assigned to Transaction Parameter 1 by Function 21. The transaction spends time in an ADVANCE block to simulate transmission of one message segment. The transaction then TRANSFERS to the LOOP block. If the message is composed of more than one segment, the transaction LOOPS to the SPLIT block where a copy Transaction is SPLIT off and is sent ahead to simulate processing in the message exchange. The original transaction goes into an ADVANCE block to simulate transmission of the next message segment. After the nth segment has been transmitted Transaction Parameter 1 will be reduced to zero by the LOOP block; this allows the transaction to proceed to the next sequential block. The transaction will then RELEASE the line (Facility 51) and proceed to simulate processing in the message exchange.

Assembly Program coding is used below:

1	2	LOC	7	8	OPERATION	19
					SEIZE	51
					ASSIGN	1, FN21
					ADVANCE	T
					TRANSFER	, PBE2
	PBC2				SPLIT	1, PBG2
					ADVANCE	T
	PBE2				LOOP	1, PBC2
	PBG2				RELEASE	51
					.	



GATHER Block

2	LOC	7	8	OPERATION	19	A
				GATHER	Nr. to be gathered k, *n, SNAj, SNA*n	

The GATHER block is similar to the ASSEMBLE block in that it gathers together a specified number of transactions from an assembly set. However, it does not terminate any of the gathered transactions. Instead, when the specified number have been gathered, all of them proceed to the next sequential block which follows the GATHER block. The GATHER block will never refuse entry to a transaction.

When a transaction enters a GATHER block, the program searches through that transaction's assembly set linkages to find if another transaction of the same set is at the GATHER block and is in the process of being gathered. There are two possibilities:

1. No other transaction is at the GATHER block in the process of being gathered (see below).
2. One or more transactions are already at the GATHER block being gathered (see below).

Initial Transaction to Arrive at GATHER Block

The processing is identical to that described earlier for the ASSEMBLE block. The field A gather count is reduced by one and again stored in word T2. The transaction is removed from the current events chain and placed in an interrupt status.

The Total (Nj) and Current (Wj) block counts are both incremented by one. The chain indicators in T9 are set to zero and the matching indicator is set to one. The overall GPSS/360 scan then proceeds to the next sequential transaction in the current events chain. If the computed gather count is zero or negative, Execution Error 607 occurs. The field A gather count must, therefore, be one or more.

Succeeding Transactions to Arrive at a GATHER Block

These transactions will discover that one or more transactions of their assembly set are currently in the process of being gathered at the GATHER block. The gather count balance stored in T2 of the initial transaction is reduced by one. If the balance is still greater than zero, the newly arrived transaction is removed from the current events chain. It is chained internally to the initial transaction in a one-way FIFO chain (the T1 word of each transaction is used for this purpose). The interrupt status of these transactions is not set to one. The total and current block counts are both incremented by one.

If the gather count balance of the initial transaction is reduced to zero, all of the gathered transactions are returned to the current events chain as the last transactions in their respective priority classes in the following order:

1. The initial transaction is rechainned first (bit 6 of T9 is set, and its matching indicator is reset to zero).
2. The remaining gathered transactions are successively put back into the current events chain, as the last transaction in their respective priority classes, in the same order in which they arrived at the GATHER block. This includes the very last transaction to enter the GATHER block.

The successful completion of the gather operation is a scan status change; i. e. , the scan status change flag is set. After all gathered transactions which are not preempted have been successfully returned to the current events chain, the overall GPSS/360 scan returns to the start of the current events chain. This assures that the scan will process all transactions which have been returned to the current events chain at the same clock time that they were gathered.

Summary of GATHER Block Operations

The effect of the GATHER block is to permit N transactions from an assembly set to advance from a GATHER block after all N transactions of that set have arrived at the same GATHER block. The attributes of all the transactions are the same as when they first arrived at the GATHER block. Only one subset of any assembly set may be in the gathering process at one GATHER block at any moment in time. It is permissible, however, for several subsets of the same set to be simultaneously in the process of being gathered at several different GATHER blocks. It is also permissible for several subsets of the same set to be gathered at the same GATHER block during successive intervals in time. Lastly, it is possible for several different assembly sets to be simultaneously in the process of being gathered at the same GATHER block.

Obviously, the specified number of transactions must enter the GATHER block. Otherwise, the initial, and possibly some succeeding, transaction(s) will remain interrupted permanently in the GATHER block. If, when the gathering operation is complete, the gathered transactions cannot subsequently move to the next sequential block, none will be considered as being in the process of being gathered because their matching indicators will be reset to zero. The current block count is reduced by one as each gathered transaction succeeds in leaving the GATHER block. As was the case with the ASSEMBLE block, if a transaction of a one-membered assembly set arrives at a GATHER block Execution Error 609 occurs.

Transactions That are PREEMPTed While in a GATHER Block

After transactions have been placed in an interrupt matching condition in a GATHER block, another transaction may enter a PREEMPT block which references a facility that the GATHER block transaction has SEIZED. The operations that are performed on such transactions are exactly the same as described earlier for PREEMPTed transactions in ASSEMBLE blocks.

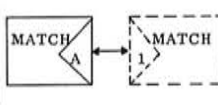
PREEMPTed Transactions That Enter GATHER Blocks

A transaction may be in the current events chain when another transaction enters a PREEMPT block

which references a facility that the first transaction has SEIZED. The operations that are performed on such transactions when they enter GATHER blocks are exactly the same as described earlier for PREEMPTed transactions which enter ASSEMBLE blocks.

MATCH Block

2	LOC	7	8	OPERATION	19	A
1				MATCH	Conjugate MATCH block number k, *n, SNAj, SNA*n	



The MATCH block serves to synchronize the progress of two transactions of an assembly set without removing them from the system. Instead of joining the pair of transactions, the MATCH block permits them to continue to advance through the block diagram. The synchronization is accomplished by matching pairs of transactions from an assembly set and allowing both members of the pair to proceed after the match has been achieved.

The MATCH block will never refuse entry to a transaction. Successfully matched transactions proceed to the next sequential block. Field A of the MATCH block specifies the number of another MATCH block, termed "the conjugate MATCH block". A MATCH block may, if desired, take itself as the conjugate block. It then operates like a GATHER block with a gather count of two. The program obtains the field A block number, and then searches the entire assembly set of which the transaction is a member. If another member of the set is waiting in a matching condition at the conjugate MATCH block, "Matching Transaction at Conjugate MATCH Block" is executed; otherwise, "No Matching Transaction at Conjugate MATCH Block" is executed.

No Matching Transaction at Conjugate MATCH Block

It has been determined that the transaction which is entering the MATCH block does not have an assembly set partner at the conjugate MATCH block. The transaction is removed from the current events chain and placed in an interrupt status on the matching chain. It will not be returned to the current events chain until a member of its assembly set has entered another MATCH block which references its current MATCH block as a conjugate

block. The Total Block Count (Nj) and the Current Block Count (Wj) are both incremented by one. The matching indicator is set to one, as in the ASSEMBLE and GATHER blocks. The overall GPSS/360 scan then proceeds to the next sequential transaction in the current events chain.

Matching Transaction at Conjugate MATCH Block

This is executed when an assembly set partner is found in a matching condition at the conjugate MATCH block. The matching indicator of the partner is reset to zero. The first member of the pair, which was placed in an interrupt status, is returned to the current events chain as the last transaction in its priority class (internally, bit 6 of T9 is set to indicate that the transaction is back in the current events chain, while the matching indicator is reset to zero). Meanwhile, the overall GPSS/360 scan continues processing the second member of the pair as if the MATCH block were a zero-time ADVANCE block.

The successful matching is a scan status change; i.e., the status change flag is set. After the first transaction has been put back in the current events chain, the second transaction attempts to move through as many zero-time blocks as possible. As soon as it encounters a blocking condition, or it spends a positive time in an ADVANCE block, the overall GPSS/360 scan will find the status change flag set and return to the start of the current events chain. This assures that the overall scan will process the first transaction at the same clock time that it was matched.

Even if a successfully matched transaction cannot subsequently move to the next sequential block, it will not be considered as available for matching if another transaction in the assembly set enters another MATCH block which addresses the first MATCH block as a conjugate block. This is because the matching indicator of successfully matched transactions has been reset to zero. The Current Block Count (Wj) is reduced by one as each matched transaction succeeds in leaving the MATCH block.

Summary of MATCH Block Operations

The effect of the MATCH block is to permit the progress of pairs of transactions from an assembly set to be synchronized. Pairs of transactions from several different assembly sets may be synchronized simultaneously at the same pair of MATCH blocks. It is also possible for several pairs of transactions

from the same assembly set to be synchronized simultaneously at different pairs of MATCH blocks. Lastly, a MATCH block may be its own conjugate, if desired; i. e. , it operates like a GATHER block with a gather count of two. Obviously, an assembly set partner must eventually enter a MATCH block which references the initial transaction MATCH block as a conjugate MATCH block. Otherwise, the initial transaction will remain permanently in the MATCH block in an interrupt status.

Transactions That Are PREEMPTed While in a MATCH Block

After a transaction has been placed in an interrupt matching condition in a MATCH block, another transaction may enter a PREEMPT block which references a facility that the MATCH block transaction has SEIZED. The operations that are performed on such transactions are exactly the same as described earlier for PREEMPTed transactions in ASSEMBLE blocks.

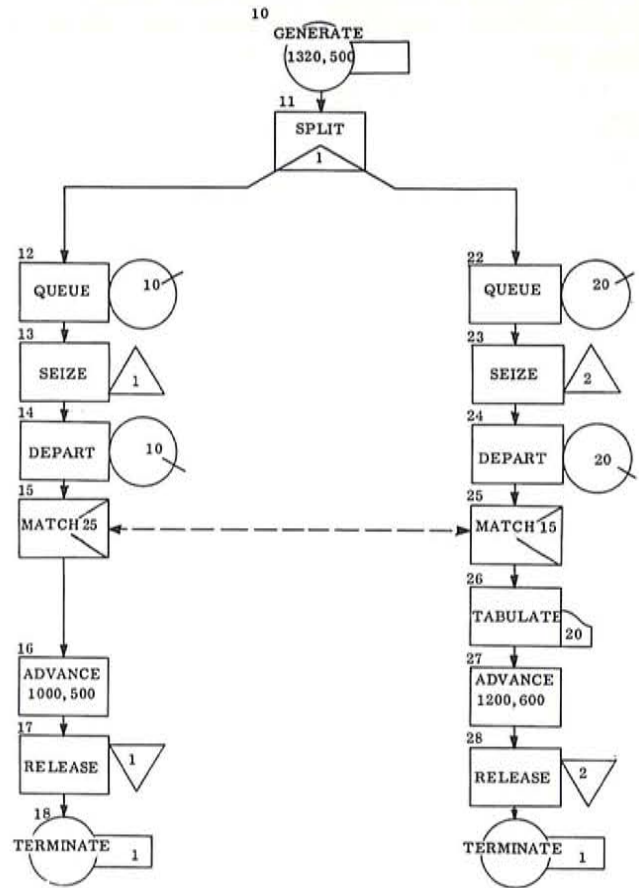
PREEMPTed Transactions That Enter MATCH Blocks

A transaction may be in the current events chain when another transaction enters a PREEMPT block which references a facility that the first transaction has SEIZED. The operations that are performed on such transactions when they enter MATCH blocks are similar to those described earlier for PREEMPTed transactions which enter ASSEMBLE blocks.

Example 1:

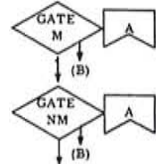
MATCH blocks generally operate in pairs. Each block specifies the other as its conjugate MATCH block. Thus, duplicates of a transaction which are in separate paths in the block diagram may be synchronized by placing a MATCH block in each path. The following example illustrates this procedure.

In the block diagram which follows, transactions are SPLIT and enter queues for Facilities 1 and 2. After a transaction obtains control of one of the facilities, it waits until its duplicate obtains control of the other facility. Both transactions then proceed independently. The TABULATE block in the example measures the interval of time required for each pair of transactions to obtain control of the two facilities.



GATE M and GATE NM Blocks

2	LOC	7	8	OPERATION	13	19	A	B
				GATE	M	Block no. j		Next block no.
				GATE	NM	k, *n, SNAj,		If GATE
						SNA*n		condition is
								false
								k, *n, SNAj,
								SNA*n



Two Standard Logical Attributes are associated with transactions. Their true or false values can be used as the arguments of GATE blocks to control the flow of the transactions within an assembly set.

1. M - is true if another member of the assembly set of the transaction that is now currently being processed at or within a GATE block is in a matching condition at the field A block.

2. NM - is true if no other members of the assembly set of the transaction currently being processed at or within a GATE block are in a matching condition at the field-A block.

A transaction is in a matching condition whenever the matching indicator has been set to

one (which is the MC column value in the transaction printout). A matching condition occurs for members of an assembly set in three ways; at ASSEMBLE, GATHER, and MATCH blocks:

1. Another transaction of the assembly set is the initial transaction still being assembled in an ASSEMBLE block, whose number j is the value of the GATE block field A argument.

2. One or more transactions of the assembly set are still being gathered at a GATHER block, whose block number j is the value of the GATE block field A argument.

3. Another transaction of the assembly set which is awaiting a reference by a conjugate MATCHing transaction is in a MATCH block whose block number j is the value of the GATE block field A argument.

Observe that since the matching indicator must be set to one, a matching condition will not exist for successfully assembled, gathered, or matched transactions which have been unable to enter the next sequential block following their ASSEMBLE, GATHER, or MATCH blocks. The matching indicators of these transactions have been reset to zero.

Unconditional and Conditional Entry into GATE Blocks

As described earlier in this chapter, the presence of a next block argument in field B of a GATE block indicates that transactions will always enter the GATE block. If the Standard Logical Attribute (M or NM) is true, the transactions proceed to the next sequential block; if it is false, they proceed to the block specified by the value of the field B argument. If no field B next block is specified, the GATE block will conditionally deny entry to transactions until the specified Standard Logical Attribute is true. The transactions then proceed to the next sequential block.

Transactions that are blocked from conditional entry GATE M or GATE NM blocks are not deactivated from the overall GPSS/360 scan by being placed in delay chains. The overall GPSS/360 scan attempts to move these transactions into the GATE M or GATE NM block every time that the scan encounters them in the current events chain. This may lengthen the execution time of a GPSS/360 simulation model.

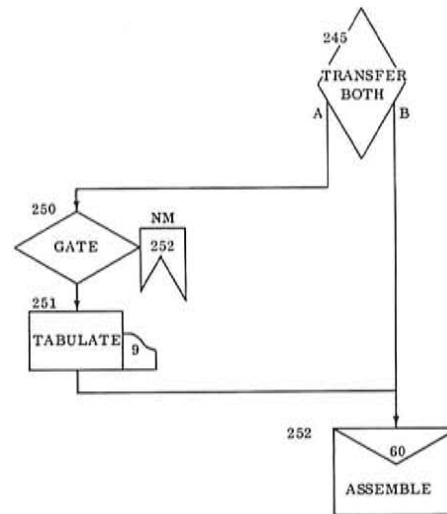
Example:

A GATE block, which tests for one of the matching

conditions, may also be used to test whether a transaction is the first member of its assembly set to arrive at an ASSEMBLE block. In the block diagram below, a transaction will enter block 250 if no other member of its assembly set is being assembled in block 252.

TEST FOR FIRST ARRIVAL AT AN ASSEMBLE BLOCK

245	TRANSFER	BOTH, 250, 252
250	GATE NM	252
251	TABULATE	9
9	TABLE	1A, 500, 100, 30
252	ASSEMBLE	60



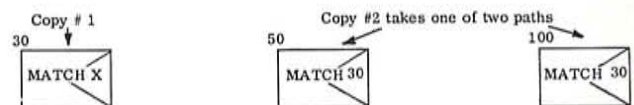
Otherwise, it will simply enter ASSEMBLE block 252. Further examples of MATCH and GATE M blocks are given below.

It should be noted that the status of the partner assembly set transactions is not altered when a transaction enters a GATE M or GATE NM block which tests one of the matching conditions.

Examples of MATCH and GATE M Blocks

Example 1:

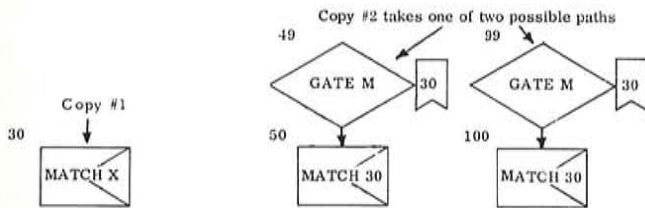
It is necessary to MATCH transaction copy number 1 of an assembly set with copy 2 which might arrive at either of the two blocks 50 or 100. Suppose copy number 1 always arrives at block 30 before copy number 2 arrives at block 50 or block 100. Then the MATCH can be made in the following way:



Copy number 1 arrives at block 30 which references some block X which need not even be a MATCH, ASSEMBLE, or GATHER block. Upon finding that no member of its assembly set is waiting to be MATCHed there, the transaction waits in block 30 in an interrupt status. Copy number 2 arrives at block 50 or block 100 and MATCHes block 30. A successful match is made with copy number 1 and both transactions are able to move to their next sequential blocks.

Example 2:

The preceding example will succeed only if copy number 1 arrives at its MATCH block before copy number 2 arrives at its MATCH block. If copy number 2 arrives first, it will find no MATCH at block 30 and will be delayed. Then when copy number 1 arrives at block 30, it will find no MATCH at block X and will also be delayed. Even if X = 50, perhaps copy number 2 is in MATCH block 100. Both transactions would then be delayed indefinitely. This can be prevented by use of GATE M blocks:



Copy number 2 is not allowed to enter block 49 or block 99 until a matching condition exists in block 30. Thus, copy number 1 will always arrive at block 30 before copy number 2 arrives at block 50 or block 100.

PRIORITY BLOCK TO CHANGE PRIORITY LEVEL OF TRANSACTIONS

2	LOC	7	8	OPERATION	19	A	B			
				PRIORITY		Priority no. k, *n, SNAj, SNA*n	[BUFFER]			
							<table border="1"> <tr> <td>A</td> </tr> <tr> <td>PRIORITY</td> </tr> <tr> <td>BUFFER</td> </tr> </table>	A	PRIORITY	BUFFER
A										
PRIORITY										
BUFFER										

The PRIORITY block is used to set the priority of a transaction to a specified value. The priority of a transaction is significant in determining when the transaction will be processed by the overall GPSS/360 scan and, thus, when it will obtain any equipment for which it has been delayed.

Remember that field E of GENERATE blocks can specify the priority level with which

transactions will enter a simulation model. If no field E priority level is specified in a GENERATE block, the transactions have a zero priority level. The priority level of a transaction remains constant until it enters a PRIORITY block.

The PRIORITY block will never refuse entry to a transaction. Transactions proceed to the next sequential block. The value of the field A argument specifies the priority level to be assigned, which may be any value between 0 and 127 inclusive. In GPSS/360, if the user specifies a priority level greater than 127, Execution Error 614 will be given.

BUFFER Option

The overall GPSS/360 scan will generally attempt to move the current transaction into some next block. However, if the word BUFFER is coded in field B, the PRIORITY block becomes the equivalent of a BUFFER block after setting the new priority level. Therefore, the overall GPSS/360 scan transfers back to process the first transaction in the current events chain. Since the BUFFER block has no time delay, the transaction that enters it will be processed again by the program later on in the same clock instant. The BUFFER option is explained further under the discussion of "BUFFER Block" later in this chapter.

Internal Operation of PRIORITY Block

The PRIORITY block assigns the new priority level to byte T7, then places the transaction last in the new priority class in the current events chain (See Figure 16). This operation occurs even if the new priority level is the same as the entering priority level. There may be cases where the analyst finds it desirable to shift the location of a transaction within its current priority class. For instance, this might be done just before entering a SPLIT block. Since the transaction is now the last one in its priority class, the newly created SPLIT transactions will immediately follow the parent transaction in the current events chain, as each is added at the end of the priority class of the parent transaction.

The status change flag is automatically set whenever a transaction enters a PRIORITY block. Consequently, the overall GPSS/360 scan will automatically transfer back to the start of the current events chain when it finishes moving the transaction which entered the PRIORITY block. This is necessary because the priority level

may have been reduced, thereby shifting both the transaction and the overall GPSS/360 scan to a lower priority level in the current events chain. The GPSS/360 scan thereby bypasses all the transactions between the former and new positions of the PRIORITY block transaction in the current events chain. Setting the status change flag to "on," however, ensures that the overall GPSS/360 scan will process such bypassed transactions.

With the BUFFER option, the overall GPSS/360 scan transfers immediately back to the start of the current events chain. The status change flag is also immediately reset to zero.

BUFFER Block

2	LOC	7	8	OPERATION	19
				BUFFER	

BUFFER

The BUFFER block provides a direct means of stopping the processing of the current transaction and, instead, transfers the overall GPSS/360 scan back to the start of the current events chain. The normal mode of the overall GPSS/360 scan is to advance each transaction through as many blocks as possible, until one of the following conditions occur:

1. A positive time ADVANCE block is encountered, in which case the transaction is merged into the future events chain.
2. All specified next blocks cannot be entered.
3. The transaction enters a MATCH, ASSEMBLE, or GATHER block, which either:
 - a. Destroys the transaction (last n-1 transactions into ASSEMBLE block), or
 - b. Places transaction in matching condition (MATCH and GATHER blocks; first transaction into ASSEMBLE block).
4. The transaction is destroyed in a TERMINATE block.

Normally, one of the above events must occur before the overall GPSS/360 scan will proceed to some other transaction. The BUFFER block is an exception to this rule. After a transaction enters a BUFFER block, the overall GPSS/360 scan will return to the first (highest priority) transaction in the current events chain. Since the BUFFER block has no time delay, the transaction that enters it will always be processed by the scan later on in the same clock instant. The BUFFER block will never refuse entry to a transaction. Transactions move to the next sequential block following the BUFFER block.

Observe that if the transaction which is entering the BUFFER block is the first one in the current events chain the BUFFER block is redundant and performs no useful function. Also, observe that none of the transactions which follow the BUFFER block transaction in the current events chain will be processed before the overall GPSS/360 scan returns to the BUFFER block transaction.

A PRIORITY block, with BUFFER coded in field B, also behaves like a BUFFER block after it has placed the transaction at the end of the priority class specified by the field A argument. This BUFFER option immediately transfers the overall GPSS/360 scan back to the start of the current events chain. Since the PRIORITY block likewise has no time delay, the transaction whose priority level was just changed will always be processed by the scan later on in the same clock instant.

The following pair of PRIORITY blocks is a simple means by which every transaction in the current events chain will be processed before a given transaction:

	ASSIGN	1, PR
k	PRIORITY	0, BUFFER
k + 1	PRIORITY	P1

PRIORITY block k places the transaction which has a priority level of $p = 0, 1, \dots, 127$, at the very end of the current events chain with a priority level of zero. The current priority level of the transaction has been saved in Parameter 1 so that it may be restored when the transaction enters block k + 1. The BUFFER option transfers the overall GPSS/360 scan back to the start of the current events chain. The overall GPSS/360 scan finally works its way through the current events chain and once again encounters the transaction in PRIORITY block k. This transaction enters PRIORITY block k + 1, which restores the transaction to its initial priority level which is still in Parameter 1.

The overall GPSS/360 scan is described in further detail below. Several examples of BUFFER blocks and PRIORITY blocks with BUFFER options are given later in this chapter.

OVERALL GPSS/360 SCAN

The overall GPSS/360 scan is outlined in Figure 23. The scan can be divided into three major phases:

1. Update clock to next most imminent block departure time in future events chain. Move all Transactions with this block departure time (BDT) into current events chain (Figure 24) from future events chain.

2. Scan each transaction sequentially in the current events chain (Figure 25).

3. If scan indicator of a transaction is off, attempt to move the transaction into some next block (Figure 26).

Update Clock (Figure 24)

At each clock time the overall GPSS/360 scan will eventually move all the way through the current events chain and complete processing the last transaction in the lowest priority class. The GPSS/360 simulation model is now, generally, in a state where no further transactions can move into some next block (exceptions to this rule will be discussed later).

The overall GPSS/360 scan now transfers to UPDATE CLOCK (see Figure 24). The absolute clock is advanced to the block departure time of the first (most imminent) transaction in the future events chain. This first transaction is unlinked from the future events chain and added to the current events chain as the last transaction in its priority class. The priority level of the transaction (0, 1, . . . 127) is stored in byte T7.

The scan then examines the second transaction in the future events chain to see if it has the same block departure time (word T4) as the new absolute clock time. If it is the same, the second transaction is likewise added to the current events chain as the last transaction in its priority class. This process continues until either a transaction with a block departure time greater than the new absolute clock time is encountered, or the future events chain is exhausted. The overall GPSS/360 scan now transfers to "Start Overall GPSS/360 Scan" in Figure 25.

The future events chain transactions which are transferred to the current events chain are of two types:

1. Transactions which have been delayed in positive time ADVANCE blocks.

2. Incipient successor transactions, which have been waiting to enter the simulation model at some GENERATE block.

A third highly specialized type of transaction will also be encountered in the future events chain. These are "dummy transactions" which trigger the tabulation of the arrival rate argument of tables

operating in the RT mode or the entry of transactions into the system when jobtapes are used. These transactions are immediately merged back into the future events chain when they are encountered.

Start Overall GPSS/360 Scan (Figure 25)

The overall GPSS/360 scan will start with the first (highest priority) transaction on the current events chain under four conditions:

1. The absolute clock has just been updated and all new transactions have been added to the current events chain from the future events chain. (This was described in Figure 24.)

2. A transaction has entered a BUFFER block (see Figure 26).

3. A transaction has entered a PRIORITY block with a BUFFER option (see Figure 26).

4. The status change flag has been set by a transaction and is subsequently found to be on by the overall GPSS/360 scan (see Figure 26).

The GPSS/360 scan tests the scan indicator (bit 0 of byte T10) of each successive transaction until it finds a transaction whose scan indicator is off; such a transaction is available for processing. The GPSS/360 scan then transfers to "Try to Move Transaction" in Figure 26. Several things may then happen to this transaction in Figure 26:

1. The transaction may fail to advance into its scheduled next block.

2. The transaction may advance through one or more zero time blocks until:

- a. it is destroyed (in a TERMINATE or ASSEMBLE block).

- b. it enters a positive time ADVANCE block and is merged into the future events chain.

- c. it is blocked trying to enter some next block.

- d. it is placed in an interrupt matching condition in a MATCH, ASSEMBLE, or GATHER block.

- e. it enters a BUFFER block or a PRIORITY block with BUFFER option.

The end result of each of the above five conditions is that the overall GPSS/360 scan voluntarily ceases or is unable to move the current transaction any further in Figure 26. The next transaction which the GPSS/360 scan processes will then be either:

1. the first current events chain transaction ("Start Overall GPSS/360 Scan" in Figure 25.)

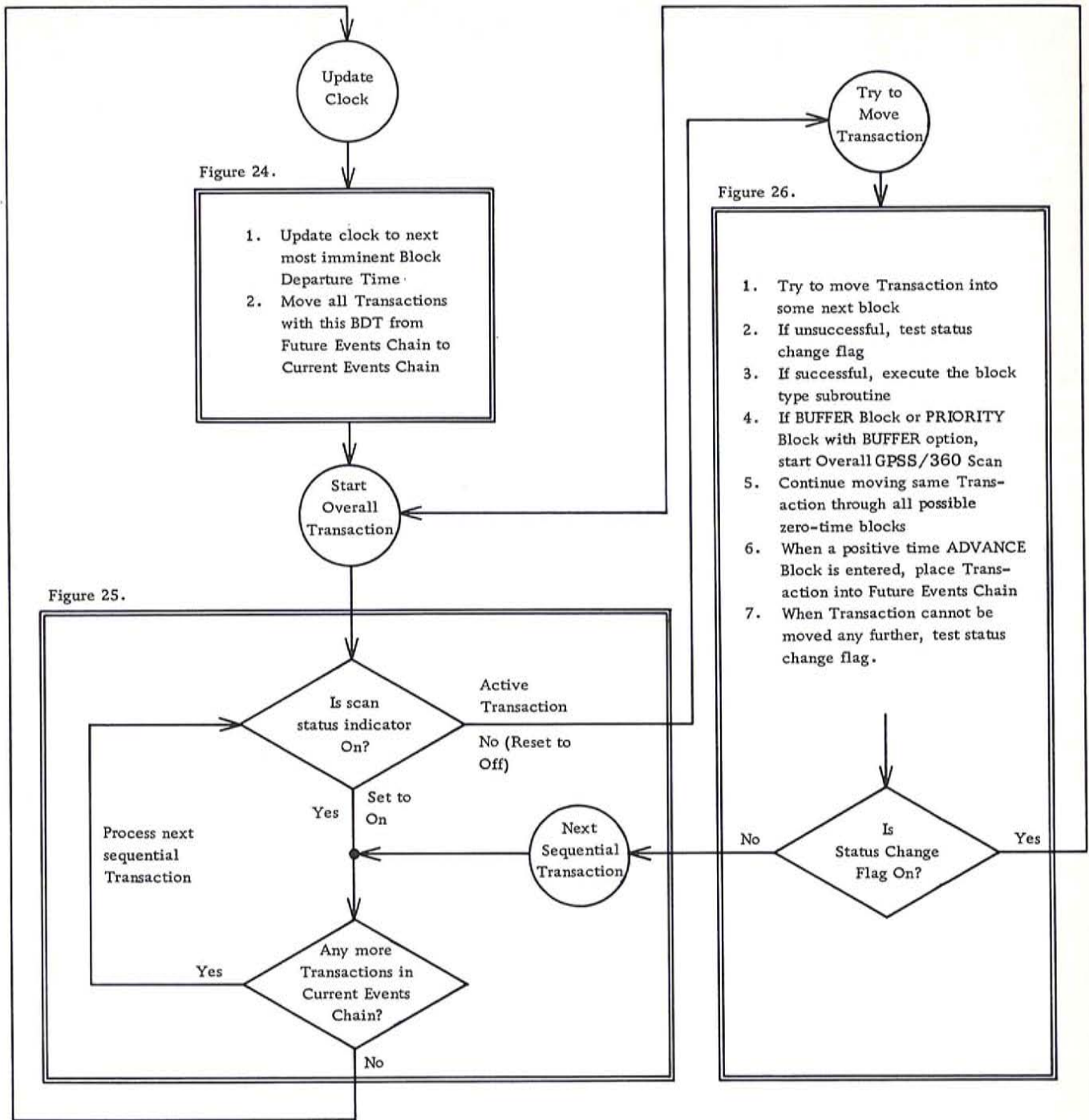


Figure 23. Overall GPSS/360 Scan

2. the "Next Sequential Transaction" in Figure 25.

Eventually, the GPSS/360 scan comes to the previously described point (discussed earlier in this chapter) where it finishes processing the last (lowest priority) transaction in the current events chain. (The GPSS/360 scan then transfers to "Update Clock" in Figure 24.

Try To Move Transaction (Figure 26)

Whenever the overall GPSS/360 scan finds a transaction whose scan indicator (bit 0 of byte T10) is off, the GPSS/360 scan will attempt to move the transaction to some next block. Byte T10 also indicates the type of next block trial. The next choice will be made in one of three ways:

1. Only one next block is to be tried (block

number stored in T5 halfword).

2. The transaction is in a TRANSFER block with a BOTH selection mode (The first next block number is stored in T5, while the second next block number is stored in T1).

3. The transaction is in a TRANSFER block with an ALL selection mode (The first next block number is stored in T5, while the last next block number is stored in T1).

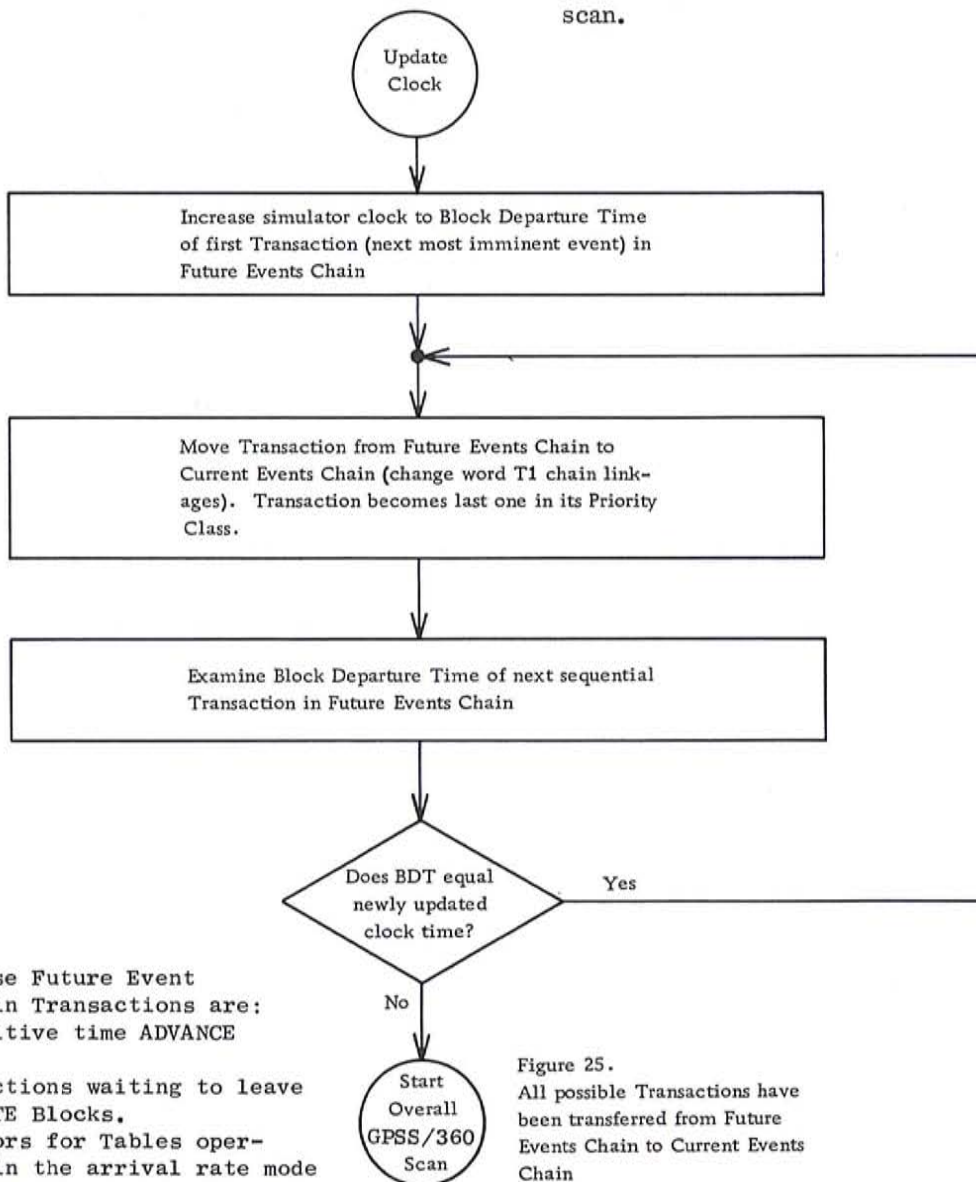
In the SF column of the transaction printout, blank, B, or A is printed out for only-one-next-block selection mode, BOTH selection mode, or ALL selection mode, respectively.

Unique Blocking Conditions

Assume that the transaction cannot move into some

next block, as shown in the upper right-hand corner of Figure 26. In most cases, there will be a unique blocking condition. The transaction will then be linked, as the first transaction, in one of eleven different types of pushdown delay chains (see Chapters 9, 10, and 11 on the delay chains associated with logic switches, facilities, and storages).

The scan indicator (bit 0 of byte T10) of the transaction will be set "on". The overall GPSS/360 scan will, therefore, not attempt to move this transaction (see Figure 25). This transaction and any other members of its delay chain will not be reactivated (i. e., bit 0 of byte T10 will not be reset to "off") until the unique blocking condition is removed. This reactivation feature contributes materially to the speed of the overall GPSS/360 scan.



- Note: These Future Event Chain Transactions are:
1. In positive time ADVANCE Blocks
 2. Transactions waiting to leave GENERATE Blocks.
 3. Operators for Tables operating in the arrival rate mode

Figure 25. All possible Transactions have been transferred from Future Events Chain to Current Events Chain

Figure 24. Overall GPS/360 Scan: Update Clock to Next Most Imminent Event

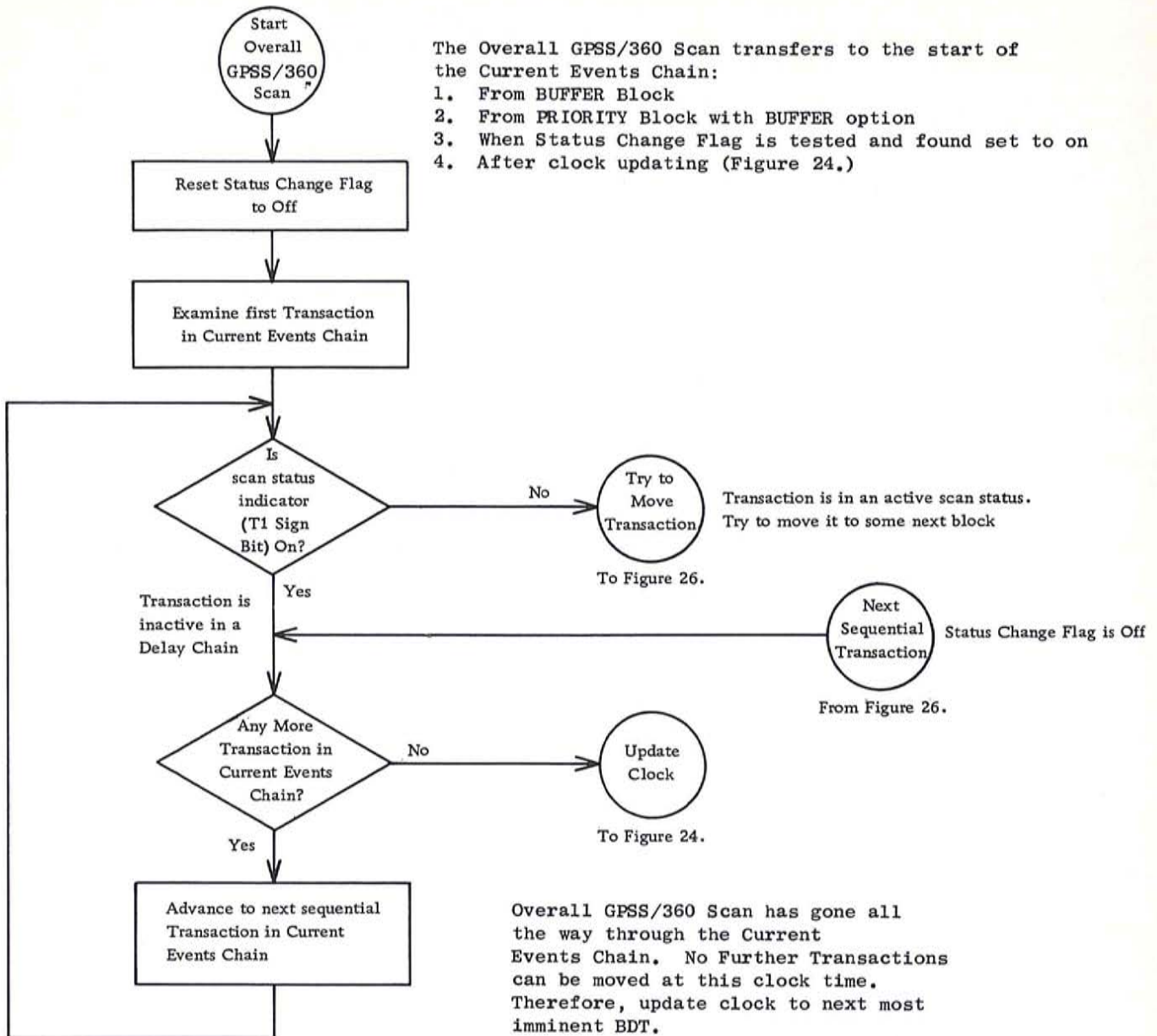


Figure 25. Overall GPSS/360 Scan: Scan of Current Events Chain (Start of Scan)

Unique blocking and subsequent scan deactivation in a delay chain will occur when a transaction tries to enter the following block types:

1. SEIZE block
2. PREEMPT block
3. ENTER block
4. Any GATE block except M and NM

Nonunique Blocking Conditions

A nonunique blocking condition may occur in trying to enter the following block types:

1. A TEST block, which involves a relation (E,

NE, L, LE, G, GE) between two Standard Numerical Attributes.

2. GATE M and NM blocks, which reference matching conditions of assembly set members at a specified block.

The most common nonunique blocking condition occurs, however, within TRANSFER blocks with a BOTH or ALL selection mode. Here, either both or all of them do. The blocks which deny entry can be any combination of SEIZE, PREEMPT, ENTER, GATE, and TEST blocks.

The crucial result of a nonunique blocking

condition is that the scan indicator (bit 0 of byte T10) of the transaction is not set, and the transaction is not linked to a delay chain. The overall GPSS/360 scan will always try to advance these blocked transactions each time that it encounters them in its scan of the current events chain, even though there has been no change in the nonunique blocking conditions. Such blocked transactions can, therefore, materially slow down a GPSS/360 model. TEST, GATE M, GATE NM and TRANSFER blocks with BOTH or ALL selection modes must, therefore, be used with discretion.

Some Next Block Can Be Entered

Assume now that some next block can be entered; this is a condition which is represented by the lefthand side of Figure 26. The block type subroutine is executed as soon as the transaction enters the block. The GENERATE block is the only block type where anything happens when a transaction leaves a block (e.g., an incipient successor transaction is created).

If the transaction has entered a BUFFER block, or a PRIORITY block with a BUFFER option, the overall GPSS/360 scan immediately transfers to start overall GPSS/360 scan (Figure 25).

If the transaction enters a TERMINATE, MATCH, ASSEMBLE, or GATHER block, the overall GPSS/360 scan may release control of the transaction. This will occur when:

1. the transaction is placed in a matching condition because:
 - a. it is the initial transaction into an ASSEMBLE block,
 - b. it is one of the n transactions in a GATHER block,
 - c. it enters a MATCH block and fails to find a matching transaction in the specified conjugate MATCH block;
2. the transaction is destroyed in a TERMINATE block, or in an ASSEMBLE block (the last n-1 transactions in the assembly set).

The overall GPSS/360 scan transfers to a test of the status change flag, as described later in this chapter. See Diagram: Figure 26).

Setting of the Status Change Flag

Various block types will cause the status change flag to be set. With the exception of the BUFFER and PRIORITY BUFFER blocks, the overall GPSS/360 scan will eventually test the status change

flag after it has ceased moving the current transaction. If the flag is set, the GPSS/360 scan transfers back to "Start Overall GPSS/360 Scan" (Figure 25). If the flag is off, the GPSS/360 scan simply proceeds to "Next Sequential Transaction" (Figure 25).

Table 14 describes the various conditions under which the status change flag is set. There are essentially four basic types of conditions:

1. The flag is unconditionally set in a SEIZE, RELEASE, PREEMPT, RETURN, ENTER, or LEAVE block, which changes the status of a logic switch, facility, or storage. This unconditional setting will ensure that all transactions which are being delayed by SEIZE, PREEMPT, ENTER, and GATE blocks will be moved if the entity status change removes the blocking condition.
2. The flag is set by the completion of matching, assembling, or gathering operations in MATCH, ASSEMBLE, or GATHER blocks.
3. The flag is set by a PRIORITY block. Observe that if the PRIORITY block has a BUFFER option, the overall GPSS/360 scan will immediately transfer back to the start of the current events chain, thereby resetting the status change flag to "off".
4. A transaction(s) is unlinked from a user chain by an UNLINK block.

Positive Time ADVANCE Block

Another major possibility is that the transaction has entered a positive time ADVANCE block. In this case, the overall GPSS/360 scan merges the transaction in the future events chain according to its new block departure time (see "Internal Operation of ADVANCE Block" earlier in this chapter). The GPSS/360 scan then transfers to a test of the status change flag as described below.

Zero Time Block

If the overall GPSS/360 scan still has control of the transaction (i.e., there has been no buffering, termination, or matching interruption) and the block is a zero time non-ADVANCE block type, the overall GPSS/360 scan will attempt to move the transaction into still another next block. The majority of block moves in GPSS/360 models are of this nature. The overall GPSS/360 scan generally moves transactions through a number of zero-time block types until it finally surrenders control of the transaction under the various conditions which are described in the preceding pages.

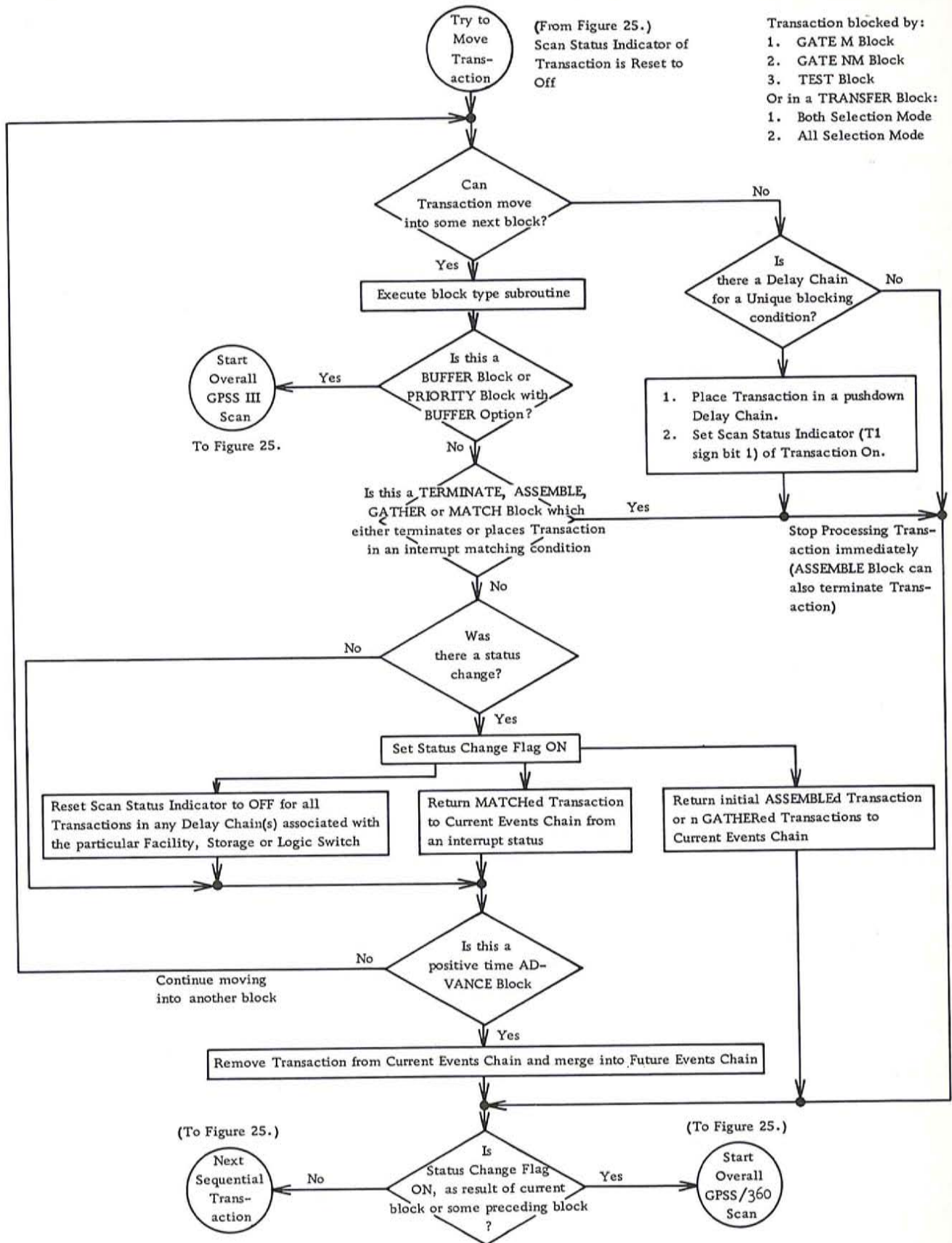


Figure 26. Overall GPSS/360 Scan: Try to Move Individual Transaction into Some Next Block

Testing of the Status Change Flag

Table 14 shows the various block moves under which the status change flag is set. It is important to recognize that, even though the transaction which is currently being processed has set the status change flag on because of the above block moves, the overall GPSS/360 scan will not necessarily test the status change flag after each block move. The lower left-hand corner of Figure 26 shows that, if the current block is a zero-time block type, the overall GPSS/360 scan will first continue to move the transaction into the next block. This is the usual case.

Figure 26 shows that the overall GPSS/360 scan will stop moving the current transaction and transfer to a test of the status change flag under the following conditions:

1. The current transaction is blocked from entering all possible next blocks.
2. Processing of the current transaction is suspended because:
 - a. it is destroyed in a TERMINATE or ASSEMBLE block.
 - b. it is placed in a matching condition in MATCH, ASSEMBLE, or GATHER block.
3. The current transaction enters a positive time ADVANCE block and is merged into the future events chain.
4. The current transaction completes an assembly or gathering operation in an ASSEMBLE or GATHER block.

Results of Status Change Flag Test. The overall GPSS/360 scan will transfer to two possible locations:

1. If the status change flag has been set on, the GPSS/360 scan transfers to processing of the first transaction in the current events chain at "Start Overall GPSS/360 Scan" (Figure 25). The flag is immediately reset to off.
2. If the flag is off, the GPSS/360 scan simply proceeds to the "Next Sequential Transaction" (Figure 25).

Table 14 summarizes the block operations which set the status change flag to on.

EXAMPLES OF BUFFER AND PRIORITY BUFFER BLOCKS

The following examples show some of the subtle problems which can be caused by the particular nature of the overall GPSS/360 scan:

Example 1:

Since the GPSS/360 scan attempts to move the current transaction through as many blocks as possible, certain logical situations are difficult to represent. In model A shown below, Queue 2 will always be empty. Assume that all transactions have the same priority level. Assume also that a transaction, which has SEIZED Facility 1 in block 11, leaves the positive-delay ADVANCE block 13. This transaction becomes the last one in its priority class in the current events chain.

The overall GPSS/360 scan moves the transaction through block 14, which RELEASES Facility 1 for use by another transaction. Transactions may have been blocked in QUEUE block 10 and put into a delay chain because Facility 1 was in use. Their scan indicators (first bit of byte T10) will be reset to off, i. e., to an active scan status. The status change flag is set "on". However, the transaction continues to move in zero-time, enters QUEUE block 15 and immediately succeeds in reSEIZING Facility 1. It then proceeds to spend a positive time in ADVANCE block 18.

The overall GPSS/360 scan finally tests the status change flag, finds it on, and transfers to the start of the current events chain. The flag is reset to off. The scan eventually encounters each one of the previously delayed transactions in QUEUE block 10, which are now in an active scan status in the current events chain. The GPSS/360 scan attempts to move each transaction into SEIZE block 11, fails to do so, and thus puts these transactions back into the same pushdown delay chain. A great deal of processing has thus accomplished nothing.

TABLE 14: BLOCK OPERATIONS WHICH SET THE STATUS FLAG TO ON

<ol style="list-style-type: none"> 1. SEIZE 2. RELEASE 3. PREEMPT 4. RETURN 5. ENTER 6. LEAVE 7. LOGIC 	}	<p>The status change flag is unconditionally set as a result of a status change in a Logic Switch, Facility, or Storage.</p>
<ol style="list-style-type: none"> 8. MATCH Block Status Change 		

The current transaction which is entering the MATCH block finds an assembly set mate in a matching interrupt condition at the conjugate MATCH block which is specified in the field A of the MATCH block. The assembly set mate is returned to the current events chain at the end of its priority class. The status change flag is set and the current transaction attempts to move on to some next block.

9. ASSEMBLE Block Status Change

The current transaction completes the assembly operation, i.e., the T2 word of the initial transaction to enter the ASSEMBLE block has been reduced to zero. The current transaction is destroyed and the initial transaction is removed from a matching interrupt condition and is returned to the current events chain at the end of its priority class. The status change flag is set. The overall GPSS/360 scan immediately transfers to a test of the flag, finds it on, and transfers to "Start Overall GPSS/360 Scan" (Figure 25.)

10. GATHER Block Status Change

The current transaction completes the gathering operation, i.e., the T2 word of the initial transaction to enter the GATHER block has been reduced to zero. All the preceding n-1 transactions are removed from a matching interrupt condition and are returned to the current events chain at the end of their respective priority classes. This occurs in the same order as the transactions arrived in the GATHER block. The current transaction is first removed from the current events chain and becomes the last transaction which is added back to the current events chain. The status change flag is set. The overall GPSS/360 scan immediately transfers to a test of the flag, finds it on, and transfers to "Start Overall GPSS/360 Scan" (Figure 25).

11. PRIORITY Block Status Change

The status change flag is unconditionally set. The overall GPSS/360 scan will either transfer back to the start of the current events chain when it finishes processing the transaction, or it will transfer immediately if the PRIORITY block has a BUFFER option.

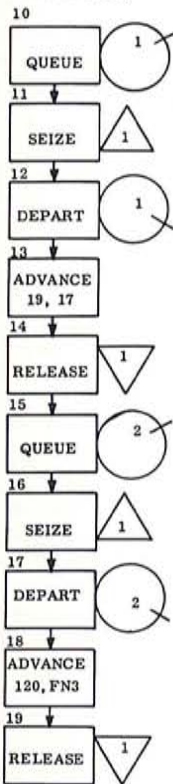
12. UNLINK Block Status Change

The status change flag is unconditionally set when one or more transactions are UNLINKed from a user chain. This ensures that these transactions will be processed at the same clock time at which they are returned to the current events chain.

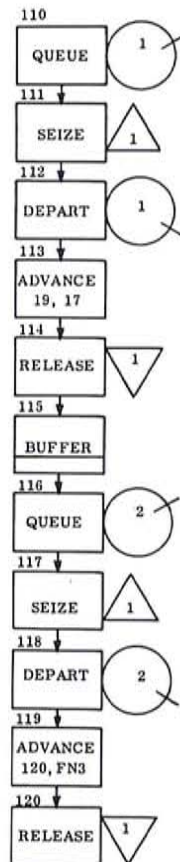
Now, consider what BUFFER block 115 accomplishes in model B. Assume again that a SEIZEing transaction leaves ADVANCE block 113, and is placed at the end of its priority class in the current events chain. The transaction releases Facility 1 in block 114, at which time any blocked delay chain transactions in QUEUE block 110 are reactivated. The transaction then moves into BUFFER block 115.

The overall GPSS/360 scan immediately stops processing the transaction and transfers back to the start of the current events chain. The GPSS/360 scan now succeeds in moving that transaction which has been delayed for the longest time into SEIZE block 111. This transaction then moves into positive time ADVANCE block 113.

MODEL A
(No BUFFER Block)



MODEL B
(With BUFFER Block)



The GPSS/360 scan may then encounter other previously delayed transactions. It fails again to move them into SEIZE block 111, and puts them back into the SEIZEing delay chain on Facility 1. The last transaction which is processed by the GPSS/360 scan is the original transaction which is still in BUFFER block 115. This transaction moves

into QUEUE block 116 and now fails to enter SEIZE block 117. The original transaction will not be able to SEIZE Facility 1 in block 117 until all of the previously delayed transactions, which preceded it in the current events chain, have SEIZED Facility 1 by entering block 111.

Note, however, that each of these transactions also goes through BUFFER block 115 and repeats the operations which were previously undergone by the original transaction. As these transactions leave ADVANCE block 113, they are placed into the current events chain as the last transaction in the particular priority class. Consequently, these transactions will not be able to enter SEIZE block 117 until the preceding transactions have done so.

Queue 2 in model B obviously has nonzero contents as opposed to model A. It is quite possible that, during all of the above block moves, additional transactions enter QUEUE block 110. Depending on their relative locations in the current events chain, these transactions may SEIZE Facility 1 in block 111 before the previous transactions SEIZE Facility 1 in block 117. These transactions will, therefore, prolong the delays in Queue 2 at block 116.

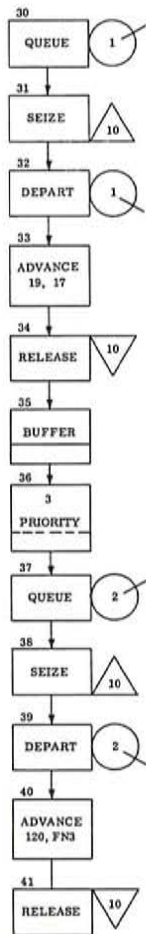
Different Priority Levels for Transactions

Example 1 has assumed thus far that all the transactions have the same priority level. Assume now that there are different priority levels. The BUFFER block may now have a different effect on the sequence in which transactions SEIZE Facility 1. As transactions are returned to the current events chain in ADVANCE block 113, they are placed at the end of their particular priority class. Assume that such a transaction is the only one with a priority level of 7. Assume that the other transactions which are being delayed in QUEUE block 110 have priorities of six or less. Even though the overall GPSS/360 scan transfers back to the start of the current events chain when the priority 7 transaction enters the BUFFER block, the GPSS/360 scan immediately picks up this transaction before encountering any of the lower priority transactions. This transaction, therefore, succeeds in reSEIZEing Facility 1 in zero time at block 117, just as if the BUFFER block was not in the model. The situation is, of course, completely different if some of the transactions which have been delayed in QUEUE block 110 have a higher priority than the SEIZEing transaction which leaves ADVANCE block 113 and enters the BUFFER block. When the overall GPSS/360 scan transfers back to the

start of the current events chain, it encounters a higher priority transaction and moves it into SEIZE block 111. When the GPSS/360 scan comes back to the transaction in the BUFFER block, it is only able to move it as far as QUEUE block 116 before the transaction fails to enter SEIZE block 117.

Example 2:

The model below illustrates a method by which a transaction may be placed "second in line" in the set of transactions which are attempting to SEIZE a facility. It is assumed that the transactions in QUEUE block 30 have Priority 2. The BUFFER block (35) forces the transaction which has just left the RELEASE block (34) to pause while the overall GPSS/360 scan returns to the start of the current events chain. If there are any transactions in QUEUE block 30 at that moment, the Priority 2 transaction which has been delayed the longest time will SEIZE Facility 10 in block 31.



LOCATION	OPERATION	A	B	C					
1	2	7	8	13	19	17			
* TRANSACTIONS LEAVING BLOCK 34 ARE SECOND									
* IN LINE TO USE FACILITY 10 (AT BLOCK 38)									
30	QUEUE	1							
31	SEIZE	10							
32	DEPART	1							
33	ADVANCE	19	17						
34	RELEASE	10							
35	BUFFER								
36	PRIORITY	3							
37	QUEUE	2							
38	SEIZE	10							
39	DEPART	2							
40	ADVANCE	120	FNS						
41	RELEASE	10							

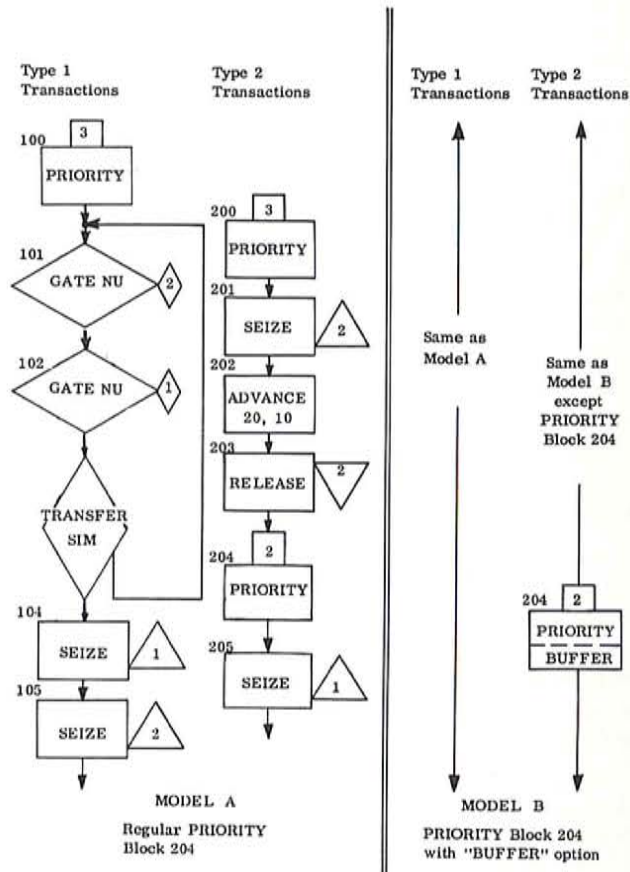
The GPSS/360 scan then fails to advance the other delayed transactions and puts them back into the deactivated SEIZEing delay chain of Facility 10.

The scan finally returns to the transaction in the BUFFER block. This transaction will then enter PRIORITY block 36, receive a priority level of 3, and thus be ahead of all the Priority 2 transactions in QUEUE block 30. This transaction is then blocked from SEIZEing the Facility in block 38.

Each transaction that exits from RELEASE block 34 will thus regain control of Facility 10 in SEIZE block 38 after one other transaction has SEIZED the facility at Block 31. If, however, there were no delayed transactions in block 30, the transaction RELEASing Facility 10 in block 34 will RESEIZE the facility in zero time in block 38.

Example 3:

Consider model A below. Type 1 transactions have their priority level set to three in PRIORITY block 100. They must then find Facilities 1 and 2 not in use simultaneously before they can succeed in SEIZEing them in blocks 104 and 105. The



TRANSFER SIM block 103 ensures that these simultaneous conditions are met.

Type 2 transactions also have their priority level set to three in PRIORITY block 200, after which they SEIZE Facility 2. After leaving ADVANCE block 202 and RELEASing Facility 2, their priority level is reduced to two with respect to SEIZING Facility 1 in block 205. However, once again the overall GPSS/360 scan will continue to move these Priority 2 transactions in zero time into SEIZE block 205, even though Type 1 transactions with a priority of three are waiting in blocks 101-103 to SEIZE Facility 1.

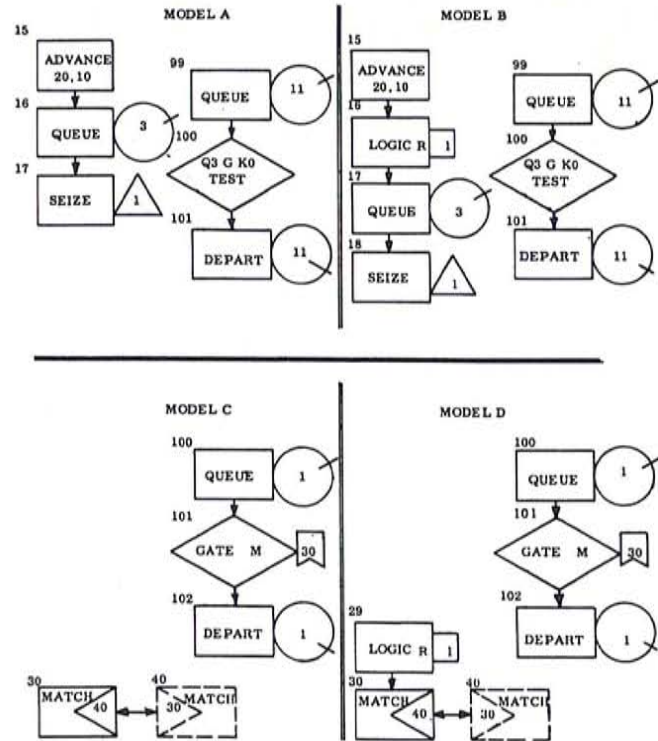
The solution is quite simple as shown in model B. The BUFFER option is coded in field B of PRIORITY block 204. The overall GPSS/360 scan returns to the start of the current events chain after placing the Type 2 transaction at the end of Priority Class 2. The GPSS/360 scan thereby succeeds in moving one of the Type 1 transactions into SEIZE blocks 104 and 105.

Example 4:

It is quite possible that the overall GPSS/360 scan will fail to move all possible transactions that could move at a particular clock time. The GPSS/360 scan may get to the last transaction in the current events chain, process it, and then transfer to the update clock routine (Figure 24) while other transactions are still able to move in the current events chain.

As an example, consider model A. Assume that the last transaction in the current events chain has just left ADVANCE block 15. It enters QUEUE block 16, but fails to enter SEIZE block 17 because another transaction is already using Facility 1. Assume further that this transaction has become the only one that is contributing to the contents of Queue 3; i.e., the Standard Numerical Attribute Q3 equals 1.

The overall GPSS/360 scan tests the status change flag, finds that it has not been set to "on," and tries to process the next sequential transaction. There is none, so the GPSS/360 scan transfers to update clock in Figure 24. Assume now that some other transaction has been delayed in QUEUE block 99 because the relational statement in TEST block 100 (Q3 is greater than zero) has been false. When the first transaction entered QUEUE block 16, Q3 became equal to one. Consequently, the relational statement in TEST block 100 is now true, and the second transaction should move into the TEST block.



* LOCATION	7	OPERATION	A	B	C
1	2	8	13	19	
* MODEL A - POSSIBLE FAILURE TO MOVE ALL POSSIBLE TRANSACTIONS					
15		ADVANCE	20	10	
16		QUEUE	3		
17		SEIZE	1		
99		QUEUE	11		
100		TEST G	Q3	K Φ	
101		DEPART	11		
* MODEL B - GUARANTEE OF MOVING ALL TRANSACTIONS					
15		ADVANCE	20	10	
16		LOGIC R	1		
17		QUEUE	3		
18		SEIZE	1		
99		QUEUE	11		
100		TEST G	Q3	K Φ	
101		DEPART	11		
* MODEL C - POSSIBLE FAILURE TO MOVE ALL POSSIBLE TRANSACTIONS					
30		MATCH	40		
40		MATCH	30		
100		QUEUE	1		
101		GATE M	30		
102		DEPART	1		
* MODEL D - GUARANTEE OF MOVING ALL TRANSACTIONS					
29		LOGIC R	1		
* BALANCE IS SAME AS MODEL C					

However, the overall GPSS/360 scan never returns to the start of the current events chain during the current clock time. The second transaction may

only be able to enter the TEST block at the next time to which the clock is updated.

Consider a similar problem in model C. Assume that a transaction has been delayed in entering GATE M block 101 because a member of its assembly set is not in a matching condition at MATCH block 30. Assume now that the last transaction in the current events chain, which belongs to this same assembly set, enters MATCH block 30. It finds that no other members of the particular assembly set is in the conjugate MATCH block 40. This transaction is then put in a matching interrupt state.

The transaction in QUEUE block 100 should now succeed in moving into GATE M block 101 because a member of its assembly set is now in a matching condition in MATCH block 30. However, this never happens because the overall GPSS/360 scan, after placing the block 30 transaction into a matching interrupt condition, immediately transfers to a test of the status change flag. It has not been set to "on" and the GPSS/360 scan tries to process the next sequential transaction. There is none, so the GPSS/360 scan transfers to the update clock routine (Figure 24) and the transaction in block 100 fails to move at the current clock time.

Models B and D show one simple solution to the above problem. LOGIC blocks 16 and 29 are inserted into the models just in front of SEIZE block 17 and MATCH block 30. Some logic switch (e.g., #1) which is not used elsewhere in the model, is reset by this LOGIC block and the status change flag is thus set to "on." Consequently, when the overall GPSS/360 scan stops processing the current transaction, it finds the flag on and hence transfers back to the start of the current events chain. This assures that the transactions which have been delayed in QUEUE blocks 99 and 100 will succeed in moving at the same clock time.

These LOGIC blocks, which do not simulate any real physical or logical action in the GPSS/360 model, behave as if they were a special SETFLAG block. There is a major difference, however, between these LOGIC blocks and the BUFFER block. While the overall GPSS/360 scan transfers immediately to the start of the current events chain in a BUFFER block, with the LOGIC block it will only transfer when it ceases to move the current transactions and tests the status change flag.

Although it is possible to change model A in other ways to avoid the use of the LOGIC block, it is almost imperative to use the LOGIC block in model D.

JOB TAPES

In parametric simulation studies, it is often desirable to have a given "constant" source of transactions which can be used as input to the various models under study. This would guarantee that transactions enter a given point at the same instant of time in all models under consideration. GPSS/360 provides the means of creating and using such a transaction tape.

WRITE Block

The function of the WRITE block is to set up certain information associated with the entering transaction, and write this information on magnetic tape. This information is written as a 416 byte record on the tape specified by the A-argument of the WRITE block. The A-argument must specify one of three tapes: JOBTA1, JOBTA2, or JOBTA3.

The symbol for the WRITE block is



8 OPERATION	19A
WRITE	JOBTA1
	JOBTA2
	JOBTA3

The WRITE block will never refuse entry to a transaction.

Operation of the WRITE Block

1. When a transaction enters a WRITE block, the difference between the current clock time and the time when the previous transaction entered the WRITE block is calculated. The difference is the interarrival time.

NOTE: The first transaction entering a WRITE block ignores step 1 and uses 0 as the interarrival time.

2. The current clock time is set in B2 of the WRITE block as the time when a transaction last entered the WRITE block.

3. The difference between the current clock time and the mark time associated with the entering transaction is calculated. This is the transit time.

4. The interarrival time, transit time, priority, number of parameters, and parameter values of the entering transaction are written on the tape specified by the A-argument of the WRITE block.

5. The transaction proceeds to the next sequential block. It is possible for several WRITE blocks to write on the same tape, and that up to three tapes may be used in a single simulation. It is not absolutely necessary to create a transaction tape by means of WRITE blocks. Other programs could be used to write information in the format previously described.

GPSS/360 control cards affecting tapes which are being used by WRITE blocks are the JOB, END, and CLEAR control cards. If a tape(s) is being used by a WRITE block, the JOB or END control card will cause an EOF to be written on the associated tape(s). The tape(s) will then be rewound and unloaded.

The CLEAR control card will cause an EOF to be written on all tapes referenced as the A-argument of WRITE blocks.

NOTE: The CLEAR control card does not cause the tape(s) to be rewound and/or unloaded.

The control card necessary to make use of the transactions written on the transaction tape is the JOBTAPE control card described in Chapter 15.

Written Information Format:

<u>Word No</u>	<u>Length</u>	
1	4 bytes	Interarrival Time (always zero for first transaction)
2	4 bytes	Transit Time (current clock-mark time of entering transaction)
3	1st byte	Unused
3	2nd byte	Priority of entering transaction
3	3rd byte	Leftmost bit=1 if transaction has fullword parameters
3	4th byte	Number of parameters associated with the transaction
4	Varies	The next 400 bytes are used to store the Parameter values associated with the entering transaction. This requires two bytes/halfword parameter and four bytes fullword parameter.
6		
.		
.		
.		
104		

HELP Block

The HELP block is provided to enable the user to write independent routines which will run in conjunction with the GPSS/360 program. The HELP block is intended only for users who are thoroughly familiar with the internal operation of the GPSS/360 program. The user must also have some S/360 programming knowledge to effectively use the HELP block.

Fields B, C, D, E, F and G of the HELP block may be any Standard Numerical Attribute (SNA), directly or indirectly addressed.

Field A of the HELP block gives the symbolic name of the first executable instruction in the user's HELP routine. This name consists of 2-6 alphameric characters the first of which must be alphabetic.

The sequence of operations associated with the HELP block is as follows:

1. The SNA's specified by fields B-G are evaluated and placed in the first six words of a 25 word area set up at input time.
2. The remaining 19 words are used to store the addresses of GPSS/360 execution routines which the user may wish to use in his HELP routine.
3. The GPSS/360 program then LINK's (OS/360) or FETCH's (DOS/360) to the user's HELP routine by means of the symbolic name coded in Field A of the HELP block.

Format of the HELP Routine

As was previously mentioned, it is assumed that the HELP block user is familiar with S/360 coding. Some basic rules on the coding of the HELP routine are given below followed by appropriate examples.

1. The first executable instruction of the HELP routine must be SAVE (14,12). For example: GPSS/360 HELP block:

First instruction of HELP routine:

```
HELP TEST, X10, V3
TEST SAVE (14, 12)
```

2. The next two instructions (shown below) are used to set up a base register for the HELP routine. Although general register 12 is suggested for this use, any of the 16 general registers except 5, 10, and 15 may be used for this purpose. Example:

```
BALR    12,0
USING   *,12
```

3. The user now begins his HELP routine using the various control words and subroutines passed to him via an address in GR 10.

(Explained later in detail.)

4. The last instruction in the HELP routine must be:

RETURN (14, 12)

This will automatically transfer control back to the GPSS/360 execution phase and send the transaction which entered the HELP block to the next sequential block.

Use of GPSS/360 Control Words and Subroutines

The HELP block subroutine sets up a 25-word table consisting of the decoded HELP block arguments, the address of the GPSS/360 control words and the addresses of various GPSS/360 execution phase subroutines. The address of this table is passed to the user in General Register 10. This enables the user to reference many of the control words, entity words, and subroutines within the GPSS/360 program.

These words may be referenced in a variety of ways, the simplest of which is the absolute (base-displacement) method. The following lists each available value and its displacement from the address in GR 10.

<u>ITEM</u>	<u>DISPLACEMENT</u>
B Field Value	0
C Field Value	4
D Field Value	8
E Field Value	12
F Field Value	16
G Field Value	20
Address of Control Words	24
Address of ADDCUR routine	28
Address of SUBCUR routine	32
Address of ADDFUT routine	36
Address of SUBFUT routine	40
Address of ADDINT routine	44
Address of SUBINT routine	48
Address of STPVAL routine	52
Address of STPVLA routine	56
Address of PRVAL routine	60
Address of PRVALA routine	64
Address of CREAT routine	68
Address of GETCOR routine	72
Address of FLOAT routine	76
Address of UNFLOT routine	80
Address of ERASE routine	84
Address of FRECOR routine	88
Address of DRAND routine	92
Address of DCOD routine	96

If the user wishes to obtain the decoded value of fields B and C of the HELP block in Registers 6 and 7 respectively he would use the following instructions:

L 6,0(10) OBTAIN B-FIELD
L 7,4(10) OBTAIN C-FIELD

Further, if he wishes to branch to the -STPVAL- routine with these values he would add the following instructions:

L 15,52(10) OBTAIN ADDR. OF
-STPVAL-

BALR 5, 15 BRANCH TO -STPVAL-
The GPSS/360 subroutine-STPVAL-would return to the HELP routine via Register 5.

The available subroutines, their purpose, entry conditions, and exit conditions are listed below:

ROUTINE: ADDCUR
PURPOSE: To add a transaction in current event chain at end of its priority class.
ENTRY
CONDITIONS: No. of the transaction to be inserted in Register 9.
EXIT
CONDITIONS: None

ROUTINE: SUBCUR
PURPOSE: To remove a transaction from the current event chain
ENTRY
CONDITIONS: No. of the transaction to be removed in Register 9. NEWN must have current block of transaction which is being removed.
EXIT
CONDITIONS: None

ROUTINE: ADDFUT
PURPOSE: To insert a transaction into the future event chain.
ENTRY
CONDITIONS: No. of the transaction to be inserted in Register 9.
EXIT
CONDITIONS: None

ROUTINE: SUBFUT
PURPOSE: To remove a transaction from the future event chain.

ENTRY

CONDITIONS: No. of the transaction to be removed in Register 9.

EXIT

CONDITIONS: None

ROUTINE: ADDINT

PURPOSE: To place a transaction in an interrupt status.

ENTRY

CONDITIONS: No. of the transaction to be inserted in Register 9.
NOWM - Number of currently active transaction
NOWMAD - Address (T1) of currently active transaction

EXIT

CONDITIONS: None

ROUTINE: SUBINT

PURPOSE: To remove a transaction from an interrupt status.

ENTRY

CONDITIONS: No. of the transaction to be removed in Register 9.

EXIT

CONDITIONS: None

ROUTINE: STPVAL, STPVLA

PURPOSE: To store a quantity in a fullword or halfword parameter

ENTRY

CONDITIONS: Entry at STPVAL assumes:
Parameter number in Register 6.
Value to be stored in Register 7.
NOWM - Number of currently active transaction.
NOWMAD - Address (T1) of currently active transaction.

NOTE: In storing halfword parameters, quantity will be checked for greater than 32767.

Entry at STPVLA assumes:
Parameter number in Register 6.
Value to be stored in Register 7.
Transaction number in Register 8.

EXIT

CONDITIONS: None

ROUTINE: PRVAL, PRVALA

PURPOSE: To obtain the value of a fullword or halfword parameter.

ENTRY

CONDITIONS: Entry at PRVAL assumes:
Parameter number in Register 6.
NOWM - No. of currently active transaction.
NOWMAD - Address (T1) of currently active transaction.
Entry at PRVALA assumes:
Parameter number in Register 6.
Transaction number in Register 7.

EXIT

CONDITIONS: Both PRVAL and PRVALA exit with the specified parameter value in Register 6.

ROUTINE: CREAT

PURPOSE: To activate the next available transaction from the common pool of inactive transactions.

ENTRY

CONDITIONS: Number of bytes for transaction words (20) and parameters (2/ Halfword or 4/ Fullword) to be obtained from COMMON in Register 6.

EXIT

CONDITIONS: Area obtained from COMMON is zeroed.

ROUTINE: GETCOR

PURPOSE: To obtain a block of COMMON storage.

ENTRY

CONDITIONS: Number of bytes requested in Register 6.

EXIT

CONDITIONS: Starting address of block in Register 7. Actual number of bytes obtained in Register 6 (All core obtained is adjusted to fullword).

ROUTINE: FLOAT

PURPOSE: Convert a fixed-point number to floating-point.

ENTRY

CONDITIONS: Number to be converted in Register 8.

EXIT
 CONDITIONS: Converted number returns in Floating-point Register 2.

ROUTINE: UNFLOT
 PURPOSE: Convert a floating point number to fixed-point

ENTRY
 CONDITIONS: Number to be converted in Floating-point Register 2.

EXIT
 CONDITIONS: Integer portion returns in Register 8. Fraction portion returns in Register 9. Sign is carried by sign of Register 8.

ROUTINE: ERASE
 PURPOSE: To destroy active transaction, return it to inactive pool and restore all COMMON storage associated with the transaction.

ENTRY
 CONDITIONS: Transaction number in Register 9.

EXIT
 CONDITIONS: None

ROUTINE: FRECOR
 PURPOSE: To return a block of COMMON storage to the available state.

ENTRY
 CONDITIONS: Address of block in Register 8. Number of bytes in block in Register 9.

EXIT
 CONDITIONS: Core returned is zeroed at exit.

ROUTINE: DRAND
 PURPOSE: To generate a random number

ENTRY
 CONDITIONS: Random number index (1-8) in Register (6)

EXIT
 CONDITIONS: Random number between 0 and 999 in Register 7.

ROUTINE: DCOD
 PURPOSE: To evaluate a GPSS/360 Standard Numerical Attribute.

ENTRY
 CONDITIONS: SNA to be decoded in Register 7.

EXIT
 CONDITIONS: Value of SNA in Register 7.

The contents of the various GPSS/360 control words are obtainable in much the same way as the addresses previously mentioned. First the user must load the address of the first control word -

L 9,24(10) GET BASE FOR CONTROL WORDS

This address (in GR 9) may now be used as a base to obtain the desired control word. For instance, if the desired word had a displacement of four fullwords from the first control word, that value could be obtained with the following sequence of instructions.

L 9,24(10) Obtain base for control words
 L 7,16 (9) Obtain contents of fifth word
 The desired value would now be in Register 7.

The following is a partial list of the available control words and their displacement (in bytes) from the base address for control words (seventh word of passed table).

Symbolic Name	Displacement	Source
XACLOC	640	Address (T1) of transaction created by CREAT Routine
NUMXAC	644	Number of transaction created by CREAT routine
CLOCK	656	Current value of absolute clock
MANY	660	Number of terminations remaining
NOWMAD	688	Address (T1) of current transaction
NOWMCOM	692	Address of COMMON words for current transaction
NOWM*	730	Number of currently active transaction

The following words provide the starting address for core allocated to:

XAC	1000	Transactions
BLO	1004	Blocks
CHA	1008	User Chains
FAC	1012	Facilities
STO	1016	Storages
LOG	1020	Logic Switches
SAV	1024	Fullword Savevalues

*Halfword

VAR	1028	Variables
FUN	1032	Functions
QUE	1036	Queues
TAB	1040	Tables
MSF	1044	Fullword Matrix Saves
MSH	1048	Halfword Matrix Saves
HSAV	1052	Halfword Savevalues
GRP	1056	Groups
BVAR	1060	Boolean Variables

The following words indicate the available quantity of each entity.

XACNUM	1064	Transactions
BLOKNUM	1068	Blocks
CHANUM	1072	User Chains
FACNUM	1076	Facilities
STONUM	1080	Storages
LOGNUM	1084	Logic Switches
FSVNUM	1088	Fullword Savevalues
VARNUM	1092	Variables
FCNNUM	1096	Functions
QUENUM	1100	Queues
TBLNUM	1104	Tables
MSFULL	1108	Fullword Matrix Savevalues
MSHALF	1112	Halfword Matrix Savevalues
HSVNUM	1116	Halfword Savevalues
GRPNUM	1120	Groups
BVARNUM	1124	Boolean Variables

The method of computing the address of a particular entity (i. e. , Transaction 6, Storage 12, Queue 7, etc.) is quite simple and can be done in the following three steps:

1. Load the entity number into a register.
2. Multiply by the number of bytes/entity.
3. Add the base address for that entity to the product.

For example: A user wishes to obtain the contents of a particular savevalue location whose index number was specified by field B of the HELP block.

L	7,0(10)	Obtain field B value
SLA	7,2	Multiply by 4 (4 bytes/ savex)
L	9,24(10)	Obtain address of control words
L	8, 1024(9)	Obtain base addr. for savevalues
AR	7,8	ADD base to product above
L	6,0(7)	Register 6 contains contents of specified savevalue

The following is a list of the GPSS/360 entities along with the number of bytes required for each.

<u>Entity</u>	<u>Basic Core Allocation</u>
Transactions	16
Blocks	12
Facilities	28
Storages	40
Queues	32
Logic Switches	6
Tables	48
Functions	32
Variables	48
Savevalues (fullword)	4
(halfword)	2
User Chains	24
Groups	4
Boolean Variables	32
Matrix Savevalues (fullword)	24
(halfword)	24

The user is referred to the chapter of the User's Manual which discusses each entity type for a detailed description of the contents of the various fields.

GROUP ENTITY

In many GPSS models, transactions represent items which could be and sometimes are categorized by common attributes. For instance if transactions represented parts in an inventory model, the parts could be categorized by their weight, color, price, time to manufacture, length of time in stock, etc. A second example might be a manufacturing model in which items to be processed could be categorized by their priority, machines required for processing, time, etc. In these examples, the attributes (color, weight, price, etc.) are usually carried by the parameters of the transactions and can be changed by passing the transactions through as ASSIGN block. However, this provides only limited and controlled access to attributes of these items. In some situations it might be necessary to change or modify certain attributes of all transactions within a given category. For example: change the price of all partA items regardless of their current status in the model -- current, future, interrupt, user chains, etc.. Without some means of categorizing transactions, and subsequent communications between transactions within a given category, modeling changes of this type become extremely awkward and sometimes impossible.

In other models, the user may wish to develop lists of numeric quantities without reserving a series of SAVEVALUES for this purpose. These numeric quantities within a category could represent the facilities which exceed a given utilization, equipment whose failure rate exceeds a given value, the queue numbers whose contents exceed a certain limit, etc.

The GROUP entity and associated blocks provide the user with a means of categorizing transactions and/or other entities. The GROUP entity also provides a means of communicating and referencing attributes of transactions which are members of a given GROUP. A GROUP is basically a list of numbers. The interpretation and meaning of a GROUP is dependent on what elements constitute a GROUP, and how the user creates, manipulates, and removes members of a GROUP within the model.

A GROUP and reference to members of a GROUP is completely independent of the status of the members which make up the group. If transactions made up a group, they could all be referenced regardless if they were on the future, current, interrupt, or user chain. For example, certain parts may be categorized by weight and color. The combinations of weight and color could be represented as GROUPS. Within the GPSS model, subsequent transaction flow could be determined by the GROUP in which the transaction is a member. The blocks associated with the GROUP entity enable the user to change certain attribute values such as priority or parameter values of all transactions which make up a GROUP. This type of operation will take place regardless of where the transactions are in the system -- future, current, interrupt, or user chains. This provides a means for transactions within a given GROUP to communicate with other members of the GROUP.

As with other entities provided by GPSS/360, there are a given number of GROUPS associated with each of the three standard sets of GPSS/360 entities. If additional GROUPS are required, they can be obtained by means of REALLOCATION.

The GROUPS operate in either one of two possible modes -- transaction or numeric. The mode of operation for a particular GROUP is determined by the first reference to the GROUP, for example, passing a transaction through a JOIN block. Once the mode of operation has been established, any subsequent reference to the GROUP must be in the established mode. When

operating in the transaction mode, entries of the GROUP represent actual transaction numbers which are members of the GROUP. When operating in the numeric mode, the entries of the GROUP represent a list of numeric values. Again, the interpretation and meaning of the entries of a GROUP operating in either the transaction mode or numeric mode is dependent on the user and on how members of the GROUP are created, manipulated, and removed.

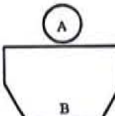
The SNA associated with the GROUP is G_j where G_j is the number of current members of GROUP j. The standard output of GPSS/360 includes GROUP information. The GROUP output information provides the GROUP number, the GROUP mode of operation (transaction or numeric), and the members of the GROUP. If the GROUP is operating in the transaction mode, the members listed will be numbers of the transactions which are currently members of the GROUP. The five blocks associated with the GROUP entity are:

JOIN
REMOVE
EXAMINE
SCAN
ALTER

The first three blocks (namely, JOIN, REMOVE, and EXAMINE) operate in either the transaction or the numeric mode. The SCAN and ALTER blocks operate only in the transaction mode.

JOIN Block

8	19 A	B
JOIN	GROUP No. Transaction and Numeric	Numeric Value Source. Numeric Mode Only.
	k, *n, SNAJ, SNA*j	k, *n SNAJ, SNA*j



The JOIN block is the means by which a transaction or numeric value is made a member of the GROUP. The JOIN block never refuses entry to a transaction.

Transaction Mode of Operation

If only the field A of the JOIN block is specified, the JOIN block references the GROUP in the transaction mode. That is, the entering transaction is made a

member of the GROUP specified by the field A of the JOIN block. If the GROUP specified by the field A is already operating in the numeric mode, an error will occur and the simulation will be terminated. A given transaction may be a member of any number of GROUPs, the only restriction being that the transaction must enter a JOIN block operating in the transaction mode for each GROUP in which the transaction is to become a member. For example:

8	19
JOIN	10

The number of the entering transaction becomes a member of GROUP 10. If this is the first reference to GROUP 10 by any transaction, the GROUP will be identified as operating in the transaction mode.

8	19
JOIN	X19

The number of the entering transaction becomes a member of the GROUP whose number is given by the contents of SAVEVALUE 19.

Numeric Mode of Operation

If the A and B fields of the JOIN block are specified, the JOIN block references the GROUP in the numeric mode. That is, the value of the SNA specified by the B field is evaluated and this value made a member of the GROUP specified by the A field. If this is the first reference to the GROUP, the GROUP is identified as operating in the numeric mode. If the SNA value specified by the B field is already a member of the GROUP specified by the A field, the value is not entered again as a member of the GROUP. That is, when a GROUP is operating in the numeric mode, any given numeric value will only appear once in the listing of the members of a given GROUP. For example:

8	19
JOIN	2, 50

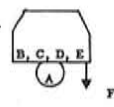
The numeric value 50 (B field) is made a member of GROUP 2.

8	19
JOIN	P3, V4

Variable 4 is evaluated and the numeric value made a member of the GROUP whose number is specified by the contents of parameter 3 of the entering transaction.

REMOVE Block

8	19	A	B	C	D	E	F
REMOVE	GROUP No.	COUNT No. of Members to be removed from GROUP	Numeric Value to be removed	Transaction Attributes for Comparison purposes	Comparison SNA	Alternate Exit	
		Xact Mode	Numeric Mode	Xact Mode	Xact Mode		
	k, *n	k, *n, SNAJ, SNA*J	k, *n, SNAJ, SNA*J	RR (Priority) Parameter No. k, *n, SNAJ, SNA*J	k, *n SNAJ, SNA*J	k, *n SNAJ, SNA*J	



The REMOVE block enables the user to remove transactions or numeric values from a GROUP. It provides the ability to remove more than a single transaction from a GROUP, and also provides a means of removing transactions from the GROUP based on the value of transaction attributes (priority or parameters).

Transaction Mode of Operation

The A field of the REMOVE block specifies the GROUP. This GROUP is searched to determine if the transaction which entered the REMOVE block is a member. The F field of the REMOVE block specifies an alternate block to which the entering transaction is sent if it is not a member of the GROUP specified by the A field. If the F field is blank, the entering transaction continues unconditionally to the next sequential block. If the entering transaction is a member of the GROUP specified by the A field, it is removed from this GROUP. For example:

8	19
REMOVE	1

If the entering transaction is a member of GROUP 1, it is removed from GROUP 1 and proceeds to the next sequential block. If it is

not a member of GROUP 1, it also proceeds to the next sequential block.

```

8          19
-----
REMOVE    5,,,,,NEXXT
  
```

If the entering transaction is a member of GROUP 5, it is removed from GROUP 5 and proceeds to the next sequential block. If it is not a member of GROUP 5, it proceeds to block NEXXT.

An alternate transaction mode of operation is provided by the REMOVE block whereby ALL or a specified number of transactions whose attributes satisfy certain conditions may be removed from the GROUP. This mode of operation is specified by entries in the B, D, and E fields of the REMOVE block. The B field specifies the SNA which determines the number of transactions to be removed from the GROUP. If the D and E fields are blank, the number of transactions specified by the B field are removed unconditionally from the GROUP. The D argument specifies the transaction attribute used for comparison purposes in determining if a transaction should be removed from the GROUP. This entry may be PR (priority) or a parameter number. Any SNA may be used to specify the parameter number. The E field specifies the SNA against which the transaction attribute will be compared. For example:

```

8          19
-----
REMOVE    1, ALL
  
```

ALL transactions which are members of GROUP 1 are removed unconditionally from GROUP 1.

```

8          19
-----
REMOVE    3, 10, , PR, 6
  
```

The first 10 transactions that are members of GROUP 3 and whose priority is equal to 6 are removed from GROUP 3.

```

8          19
-----
REMOVE    3, ALL, , 5, 100, NEXXT
  
```

Remove all transactions from GROUP 3 which have a value of 100 in parameter 5. If the GROUP has no members which meet this condition, send the entering transaction to block NEXT.

Numeric Mode of Operation

When using the REMOVE block in the numeric mode, only the A and C fields are required, and the F field is optional. The B, D, and E fields are not used. The GROUP specified by field A is examined for the value specified by the C field SNA. If the value is found, it is removed and the transaction proceeds to the next sequential block. If the value is not found, the F field is examined for an alternate exit. If an alternate exit is specified, the entering transaction proceeds to it. If an alternate exit is not specified, the entering transaction proceeds unconditionally to the next sequential block. For example:

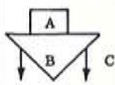
```

8          19
-----
REMOVE    5, , V3,,,NEXT
  
```

Remove the quantity j, the value of Variable 3 from GROUP 5. If this value is not found in GROUP 5, the entering transaction proceeds to block NEXT.

EXAMINE Block

8	19	A	B	C
EXAMINE	GROUP No.		Numeric Value Numeric Mode	Alternate Exit
	k,*n SNAJ,SNA*J		k,*n SNAJ,SNA*J	k,*n SNAJ,SNA*J



The EXAMINE block enables the user to select the path a transaction will take based on GROUP membership.

Transaction Mode of Operation

The A and C fields of the EXAMINE block must be specified when operating in the transaction mode. If the entering transaction is a member of the GROUP specified by the A field, it proceeds to the

next sequential block. If it is not a member of the GROUP, it proceeds to the block specified by the C field. For example:

```

8      19
EXAMINE 1, NEXT
  
```

If the entering transaction is a member of GROUP 1, it proceeds to the next sequential block. If it is not a member of GROUP 1, it proceeds to block NEXT.

Numeric Mode of Operation

The A, B, and C fields of the EXAMINE block must be specified when operating in the numeric mode. If the numeric quantity specified in the B field is a member of the GROUP specified by the A field, the entering transaction proceeds to the next sequential block. If it is not a member of the GROUP, the entering transaction proceeds to the block specified by the C field. For example:

```

8      19
EXAMINE 5, X15, ABCD
  
```

If the numeric value given by the contents of SAVEVALUE 15 is a member of GROUP 5, the entering transaction proceeds to the next sequential block. If the value is not a member of GROUP 5, the entering transaction proceeds to block ABCD.

The ability to obtain and modify attributes of transactions within a given GROUP is provided by the SCAN and ALTER blocks. The operations specified by these blocks takes place regardless of the status of members of the GROUP included on current, future, interrupt, user chains, etc.

The SCAN and ALTER blocks operate only in the transaction mode.

SCAN Block

The SCAN block provides the following capabilities:

1. Determine whether a transaction with certain attribute values is a member of a given GROUP.

B	A	B	C	D	E	F
SCAN	GROUP No.	Transactions Attribute for Comparison	Comparison Value for B Argument	Attribute to be Obtained if Match is made	Parameter Number in which to place D Argument Value	Alternate Exit
	k, *n SNAj, SNA*j	PR (Priority) Parameter k, *n SNAj SNA*j	k, *n SNAj SNA*j	PR (Priority) k, *n SNAj, SNA*j	k, *n SNAj SNA*j	k, *n SNAj SNA*j



2. Obtain attribute values of a member of the GROUP which meets certain specific conditions.

3. Modify subsequent transaction flow if a member which satisfies specific conditions is not found.

Operation if A, B, C, and F Fields Are Specified

The A field specifies the GROUP whose members are to be examined. The B field specifies the transaction attribute which is to be examined. This may be PR (priority) or a parameter. (NOTE: if Pn or *n, etc., is coded in the B field, the value of parameter n is interpreted as the desired parameter number.)

The C field specifies the SNA against which the value of the transaction attribute will be compared.

The F field specifies an alternate exit for the entering transaction if no member is found which satisfies the specified conditions.

If the F field is blank, the entering transaction unconditionally proceeds to the next sequential block. For example:

```

8      19
SCAN 1, PR, 6, , BCDF
  
```

GROUP number 1 (A field) is examined to determine if a transaction with a priority (B field) of 6 (C field) is a member. The first transaction found which has a priority of 6 causes the entering transaction to proceed immediately to the next sequential block. If all members of the GROUP are examined and the condition is not satisfied, the entering transaction proceeds to block BCDF.

Operation if A, B, C, D, E, and F Fields Are Specified

The B field specifies the attribute of the members of the A field GROUP which are compared against the value of the C field SNA. If PR is coded in the B field, the priority of each member of the GROUP is compared against the C field value. If any SNA is coded in the B field, it is evaluated and interpreted as a parameter number of the members. (NOTE: If Pn or *n is coded in the B field, the value of parameter n is interpreted as the desired parameter number.)

If a match is found between the B field attribute and the C field SNA, the D field attribute of the matching transaction is obtained and placed in the parameter of the entering transaction, specified by the E field. The D field is interpreted exactly as was the B field, i.e., PR = priority, any SNA = parameter no. The entering transaction then proceeds to the next sequential block

If no match is found, the entering transaction proceeds to the alternate block specified in the F field. If no alternate block is specified, the entering transaction proceeds to the next sequential block. For example:

```

8          19
-----
SCAN      1, PR, 25, 5, 1
  
```

Scan GROUP 1 for the first transaction with a priority of 25. If one is found, obtain the value of parameter 5 and place it in parameter 1 of the entering transaction. If no match is found, proceed to next block.

```

8          19
-----
SCAN      4, 10, X3, PR, 6, NEXT
  
```

Scan GROUP 4 for the first transaction with parameter 10 equal to the value of X3. If one is found, obtain its priority and place it in parameter 6 of the entering transaction. If none is found, send entering transaction to block NEXT.

ALTER Block

The ALTER block enables the user to modify attributes of transactions which are members of a given GROUP. The operation of the ALTER block provides.

1. Unconditionally modify a given attribute of ALL members of a GROUP.
2. Modify a given attribute of a number of members of a GROUP.

3. Modify a given attribute if some other attribute meets a specified condition.

The A field specifies the GROUP number. The B field specifies ALL or a COUNT which can be any SNA. The C field is the attribute of transactions in the GROUP which will be altered. PR in the B field means the priority and any SNA is interpreted as a parameter number. The D field SNA is the value to replace the transaction attribute specified by the C field. The E, F, and G fields are all optional. The E field specifies a transaction attribute (PR or parameter no.) to be matched with the F field SNA before the altering will take place. The G field indicates an alternate exit for the entering transaction if the A field GROUP has no members or if a matching condition is specified in fields E and F and no transaction is found which satisfies the condition. For example:

```

8          19
-----
ALTER     1, ALL, PR, 100
  
```

Alter the priority of all transactions which are members of GROUP 1. All transactions will be assigned a priority value of 100.

```

8          19
-----
ALTER     2, 10, 35, X10, ,, NEXT
  
```

Alter the value of parameter 35 to the value of savevalue 10 for the first 10 transactions in GROUP 2. If there are less than 10 members in GROUP 2, alter all. If there are no members in GROUP 2 send the entering transaction to block NEXT.

```

8          19
-----
ALTER     3, ALL, PR, 0, 1, 10, NEXT
  
```

Alter the priority to 0 of all transactions in GROUP 3 which have a 10 in parameter 1. If no transactions which are members of GROUP 3 have a 10 in parameter 1, send the entering transactions to block NEXT.

To account for possible switching of priorities, the current event chain scan should be reinitiated after an ALTER block which alters priorities.

If priorities are altered and the transaction is in the current event chain, it is removed from its former priority class and placed at the end of the new priority class.

8	19	A	B	C	D	E	F	G
ALTER	GROUP No.	COUNT	Member Attribute to be Altered	Value to replace Attribute	Matching Transaction Attribute	Matching SNA	Alternate Exit	
	k, *n SNAJ, SNA*J	ALL k, *n, SNAJ, SNA*J	PR (Priority) Parameter k, *n SNAJ, SNA*J	k, *n SNAJ, SNA*J	PR (Priority) Parameter k, *n SNAJ, SNA*J	k, *n SNAJ, SNA*J	k, *n SNAJ, SNA*J	

CHAPTER 8: SAVEVALUE ENTITIES

GENERAL PROPERTIES OF SAVEVALUE ENTITIES

Savevalue entities are used in simulation models to retain the values of other Standard Numerical Attributes for future reference. Each fullword savevalue may contain a signed 31-bit value of a Standard Numerical Attributes and each halfword savevalue may contain a signed 15-bit value of a Standard Numerical Attribute. The standard GPSS/360 program for a 128K machine has core allocated for 400 fullword savevalues and 200 halfword savevalues.

STANDARD NUMERICAL ATTRIBUTES

There are four Standard Numerical Attributes associated with savevalues:

Xj = current value saved in Fullword Savevalue j

XHj = current value saved in Halfword Savevalue j

MXj (m, n) = current value saved in mth row and nth column of Fullword Matrix Savevalue j

MHj (m, n) = current value saved in mth row and nth column of Halfword Matrix Savevalue j.

These Standard Numerical Attributes may be indirectly addressed as follows:

X*n = current value saved in savevalue whose location is given by value of Parameter n

and similarly for XH*n, MX*n (m, n), and MH*n (m, n).

STANDARD LOGICAL ATTRIBUTES

There are no Standard Logical Attributes associated with savevalue entities. Consequently, the status of a savevalue cannot be referenced in a GATE block to control the flow of transactions. However, it is quite simple to use the value of the Standard Numerical Attribute Xj as one of the arguments in a TEST block. For example:

2	LOC	7	8	OPERATION	19	A	B	C
				TEST	E	Xj	K0	Alternate block may or may not be coded in C-field
				TEST	NE	Xj	K0	
				TEST	L	Xj	K0	
				TEST	G	Xj	K0	

The above four TEST blocks control the flow of transactions, depending on whether the following conditions regarding the savevalue are true or false.

1. Xj is zero
2. Xj is nonzero
3. Xj is negative, i. e., less than zero
4. Xj is positive, i. e., greater than zero

The TEST block can operate in an unconditional or conditional entry mode depending on whether an alternate block number is coded in field C.

If Xj, XHj, MXj (m, n), or MHj (m, n) is used as single elements within a Boolean variable statement, it is interpreted as one if nonzero and zero if zero.

SAVEVALUE BLOCK

2	LOC	7	8	OPERATION	19	A	B	C
				SAVEVALUE	SNAj, SNA*n (blank)	k, *n	SNAj, SNA*n, k, *n	SAVEVALUE A, B, C, H

The SAVEVALUE block serves the function of storing the value of any of the Standard Numerical Attributes for further reference. A SAVEVALUE block never refuses entry to a transaction. Transactions proceed to the next sequential block following the SAVEVALUE block. The parameters of the entering transaction may be saved, if desired. References to the stored quantity can subsequently be made by the Standard Numerical Attributes Xj, XHj, MX, (k, l), or MHj (k, l). The values of all the savevalue quantities are initially zero. The field B argument is used to specify the Standard Numerical Attribute to be stored. The value of the field A argument specifies the savevalue location (j) in which field B is to be saved.

Field C is used to denote halfword savevalue or fullword savevalue. If a halfword savevalue is desired, the character 'H' is placed in field C. If

there is no entry in field C, fullword savevalue is assumed. Halfword savevalues are provided in GPSS/360 to optimize the use of core when the user is dealing with numbers of magnitude less than $2^{15}-1$.

NOTE: If any overflow occurs during computation of the integer to be saved, only the low order 15 bits are retained for halfword savevalues and the following warning message will be printed:

WARNING EXECUTION ERROR NUMBER 851.
BLOCK NUMBER XXXX. CLOCK YYYY.
SIMULATION CONTINUES.

where XXXX = SAVEVALUE block at which
the error occurred.

YYYY = Clock time at which error
occurred.

This message is only printed the first time an overflow occurs as a result of a halfword SAVEVALUE operation and the simulation continues.

Replacement, Addition, and Subtraction in Savevalue Location

The field A Standard Numerical Attribute argument can be followed by three possible characters (blank, +, or -) which define modes of operation.

Replacement Mode

Replacement mode is indicated by "blank." The value of field B Standard Numerical Attribute replaces the current value in the field A savevalue location.

Example:

SAVEVALUE 10, V10

The value of Arithmetic Variable 10 replaces the current value in Savevalue location 10.

Addition Mode

Addition mode is indicated by "+". The value of the field B Standard Numerical Attribute is added to the current value in the field A savevalue location.

Example:

SAVEVALUE *2+, X*5

The value currently stored in the savevalue location, whose index number is given by the value of Parameter 5 of the transaction currently being processed, is added to the current value in the savevalue location whose index number is given by the value of Parameter 2. The value of X*5 remains unchanged.

Subtraction Mode

Subtraction mode is indicated by "-." The value of the field B Standard Numerical Attribute is subtracted from the current value in the field A savevalue location.

Example:

SAVEVALUE 1-, K4, H

The constant K4 is subtracted from the current value in Halfword Savevalue location 1.

Matrix Savevalues

Matrix Savevalues provide the user with the ability to associate additional attributes with GPSS/360 entities such as facilities, storages, logic switches, user chains, etc. For example, if a facility represented a machine, certain characteristics (such as number of failures, number of different job types processed, time machine was last used, job type machine is currently processing, time machine is scheduled for maintenance, etc.) might be of interest to the user.

The user may retain such information in savevalues arranged in an M x N Matrix, where M is the number of rows and N is the number of columns. The user may specify the number of rows and columns for the matrix savevalue as well as the capacity (i. e. , fullword or halfword) by means of a MATRIX definition card.

MATRIX Definition Card

The MATRIX definition card defines the dimensions of a Matrix Savevalue in the following manner:

1. Matrix Savevalue number in columns 2-6.
2. X or H in field A specify fullword or halfword respectively.
3. Field B defines the number of rows.
4. Field C defines the number of columns.

Examples:

- 1 MATRIX X, 5, 5 A 5x5, fullword matrix
- 2 MATRIX H, 3, 4 A 3x4, halfword matrix

MSAVEVALUE Block

The MSAVEVALUE block is used to enter values into a Matrix Savevalue in much the same way a SAVEVALUE block is used to enter values into a savevalue. The MSAVEVALUE block never refuses entry to a transaction and all transactions proceed to the next sequential block.

Field A specifies the Matrix Savevalue location (j) in which the field D quantity is to be saved. Fields B and C specify the row number (m) and the column number (n) respectively. The field D argument specifies the quantity to be stored in the particular row and column of the specified Matrix Savevalue. Any of the four fields may be specified as Standard Numerical Attributes. If a Halfword Matrix Savevalue is desired, the character 'H' is placed in field E. If there is no entry in field E, a Fullword Matrix Savevalue is assumed. Halfword Matrix Savevalues in GPSS/360 are used to optimize core when the user is dealing with numbers of magnitude less than $2^{15}-1$.

If any overflow occurs during computation of the integer to be saved, only the low-order 15 bits are retained for Halfword Matrix Savevalues and the following warning message will be printed:
 WARNING EXECUTION ERROR NUMBER 852.
 BLOCK NUMBER XXXX. CLOCK YYYY.
 SIMULATION CONTINUES.

where XXXX = MSAVEVALUE Block at which the error occurred.

YYYY = Clock time at which the error occurred.

This message is printed only the first time an overflow occurs as a result of a Halfword Matrix Savevalue operation and the simulation continues.

TABLE 15: CORE ALLOCATION FOR FULLWORD (HALFWORD) MATRIX SAVEVALUE ENTITIES

SYMBOL	LENGTH	SOURCE
MS1 (MSH1)	4 bytes	Address of COMMON area where Matrix Points are stored.
MS2 (MSH2)	4 bytes	First halfword = Number of rows Second halfword = Number of columns.
MS3(MSH3)	4 bytes	Used during evaluation of Matrix to store column number while decoding row number SNA.
MS4 (MSH4)	4 bytes	Used during evaluation to save the return address to the DCOD routine. Also serves as a cyclic indicator.
MS5 (MSH5)	4 bytes	Used during evaluation to save the base for Variable or Function words when the Matrix evaluation routine is called from those evaluation routines.
MS6 (MSH6)	4 bytes	Used during evaluation to save the base for the calling routine.

TABLE 16: FORMAT for MATRIX SAVEVALUE POINTS IN COMMON

Each defined Matrix Savevalue has associated with it (MxN) fullwords (halfwords) where M = Number of rows and N = Number of columns. These words are obtained from the COMMON area and the address stored in Matrix Savevalue reference word MS1 or MSH1 (see Table 15).

Each of these words (halfwords) contain the value of a particular row and column of the Matrix as outlined below for a Matrix Savevalue with 4 rows and 4 columns.

2 LOC	7 8 OPERATION	MATRIX SAVEVALUE LOCATION	ROW NUMBER	COLUMN NUMBER	STANDART NUMERICAL ATTRIBUTE TO BE SAVED	SPECIFIED IF HALFWORD NATRIX SAVEVALUE BEING REFERENCED	MSAVEVALUE
		19 A	B	C	D	E	A, B, C, D, E
	MSAVEVALUE	SNAj SNA*n, (blank) K, *n	SNAj, SNA*n, K, *n	SNAj, SNA*n, K, *n	SNAj, SAN*j K, *n	H or blank	

TABLE 16: (Cont.)

R1-C1	R1-C2	R1-C3	R1-C4
R2-C1	R2-C2	R2-C3	R2-C4
R3-C1	R3-C2	R3-C3	R3-C4
R4-C1	R4-C2	R4-C3	R4-C4

The formula for finding the displacement (from address in MS1) for a particular row (r) and column (c) of an MxN Matrix is as follows:

$$[c(r-1)] + (c-1)$$

this value is then multiplied by either 2 (halfword) or 4 (fullword) to obtain the displacement.

Examples:

MSAVEVALUE 3, 1, 1, V10

The value of Variable 10 will be saved in row 1, column 1 of Matrix Savevalue number 3.

MSAVEVALUE X1, X2, X3, FN4, H

The values in Savevalues 1, 2, and 3 will determine the Halfword Matrix number, row, and column, respectively, where the computed value of Function 4 will be saved.

The MSAVEVALUE block may also operate in the addition or subtraction modes just as the SAVEVALUE block. This is done by coding a + or - following the field A Matrix number.

MSAVEVALUE 2+, 4, 5, Q3

The current contents of Queue 3 will be added to the current value of row 4, column 5 of Matrix Savevalue number 2.

Matrix Savevalues as Standard Numerical Attributes. Matrix Savevalues may be used in any block field where SNAj is specified. The mnemonic for Matrix Savevalues is MX for fullword and MH for halfword followed by the entity number. This must be followed by the row and column numbers in parentheses.

Examples:

ASSIGN 2, MX1 (3, 4)

The value in row 3, column 4 of fullword Matrix Savevalue number 1 will be ASSIGNED to

Parameter 2 of the entering transaction.

SEIZE MH2 (4, 4)

The entering transaction will SEIZE the facility whose number is in row 4, column 4 of halfword Matrix Savevalue number 2.

Redefinition of Matrices

If a matrix is redefined during a simulation job, the number of elements in the new matrix is compared with that of old matrix. When there are more elements in the new matrix, core for the old matrix elements is freed and core for new matrix elements is obtained from available COMMON storage. When there are less elements in the new Matrix, the core which was used to define the old matrix is reused and any unneeded core is returned to the COMMON pool.

INITIAL CARD

SAVEVALUES

2	LOC	7	8	OPERATION	SAVEVALUE LOCATION	VALUE TO BE STORED IN SAVEVALUE LOCATION	
					19	A	B
				INITIAL	Xj	aaaaaa	aa
					XHj	-bbbb	bb

2	LOC	7	8	OPERATION	MATRIX SAVEVALUE LOCATION	VALUE TO BE STORED
					19	
				INITIAL	MXj (k, l), aaaaaaa	
					MHj (k, l), -bbbb	

Where: j = Matrix Savevalue number
 k = Row number
 l = Column number

The INITIAL input card permits the initialization of the value in a Savevalue location. The INITIAL card avoids the wasteful use of SAVEVALUE blocks to load constants into a model. INITIAL cards can also be read in during a simulation run. The B, C field value will replace the current value in field A Savevalue location.

For nonMatrix Savevalues the character X or XH must appear with the SAVEVALUE number in

field A. For Matrix Savevalues, MX or MH must be specified. Blanks are not allowed between any of the identification characters; e. g. , Xb88 or X8b8 are not allowed.

The value to be entered in fullword Savevalue locations may be up to ten characters in length. No blanks are allowed between characters of the value to be entered. Negative values may be specified by preceding the number with a minus (-) sign. If any of the above conditions are violated, a comment, ERROR IN ABOVE CARD, will follow the listing of the card in the printout. When an error occurs, the initialization of the Savevalue specified by the card will be deleted.

2	LOC	7	8	OPERATION	19	A	B	C
				INITIAL		X88	120000	0

The value 1200000 will be stored in Fullword Savevalue location 88.

2	LOC	8	OPERATION	19
			INITIAL	MH3 (2, 4), -33

The value -33 will be stored in row 2, column 4 of Halfword Matrix Savevalue location 3.

Since the initialized values may be up to ten digits long, actually in the range $\pm(2^{31}-1)$, the INITIAL card may be used to circumvent the six-digit limit of the usual GPSS/360 card field. For instance, the following ADVANCE block might be used in conjunction with the Savevalue initialized above:

2	LOC	7	8	OPERATION	19	A	B
100				ADVANCE		X88	

ADVANCE block 100 would have a constant delay time of 1200000 clock units.

Assembly Program Coding of INITIAL Card

Since fields B and C are combined, a comma is not required to separate columns 24 and 25. The

following coding is incorrect:

2	LOC	7	8	OPERATION	19	A, B, C
				INITIAL		X88,1200000

The following assembly program coding should be used:

2	LOC	7	8	OPERATION	19	A, B, C
				INITIAL		X88,1200000

Initial Card - Extended Format

In addition to the format just described, GPSS/360 allows the user to use the following extended format in the initial card:

```

8          19
INITIAL  X1, 40/X3, 50/XH1-XH8, 100
INITIAL  MX3(4, 5), -7/MH1-MH2(2-3, 1-2) -20
  
```

If the elements of Savevalues to be initialized are located in a contiguous area, these may be initialized all at once. The first initial card assigns 40 and 50 to Fullword Savevalues 1 and 3 respectively and 100 to Halfword Savevalues 1, 2, 3, 4, 5, 6, 7, and 8. The second initial card assigns -7 to the fourth row and fifth column element of the Fullword Matrix Savevalue 3. It also sets up the following Halfword Matrix Savevalues.

		MH1			MH2		
row \ col	1	2	3	row \ col	1	2	3
1				1			
2	-20	-20		2	-20	-20	
3	-20	-20		3	-20	-20	

In GPSS/360 the user also has the ability to preset logic switches with the use of initial card.

```

8          19
INITIAL  LS1/LS4-LS5
  
```

Here Logic Switches 1, 4, and 5 are set.

The user may initialize logic switches, Matrix Savevalues, and regular Savevalues with one initial card. Whenever multiple initialization is used by

the use of a dash between the lower index number and the upper index number, the upper index number should never be equal to or less than the lower index number. The following initial cards are illegal and appropriate error messages are described below.

```

8          19
INITIAL   MH1-MX2(1, 2), 3
INITIAL   MX2(1-1, 2), 4
INITIAL   X7-X3, 5
  
```

The first card has a mixed SNA, MH and MX. The second card has an illegal row number specification. The third card has the lower index number and the upper index number reversed.

Since GPSS/360 words are four bytes long, the maximum number of characters which can be assigned to Fullword Savevalues and Fullword Matrix Savevalues and Fullword Matrix Savevalues is ten digits and a sign; specifically, $2^{31}-1$ is the maximum.

For Halfword Savevalues and Halfword Matrix Savevalues the maximum is $2^{15}-1$. If anything greater than this is to be stored, only the low-order 15 bits will be retained and warning message 852 will be printed. Also, the user must have defined the matrix before it can be referenced in INITIAL Card.

EFFECT OF RESET, CLEAR AND JOB CARDS

All Savevalue and Matrix Savevalue locations are initially zero. Neither Savevalues nor Matrix Savevalues are changed by a RESET card. All Matrix Savevalues are set to zero by a CLEAR card and all Savevalues will be zeroed unless the user specifies otherwise by means of the selective clear feature explained in Chapter 15.

STANDARD SAVEVALUE STATISTICAL OUTPUT

The contents of all nonzero Savevalue locations are printed out in the following format as part of the standard statistical output at the end of each run, or after an execution error:

CONTENTS OF FULLWORD SAVEVALUES (NONZERO)

```

SAVE, LOC  VALUE LOC  VALUE LOC  VALUE
      1      1   26    10   35   60401
      101     73  539     6
  
```

CONTENTS OF HALFWORD SAVEVALUES (NONZERO)

```

SAVEX, LOC  VALUE LOC  VALUE LOC  VALUE
      1      20    2    35    10    41
  
```

CONTENTS OF FULLWORD MATRIX 1

```

COLUMN  1      2      3      4      5
ROW     1  10    0      1      0      2
        2  20    0      2      0      2
        3   0    0      3      0      2
        4   5    0      4      0      3
        5   1    0      5      0      3
        6   2    0      6      0      3
  
```

CONTENTS OF HALFWORD MATRIX 7

```

COLUMN  1      2      3
ROW     1   4      4      5
        2   5      4      5
        3   6      4      5
        4   7      6      6
  
```

PRINT BLOCK OUTPUT

The contents of Savevalue locations can be printed out dynamically during a simulation run by passing transactions through a PRINT block of the following type:

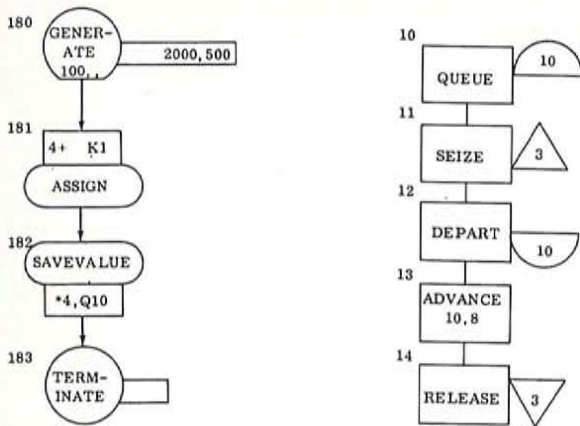
2	LOC	7	8 OPERATION	19 A	B	C	
			PRINT	k ₁	k _u	X or blank	(Fullword Savevalue)
			PRINT	k ₁	k _u	XH	(Halfword Savevalue)
			PRINT	k ₁	k _u	MX	(Fullword Matrix Savevalue)
			PRINT	k ₁	k _u	MH	(Halfword Matrix Savevalue)

The contents of all nonzero Savevalue locations are printed out from the field A lower limit k₁, up through the field B upper limit k_u. Field C of the PRINT block can be X, XH, MX or MH. If field C is blank, X is assumed.

EXAMPLES OF SAVEVALUE ENTITIES AND SAVEVALUE BLOCKS

Example 1:

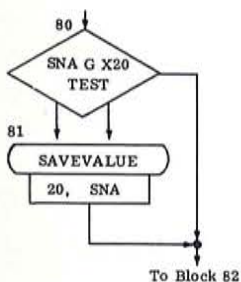
The following block diagram illustrates how a chronological graph of a queue length may be obtained. The length of Queue 10 is sampled



every 100 clock units, and placed in Savevalue location *4. Five hundred samples will be taken, and by using indirect addressing, a separate Savevalue location is used for each sample. Locations 1 through 500 are used, with the first sample taken at clock time 2000. The contents of the Savevalue locations are printed at the end of the simulation run, with the exception of those locations which contain zero.

Example 2:

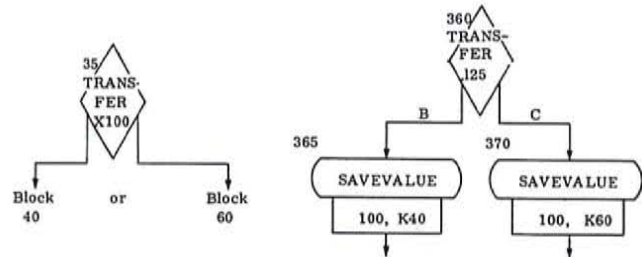
The following block diagram shows how the maximum value of a Standard Numerical Attribute may be measured. SAVEVALUE block 81 will be entered whenever a new maximum is attained. Savevalue location 20 is used to record the maximum value, which will be printed at the end of the simulation run.



Example 3:

Savevalues are frequently useful in performing block diagram logic. In the following example, the transaction which exits from TRANSFER block 360 modifies the path to be followed by the

transactions that enter block 35. TRANSFER block 35 obtains the value in Savevalue location 100, and uses that value as the next block selection for transactions entering the block.



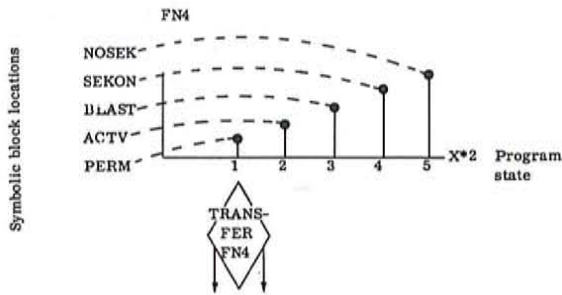
Example 4:

A program in a data processing system, such as an airlines system, can be in one of five resident states. These programs can be shared by several transactions. The current state of the program can be stored in a Savevalue location which is associated with each program. A change in the state of a program can be simulated by simply changing the number in the Savevalue location with a SAVEVALUE block. The numbers 1-5 could be used to indicate the following states.

<u>SAVEVALUE</u>	<u>PROGRAM STATE</u>
1	Program is permanently in core
2	Program is temporarily in core, and is being used by one or more transactions
3	Program is temporarily in core and is on a blast list because it is not being used by any transaction
4	Program is out of core, but a read into core has been initiated
5	Program is out of core, and no read into core has been initiated as yet

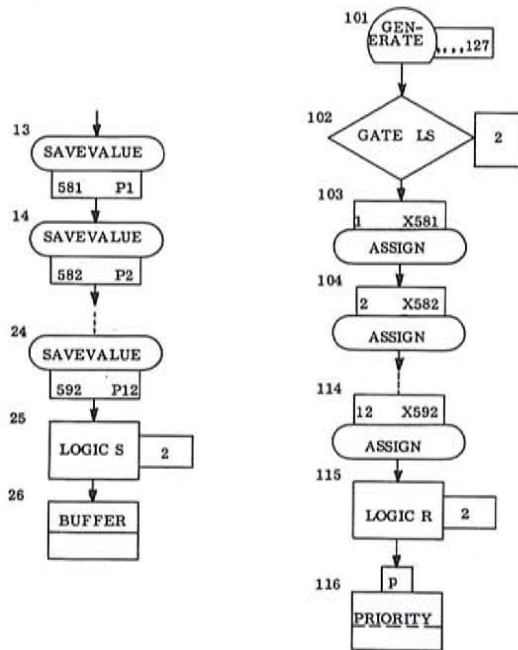
As transactions transfer from one program to another, the resulting operations will depend on the state of the program being transferred into. Assuming that the new program number is carried in Transaction Parameter 2, the

following List Function (Ln) can be used to route transactions to the appropriate part of a simulation model:



Example 5:

Savevalue locations can be used to transfer parameter values from one transaction to another. As an extreme example, consider the following group of blocks which allows one transaction to create another transaction which has the same parameter values but is not a member of the same assembly set (as would happen in a SPLIT block).



The original transaction saves Parameters 1-12 in Savevalue locations 581-592. It puts Logic Switch 2 in a set condition, whereupon the Priority 127 transaction (which has been blocked in GENERATE block 101 in an inactive delayed

status) is activated. The original transaction then enters BUFFER block 26 which transfers the overall GPSS/360 scan back to the start of the current events chain.

The GPSS/360 scan immediately encounters the active Priority 127 GENERATE block transaction. This transaction succeeds in entering GATE LS block 102. Before it leaves the GENERATE block, it creates an incipient successor transaction at the GENERATE block. The new transaction, which belongs to its own independent assembly set, then proceeds to ASSIGN to its Parameters 1-12 the value in Savevalue locations 581-592. The new transaction then puts Logic Switch 2 in a reset condition. Its normal priority is then assigned in PRIORITY block 116.

The overall GPSS/360 scan next encounters the incipient successor transaction as the last one in the Priority 127 class of the current events chain. This transaction attempts to enter GATE LS block 102, finds that Logic Switch 2 is in a reset condition, and hence is deactivated. The overall GPSS/360 scan finally comes back to the original transaction in BUFFER block 26 and attempts to move it into the next sequential block.

Example 6: Matrix Savevalue

An automobile dealer has set up the list of available cars together with their descriptions and prices. They are listed in the order of their arrival to the dealer.

The above data can be saved in the Halfword Matrix 1 defined as follows:

1 MATRIX H, 100, 8

Data Content of Inventory Matrix

ROW	COLUMN							
	1 Year	2 Cylinder	3 Type	4 Door	5 Automatic Transmission	6 Power Brake	7 Radio & Heater	8 Price
Car 1	1957	6	1=Ford	2	1=Yes	0	11=Yes	450
2	1961	8	2=Chevy	4	0=No	1	11	800
3	1941	4	1=Ford	2	0	01=Heater		500
4	1963	6	3=Ply- mouth	4	1	10=Radio only		1100
.								
.								
.								
.								
.								
100 (max)								

Specification for a required car are transferred from SAVEVALUE locations to Matrix Savevalue 2 by the Block sequence 8-11 shown in Figure 27. To determine whether a car with the required specifications is available, Block sequence 20-26 is executed. When a match is found, the price (column 8) of the car which meets the required specifications will be stored in Halfword Savevalue 2.

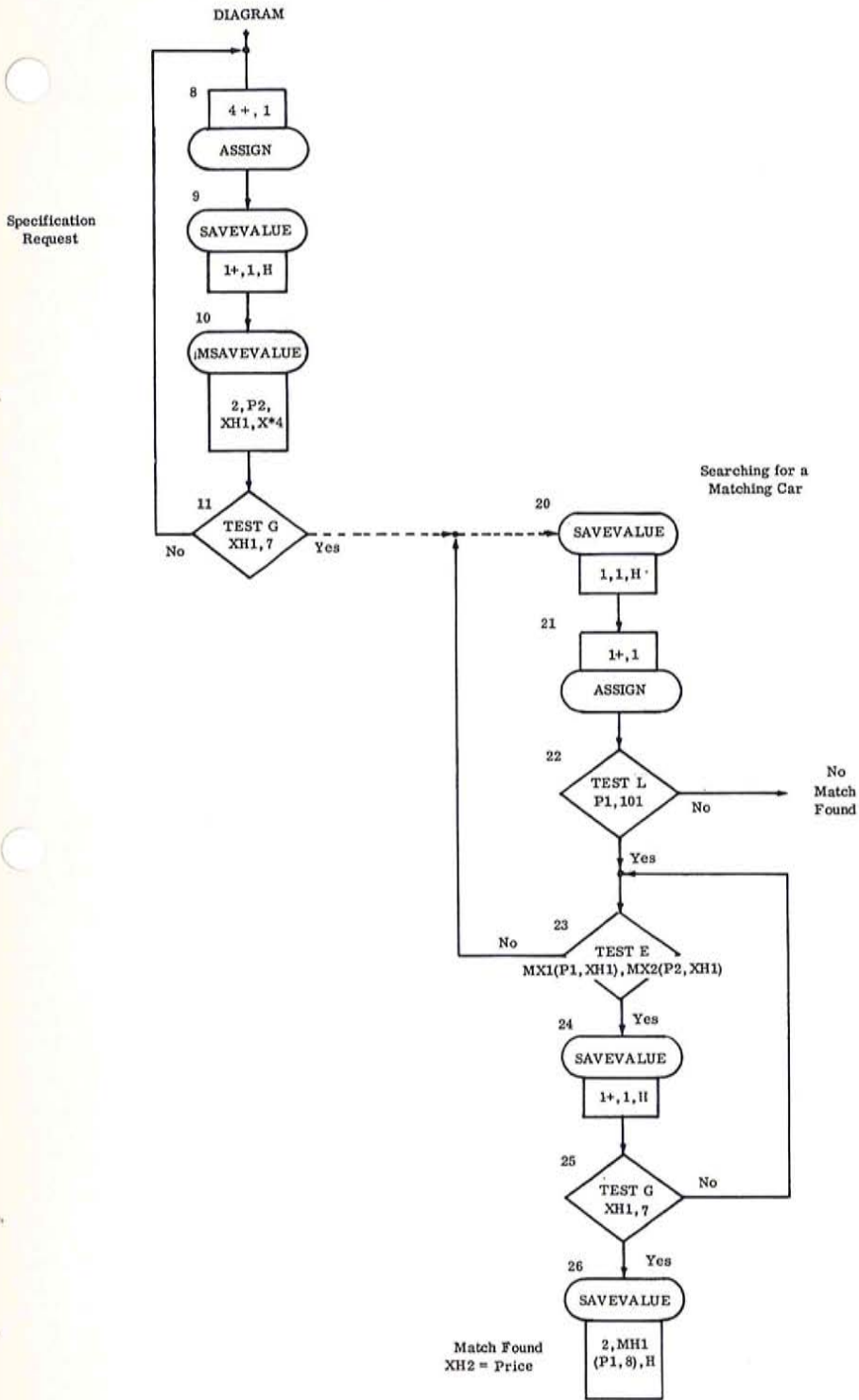


Figure 27.

CHAPTER 9: LOGIC SWITCH ENTITIES

GENERAL NATURE OF LOGIC SWITCH ENTITIES

Logic Switch entities are used in simulation models to represent binary states which may be either of a logical or a real physical nature. GATE LR and GATE LS blocks control the flow of a transaction as a function of the states of Logic Switches. The Standard GPSS/360 program for a 128k machine has core allocated for 400 Logic Switches numbered 1, 2, 3, ... 400. Execution Error 501 will occur if a Logic Switch number greater than the quantity allocated is referenced. Table 17 describes the attributes which are stored in three halfwords (L1, L2, and L3) required for each Logic Switch.

Each Logic Switch can be in one of two states, indicated by the L1 halfword:

1. Reset condition - L1 halfword is zero; i.e., the switch is off.
2. Set condition - L1 halfword is not zero; i.e., the switch is on.

The set condition is comparable to the on position of a switch, and the reset condition is comparable to the off position. All Logic Switches are initially in the reset or off condition.

STANDARD NUMERICAL ATTRIBUTES

There are no Standard Numerical Attributes associated with Logic Switch entities.

TABLE 17: S/360 CORE ALLOCATION FOR LOGIC SWITCH ENTITIES

<u>Symbolic Core Location</u>	<u>Size</u>	<u>Quantity</u>	<u>Source</u>
L1	Halfword	Status of Logic Switch	Zero if switch is reset; nonzero if switch is set. Status is altered by the Logic block
L2	Halfword	Delay Chain origin for the Logic Switch to be set	Each time a transaction at a GATE LS block fails to advance because the Logic Switch is reset, the number of that transaction is put in this field (unless the transaction is in a TRANSFER block with a BOTH or ALL selection mode). The previous entry in the field is placed in the first halfword of T1 of the current transaction, thus forming a pushdown delay chain of transactions waiting for the Logic Switch to be set by a LOGIC S block.
L3	Halfword	Delay Chain origin for the transactions waiting for the Logic Switch to be reset	This pushdown delay chain is formed in the above manner, and is activated whenever the Logic Switch is reset by a LOGIC R block.

STANDARD LOGICAL ATTRIBUTES

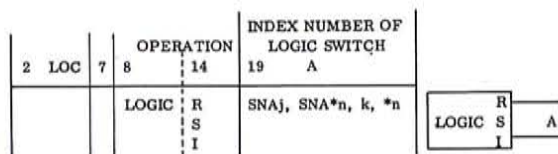
Each Logic Switch entity has two Standard Logical Attributes which, in turn, have the two values: true and false.

1. LRj is true if Logic Switch j is in a reset condition; it is false if in a set condition.

2. LSj is true if Logic Switch j is in a set condition; it is false if in a reset condition.

These Standard Logical Attributes are used in GATE LR and GATE LS blocks, as described below, to control the flow of transactions. Generally, these GATE LR and GATE LS blocks operate in a conditional entry mode; i. e., no field B alternate block is specified so the transactions cannot enter the GATE block unless the specified attribute LRj or LSj is true. Logic Switches can thereby simulate unit capacity physical equipment with conditional entry GATE blocks.

LOGIC BLOCK



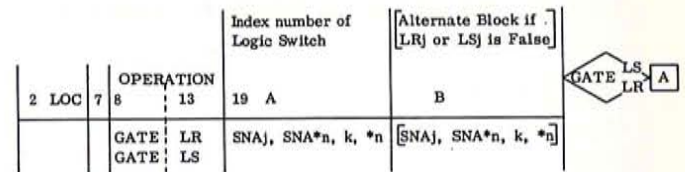
The LOGIC R, LOGIC S, and LOGICI blocks establish logical conditions (reset, set), which may be tested elsewhere in a model. A LOGIC block thus differs from the facility blocks (SEIZE, PREEMPT), which can deny entry to transactions. Transactions proceed to the next sequential block following the LOGIC block.

When a transaction enters a LOGIC block the condition of the Logic Switch j, whose number is specified by the value of the field A argument, may be altered in one of the following three ways, depending on the auxiliary field mnemonic:

- S The specified Logic Switch will be placed in a set condition.
- R The specified Logic Switch will be placed in a reset condition.
- I The specified Logic Switch will be inverted; that is, if it was set just before the transaction entered the LOGIC block, it will be placed in the reset state. If the Logic Switch was reset, it will be placed in the set state by the transaction.

The mnemonics S, R, or I must be coded in the auxiliary field (column 14). Observe that a Logic Switch may already be in the state specified by the auxiliary field mnemonic. In this case, nothing will be changed by LOGIC block.

GATE LR AND GATE LS BLOCKS



If a field B alternate block is specified, the GATE LR or GATE LS block operates in an unconditional entry mode; i. e., transactions can always enter the GATE block. If the Standard Logical Attribute LRj or LSj is true, the transaction moves to the next sequential block following the GATE block. If LRj or LSj is false, the transaction moves to the alternate block specified in field B.

If a field B alternate block is not specified, the GATE LR or GATE LS block operates in a conditional entry mode; i. e., transactions can enter the GATE block only if the Standard Logical Attribute LRj or LSj is true. If LRj or LSj is false, the transaction is placed in a pushdown delay chain (as described later in this chapter) and deactivated from the overall GPSS/360 scan. The only exceptions are transactions in TRANSFER BOTH or ALL blocks. When another transaction subsequently passes through a LOGIC block which makes the specified attribute true, all transactions in the delay chains are activated. The overall GPSS/360 scan may then be able to advance one or more of these transactions (as well as those in TRANSFER BOTH or ALL blocks) into the conditional entry GATE LR or GATE LS block.

EFFECT OF RESET, CLEAR, AND JOB CARDS

All Logic Switches are initially in a reset or off condition. Logic Switch conditions are not changed by a reset card. All Logic Switches are reset to an off condition by clear and job cards.

STATISTICAL OUTPUT

There is no statistical output for the status of Logic Switches at the end of a run. However a printout of Logic Switches in a set status (on) will appear after a running error.

EXAMPLES OF LOGIC, GATE LR, AND GATE LS BLOCKS

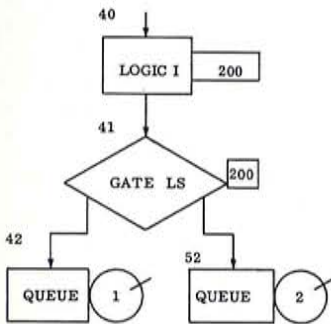
Example 1:

The following block diagram may be used to send transactions to alternate paths. The first

transaction that enters block 40 will set Logic Switch 200, and thus will enter the GATE block and proceed to the next sequential QUEUE block 42. The second transaction that enters the LOGIC block, however, will reset Logic Switch 200. The Standard Logical Attribute LS200 is now false and the transaction will thus be diverted to the alternate path, QUEUE block 52. The third transaction will follow the first, and the fourth will follow the second, etc.

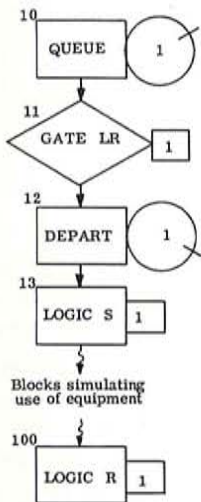
* ALTERNATOR

40	LOGIC I	200
41	GATE LS	200, 52
42	QUEUE	1
52	QUEUE	2

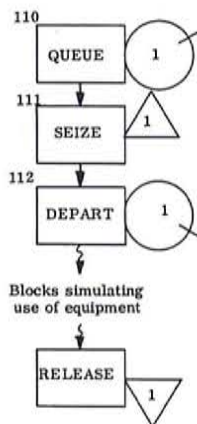


Example 2:
Logic Switches can be used to simulate unit capacity equipment, similar to facility entities. The following block diagram compares these uses of Logic Switches and facilities.

Logic Switch 1 as a Unit Capacity Equipment



Facility 1 as a Unit Capacity Equipment



Since Logic Switches are initially in a reset condition; the first transaction succeeds in entering GATE LR block 11. It immediately closes entry to this GATE block by placing Logic Switch 1 in a set condition in LOGIC block 13.

A succeeding transaction will be able to enter GATE LR block 11 only when the preceding transaction places Logic Switch 1 back into a reset condition in LOGIC R block 100. Logic Switch 1 is therefore similar to Facility 1, with the important exceptions (described in Chapter 10) that no usage statistics are gathered and PREEMPTING is not allowed. If these last two factors are unimportant, then it is often desirable to use Logic Switches to simulate unit capacity equipment instead of facilities. Each Logic Switch entity requires only three half-words, while each Facility requires seven fullwords.

Example 3:

The polling of messages at communication line terminals is generally simulated with Logic Switches and artificial polling transactions. The block diagram of Figure 28 shows some of the blocks in a typical polling model. The actual messages enter QUEUE block 11 and attempt to enter GATE LS block 12. Parameter 6 of each transaction contains its input terminal number. Parameter 5 contains the line number with which the terminal is associated.

Since each entity, (QUEUE, Logic Switch, Facility) is referenced indirectly (*6 and *5,) transactions representing each type of message pass through the same block stream.

Each terminal, therefore, has a Logic Switch whose index number is the same as the terminal number (*6). The terminal Logic Switch *6 will be placed in a set condition only in LOGIC block 51 by an artificial polling transaction which is associated with the particular communication line to which the terminal is connected.

There will be one artificial polling transaction for each communication line. Each polling transaction carries the line number (*5) in Parameter 5. However, the terminal number (*6) in Parameter 6 can be varied, depending on the polling discipline being used; this, thereby controls which terminal is allowed to transmit input messages. This is accomplished in LOGIC block 51, where the Logic Switch representing the currently active terminal, (*6) is put into a set condition. Observe that the artificial polling transaction will first TEST, in block 50, whether or not there are any messages to be transmitted at terminal *6, i.e., whether

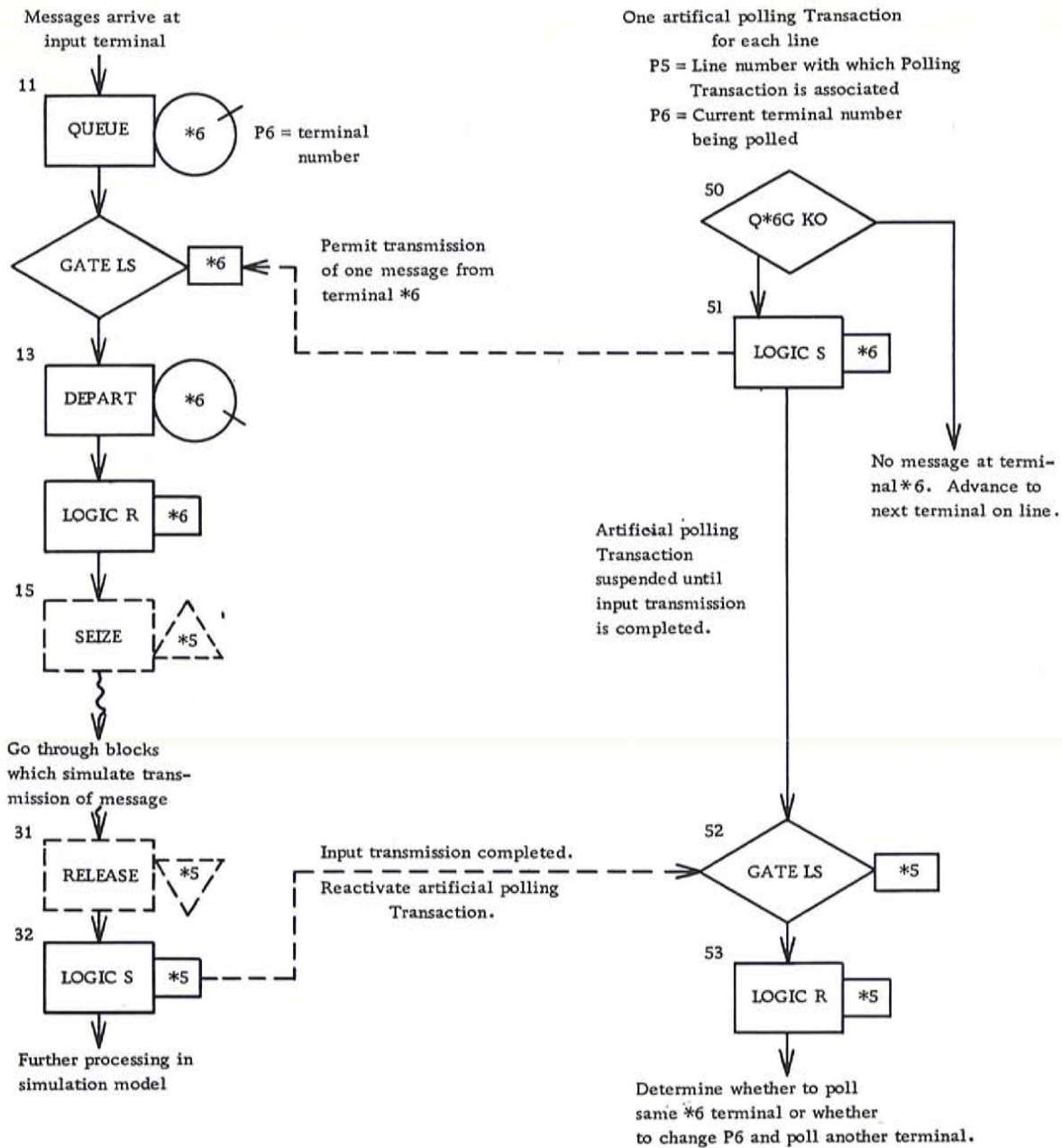


Figure 28. Simulation of Communication Line Polling.

$Q*6$ is greater than zero. If there are no transactions in $Q*6$ the polling transaction bypasses LOGIC S block 51.

Whenever an input message succeeds in entering GATE LS block 12 it immediately puts the terminal Logic Switch *6 back into a reset condition. The message may SEIZE line Facility *5, primarily for the statistical purpose of obtaining the average line utilization. This facility may, however, also be PREEMPTed by the artificial polling transaction when it sends high-priority polling messages to the

terminals. This would happen, for instance, if the line was half duplex; i. e., it could only transmit messages in one direction at any time.

After putting Logic Switch *6 in a set condition, the artificial polling transaction hangs up on GATE LS block 52, and references a Logic Switch (*5) which is associated with the line (not the terminal). It will only be able to proceed to further polling when the actual message, which is being transmitted over the input line, moves through LOGIC S block 32 and puts the line Logic Switch *5 into a set condition.

As soon as this happens, the artificial polling transaction typically puts the line Logic Switch *5 back into a reset condition so that it can subsequently hang up on GATE LS block 52.

Example 4:

A typical way to simulate the use of the central processing unit (CPU) in a data processing system is to simply attempt to SEIZE the CPU facility.

Eight priority classes of messages can be created to control the sequence in which transactions are permitted to seize the CPU. For more detailed and sophisticated simulation of a control programming system, Logic Switches and an artificial central control program transaction can be used. The block diagram of Figure 29 shows some typical blocks in a control programming simulation model.

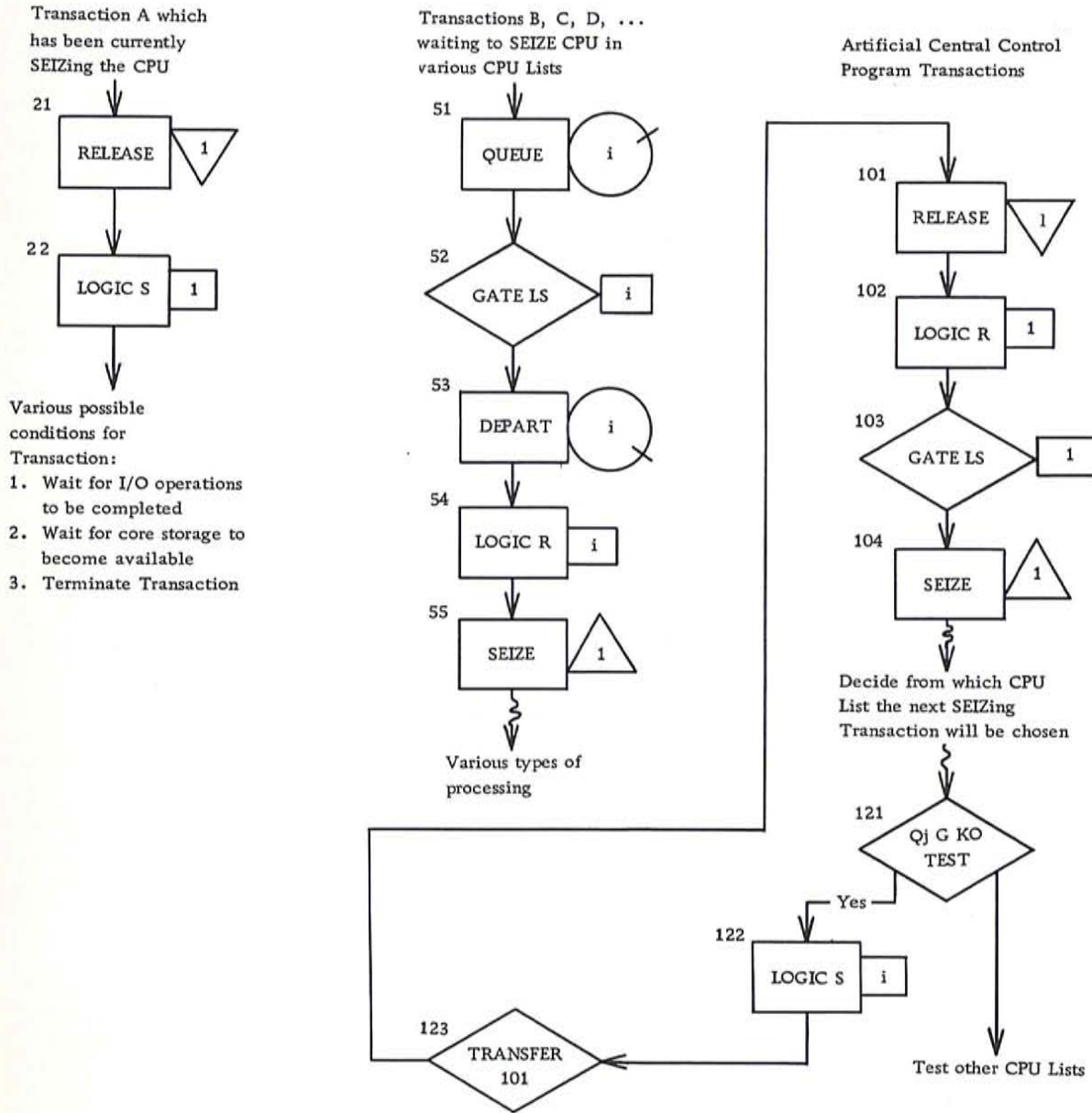


Figure 29. Control Programming Simulation

Assume that Transaction A, which is currently SEIZING the CPU (Facility 1), finally RELEASES the CPU in block 21. It next places Logic Switch 1 in a set condition in LOGIC S block 22 and thereby activates the artificial central control program transaction which has been hung up on GATE LS block 103. There are a variety of conditions under which Transaction A will surrender control of the CPU. These include:

1. One or more I/O operations have been initiated, and Transaction A must wait until these are completed.
2. Core storage is not available to be allocated, and Transaction A must wait until it becomes available.
3. Transaction A may have completed all its processing, and will be destroyed in a TERMINATE block.

Assume that, meanwhile, other transactions (B, C, D, ...) which want to SEIZE the CPU have been placed in various CPU lists. These lists are represented by blocks such as 51 to 54 and an associated *i*th Queue. Observe that one of these blocked transactions will succeed in SEIZING Facility 1 only when the central control program transaction decides to put a particular *i*th Logic Switch into a set condition, e.g., in LOGIC S block 122.

When the central control program transaction finally decides from which CPU list the next transaction will SEIZE the CPU, it TRANSFERS back to block 101 where it RELEASES the CPU, puts Logic Switch 1 in a reset condition, and then hangs up on GATE LS block 103, which is waiting for the chosen transaction to reactivate it by putting Logic Switch 1 in a reset condition.

Observe that use of the artificial central control program transaction can virtually supersede the overall GPSS/360 scan and its 128 priority classes in determining which transaction next SEIZES the CPU.

Example 5: A Subtle Delay Chain Problem. Assume that a GATE LR or GATE LS block is operating in a conditional entry mode; i.e., there is no alternate field B block. Transactions which are delayed at such a block will be put in a unique pushdown delay chain which is associated with a particular Logic Switch (the only exception is if the transaction is in a TRANSFER block with a BOTH or ALL selection mode where there are nonunique blocking conditions). Suppose a transaction is attempting to enter the following type of GATE Block:

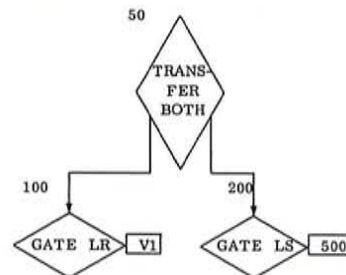
GATE LR SNAj, SNA*n

The field A Standard Numerical Attribute gives the index number of a particular Logic Switch. Suppose it was $V1 = Q10 + K1$; i.e., the Logic Switch number is given by the current contents of Queue 10 (= 0, 1, 2, ...) plus the constant 1. Therefore, the following GATE block would reference Logic Switches 1, 2, ... etc.

Assume now that the transaction attempts to enter the GATE LR block when $Q10 = 7$. The referenced Logic Switch 8 (i.e., $V1 = 7 + 1 = 8$) is in a set condition. The transaction is therefore put in the GATE LR delay chain which is associated with Logic Switch 8. More importantly, the scan status indicator, (bit 0 of byte T10), of the delayed transaction is set to one, thereby deactivating the transaction from being processed by the overall CPSS/360 scan. This transaction will remain inactive until Logic Switch 8 is reset and the delay chain is reactivated.

Assume that, at some later point in time, the contents of Queue 10 are reduced to six. Assume also that Logic Switch 8 is still in a set condition, but Logic Switch 7 is in a reset condition. The delayed transaction will never be processed by the overall GPSS/360 scan, even though it could enter the GATE LR block which would now be referencing Logic Switch 7, instead of 8 as formerly. This may or may not be the condition which the analyst wishes to simulate.

The following three blocks could be used where it is desired to keep the transaction continually active, so that the overall GPSS/360 scan will always attempt to move the transaction into GATE LR block 100. The Logic Switch number which is referenced will then be the current value of Arithmetic Variable V1.



The following assembly program coding would be used:

```

TRANSFER BOTH, 100,200
GATE LR V1
GATE LS 500
  
```

This approach requires that Logic Switch 500 will always be in its initial reset condition, so that transactions will never succeed in entering GATE LS block 200. Because they are in a TRANSFER block with a BOTH selection mode, the delayed transactions are never placed in a delay chain and are

never deactivated from being processed by the overall GPSS/360 scan.

PUSHDOWN DELAY CHAINS FORMED BY CONDITIONAL ENTRY GATE LR AND GATE LS BLOCKS

Table 16 describes three halfwords (L1, L2, and L3) which represent each Logic Switch entity. L1 halfword is zero when the Logic Switch is reset, and nonzero when the Logic Switch is set.

Consider what happens when transactions attempt to enter a conditional entry GATE LS block when the Logic Switch referenced by the field A argument is in a reset condition. Assume that the block from which the transaction is attempting to enter the GATE LS block is any block type except a TRANSFER block with a BOTH or ALL selection mode. Transactions in these last two block types will never be put in pushdown delay chains.

Assume that Transactions 600 and 35 successively fail to enter the following GATE LS block:

```
103 GATE LS 300
```

Figure 30 shows the pertinent attributes of Logic Switch 300 and of the two transactions, after each of the failures to enter the above GATE LS block.

When Transaction 600 becomes the first transaction to fail to enter GATE LS block 103, the following occurs:

1. The number (600) of the transaction is placed in L2 halfword for Logic Switch 300.
2. The scan status indicator (bit 0 of byte T10) of Transaction 600 is set to one, thereby deactivating Transaction 600 from being processed by the overall GPSS/360 scan.
3. The first halfword of T1 transaction word of Transaction 600 is set to zero; this indicates that the transaction is the first one put onto the LS delay chain for Logic Switch 300.

Assume now that Transaction 35 also attempts to enter GATE LS block 103 while Logic Switch 300 is still in a reset condition. The following steps occur:

1. The number of the transaction (35) is placed in L2 halfword for Logic Switch 300.
2. The scan status indicator of Transaction 35 is set to one, thereby deactivating Transaction 35 from being processed by the overall GPSS/360 scan.
3. The first half of T1 word of Transaction 35 is set to 600, thereby creating a pushdown delay chain which links Transaction 35 to 600.

If any further transactions attempt to enter GATE LS blocks which reference Logic Switch 300, steps 1, 2, and 3 above for Transaction 35 repeated. The number of the most recently delayed transaction is

put in L2 halfword. Concurrently, the number of the previous delayed transaction is transferred from L2 halfword to the first of T1 word of the most recently delayed transaction. (The contents of T1 halfword are not printed out in the transaction printout.)

Effect of LOGIC S Block on GATE LS Delay Chain

Assume now that another transaction enters a LOGIC S block which references Logic Switch 300, and puts this switch into a set condition. The LOGIC block subroutine checks if there is a GATE LS delay chain hung up on Logic Switch 300. This is indicated by the presence of a nonzero transaction number L2 halfword. In our example, there is such a delay chain.

The LOGIC block subroutine then zeroes out L2 halfword. It also zeroes out the first half of T1 word linkage of each transaction in the delay chain. Most importantly the scan status indicator (first bit of T10 byte) is reset to zero so that the overall GPSS/360 scan will once again attempt to move each one of the delayed transactions into GATE LS Block 103, or other GATE LS blocks which reference Logic Switch 300. The end of the pushdown delay chain is indicated by the zero which is stored in the first half of T1 word of Transaction 600; i. e., there is no next transaction in the delay chain after Transaction 600.

The status change flag is set "on" when the state of a Logic Switch is changed in a LOGIC block. Consequently, the overall GPSS/360 scan will automatically transfer back to the start of the current events chain when it finishes moving the transaction which entered the LOGIC block. The GPSS/360 scan may, of course, transfer earlier if the transaction enters a BUFFER block or a PRIORITY block with a BUFFER option. Observe that the relative positions of the delayed transactions in the pushdown delay chain have no relationship to the relative positions of these transactions in the current events chain. The current events chain linkages (which are defined by Transaction word T2) govern the order in which the overall GPSS/360 scan will attempt to move the transactions into the previously blocked GATE LS blocks.

By the time that the overall GPSS/360 scan gets around to processing the reactivated transactions that were once in a pushdown delay chain, it is possible that:

1. None of the reactivated transactions will be able to enter the GATE LS block, because some other transaction has entered a LOGIC R block and has thereby restored the blocking condition.
2. All of the reactivated transactions will be able to enter the GATE LS block.

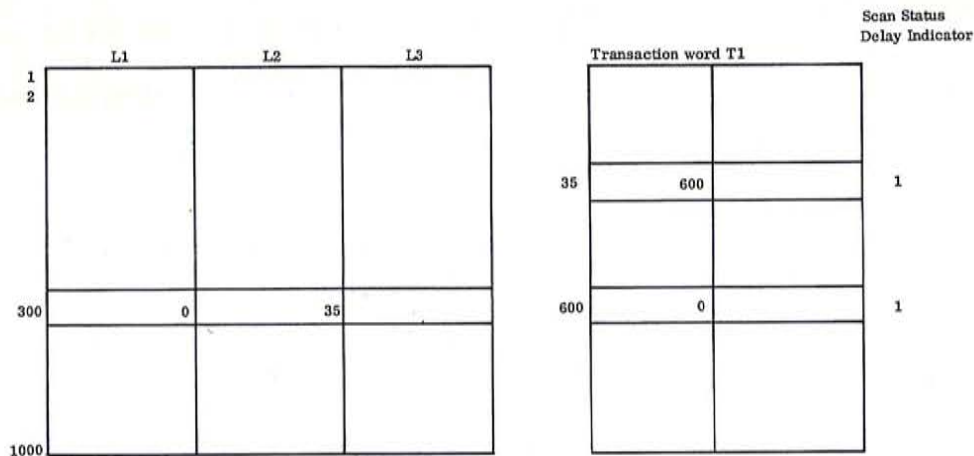


Figure 30. Pushdown Delay Chain Formed by GATE LS Block

3. Only some, possibly just one, of the reactivated transactions will be able to enter the GATE LS block. Example 2, "Unit Capacity Equipment" earlier in this chapter, shows a typical unit capacity situation where only one transaction succeeds in entering GATE LR block 11 because this transaction itself puts Logic Switch 1 into a blocking set condition.

GATE LR Delay Chain and LOGIC R Blocks

The same procedures are followed when a Logic Switch is in a set condition and transactions attempt to enter conditional entry GATE LR blocks which reference the particular Logic Switch. The only differences are that the pushdown delay chain starts from L3 halfword rather than L2 halfword. Whenever a transaction enters a LOGIC R block, the LOGIC block subroutine checks whether L3 halfword contains a transaction number; i. e., whether there is a pushdown delay chain of transactions hung upon one or more GATE LR blocks which reference the particular Logic Switch.

Overall Pushdown Delay Chain Considerations

Facilities (Chapter 10) and Storages (Chapter 11) also have various pushdown delay chains associated

with them. These are created when transactions fail to enter the following block types:

- | | |
|------------|----------|
| FACILITIES | STORAGES |
| SEIZE | ENTER |
| PREEMPT | GATE SE |
| GATE U | GATE SNE |
| GATE NU | GATE SF |
| GATE I | GATE SNF |
| GATE NI | |

These delay chains also begin in various S/360 core words which are associated with each facility or storage. The first half of Transaction word T1 is again used to create the one-way, push-down chain linkages. Transactions which are trying to enter from a TRANSFER block with a BOTH or ALL selection mode are, again, exceptions. Because the blocking condition is nonunique, they cannot be put in a unique delay chain. Therefore, they always remain active with respect to the overall GPSS/360 scan. The "Subtle Delay Chain Problem" (Example 5), discussed earlier in this chapter, applies to all these delay chains.

TABLE 18: S/360 CORE ALLOCATION FOR FACILITY ENTITIES (continued)


<u>SYMBOL</u>	<u>LENGTH</u>	<u>QUANTITY</u>	<u>SOURCE</u>
F3	4 Bytes	Cumulative time integral of Facility usage.	This word is a fixed-point sum of the total number of clock units for which the Facility has been either SEIZED or PREEMPTed, or both. The sum is updated whenever the Facility changes status.
F4	4 Bytes	Clock time of last status change.	This word contains the clock time at which the time integral was last updated. The absolute clock time is placed in this word each time the Facility changes status, and also when a RESET or CLEAR card is read.
F5	2 Bytes	Delay chain origin, for all transactions waiting for the Facility to be in use, i. e. , SEIZED and/or PREEMPTed.	Each time that a transaction at a GATE U block fails to advance because the Facility is not in use, the number of that transaction is put in this field (unless the transaction is in a TRANSFER block with BOTH or ALL selection mode). The previous transaction number in the field is placed in the first two bytes of Transaction word T1, thus forming a pushdown delay chain of transactions waiting for a type of service on the Facility. When a transaction enters a SEIZE or PREEMPT block, the delay chain is reactivated; that is, the scan indicators of all transactions on the delay chain are set on and the delay chain origin is set to zero.
F6	2 Bytes	Delay chain origin, for all transactions waiting for the Facility not to be in use.	This pushdown delay chain is formed in the same manner as described above when a transaction fails to advance because the Facility is in use (SEIZED and/or PREEMPTed). The chain is reactivated whenever a transaction enters a RELEASE or RETURN block. These transactions are either trying to SEIZE the Facility or are blocked by a GATE NU block.
F7	2 Bytes	Delay chain origin, for all transactions waiting for the Facility to be PREEMPTed.	This pushdown delay chain is formed in the same manner as described above, and is cleared whenever a transaction enters a PREEMPT block. These transactions are blocked by a Gate 1 block.
F8	2 Bytes	Delay chain origin, for all transactions waiting for the Facility not to be PREEMPTed.	This pushdown delay chain is formed in the same manner as described above, and is cleared whenever a transaction enters a RETURN block. These transactions are either trying to PREEMPT the Facility or are blocked by a GATE NI block.
F9	4 Bytes	Entry Count. This is Standard Numerical Attribute FC _j .	This word is incremented by one whenever a transaction enters a SEIZE or PREEMPT block.
F10	4 Bytes	Address of multiple PREEMPT list.	This is the address of the COMMON area containing a list of transactions which have been PREEMPTed by other transactions of higher priority in a PREEMPT block operating in the priority mode.

INTERNAL NONADDRESSABLE ATTRIBUTES

Table 18 describe the various internal attributes which are stored in the seven Facility words but are not directly addressable by the user. These include:

1. The number of the transaction currently SEIZEing the Facility (F1).
2. The number of the transaction currently PREEMPTing the Facility (F2).
3. The absolute clock time of the last status change, either from in use to not in use, or from not in use to in use F4 word).
4. Cumulative time integral, i. e. , the total number of clock units during which the Facility was in use, being SEIZED and/or PREEMPTed (F3 word). This is the total since the last RESET or CLEAR card (or during the first simulation run of a job).
5. The number of the first transaction in each of four pushdown delay chains which are associated with each Facility (F5, F6, F7, F8).

SEIZE BLOCK

2	LOC	7	8	OPERATION	INDEX NUMBER OF FACILITY	SEIZE 
				SEIZE	19 A SNAJ, SNA*n, k, *n	

A transaction is not permitted to SEIZE a Facility which is already in use, i. e. , being SEIZED and/or PREEMPTed by other transaction(s). When a transaction attempts to enter a SEIZE block, the SEIZE block subroutine tests whether the Facility is in use or not by testing whether the word consisting of F1 and F2 of the Facility referenced by the field A argument is zero or nonzero.

Procedure When Facility Is Not in Use

Word F1-F2 is zero when the Facility is not in use, and can therefore be SEIZED. The SEIZE block subroutine places the number of the new SEIZEing transaction in F1. The current absolute clock time is placed in word F4 to record the time of the latest status change, i. e. , from not in use to in use. The total entry count (word F9) is incremented by one. The SEIZEing transaction then attempts to enter the next sequential block which follows the SEIZE block. The Facility will remain SEIZED until the transaction enters a corresponding RELEASE block, at which time the transaction number is removed from F1. The SEIZEing transaction can move through an unlimited number of blocks after SEIZEing a Facility, and before RELEASEing it.

Procedure When Facility Is in Use

Word F1-F2 is nonzero when the Facility is in use, i. e. , being SEIZED and/or PREEMPTed. This is because transaction numbers are in F1 and/or F2. The transaction, therefore, is not permitted to enter the SEIZE block. Instead, it is linked into a push-down delay chain of transactions which also may include transactions blocked by SEIZE or GATE NU blocks which reference the particular Facility. (As usual, the transaction is not chained if it is in a TRANSFER BOTH or ALL block.) The number of this latest delayed transaction is placed in F6, in exactly the same manner as described for GATE LR and GATE LS delay chains in Chapter 9. The scan status indicator of the transaction is set to one, thereby deactivating the transaction from processing by the overall GPSS/360 scan.


Interactions with Other Facilities and Transactions

A transaction may SEIZE any number of Facilities, and may PREEMPT any number of transactions on other Facilities. A Transaction which has SEIZED a Facility is subject to being PREEMPTed on that Facility. A transaction may be PREEMPTed on as many as 127 Facilities which it has SEIZED. These PREEMPTing operations are described under the PREEMPT block later in this Chapter.

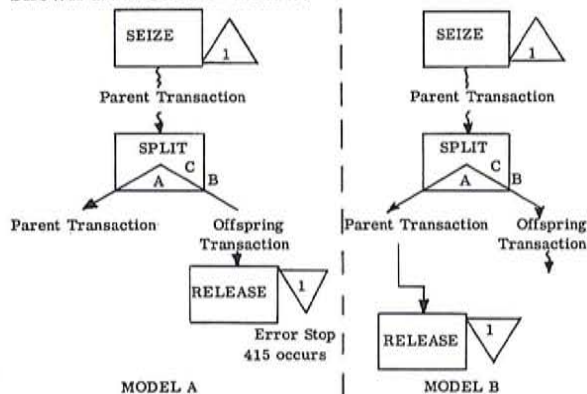
Status Change Flag and Reactivation of GATE U Delay Chain

When a transaction enters a SEIZE block there may be a GATE U delay chain waiting for the Facility to be in use (this is indicated by a transaction number in F5). Each transaction in the GATE U delay chain is reactivated, i. e. , their scan status indicators are reset to zero. The status change flag is also automatically set to "on" whenever a transaction enters a SEIZE block. Consequently, the overall GPSS/360 scan will automatically transfer back to the start of the current events chain when it finishes moving the transaction which entered the SEIZE block. This insures that all of the reactivated transactions in the former GATE U delay chain will be processed by the overall GPSS/360 scan.

RELEASE BLOCK

2	LOC	7	8	OPERATION	FACILITY NUMBER	RELEASE 
				RELEASE	19 A SNAJ, SNA*, k, *n	

The RELEASE block serves the function of removing from a Facility the transaction which has SEIZED it. The RELEASEing operation is performed immediately upon entering the RELEASE block. There is no further effect on the Facility if the transaction is subsequently prevented from entering the next sequential block following the RELEASE block. A transaction is never refused entry to a RELEASE block. The Facility referenced by the field A argument of the RELEASE block must have been SEIZED by the entering transaction. Otherwise, Running Error Stop 415 will occur: "Facility RELEASE by a Transaction not SEIZEing it". No other transaction except the one that originally SEIZED a Facility is entitled to RELEASE it. As an example, the analyst must avoid the classic error shown in model A below:



In model A, the offspring transaction, which has a different number than the parent transaction, attempts to RELEASE Facility 1 which has been SEIZED by the parent transaction. Execution Error #415 occurs immediately. Model B shows the correct procedure. The parent transaction will be the only transaction proceeding to the next sequential block after the SPLIT block (the one or more offspring move to field B next block). It is therefore possible to keep track of the parent transaction in the block diagram and thereby ensure that it will RELEASE the same Facility that it SEIZED.

How a Wrong Transaction with the Correct Number Can RELEASE or RETURN a Facility

The following unusual situations might occur in a GPSS/360 model. A transaction, say number 100, SEIZES or PREEMPTS a Facility, say number 10. Subsequently, Transaction 100 is destroyed in a TERMINATE or ASSEMBLE block without ever having RELEASED or RETURNED Facility 10. The four 360 words (T1, T2, T3, T4) of Transaction 100 are returned to the internal chain of unused transactions. F1 of Facility 10 still contains the

number 100, indicating that it is being SEIZED or PREEMPTed by Transaction 100.

The GPSS/360 user should be aware of three unusual things which can now happen. First, a statistical printout can occur while Transaction 100 is still in the internal chain of unused transactions. Consequently, the statistics for Facility 10 show that it is being SEIZED or PREEMPTed by Transaction 100, but no Transaction 100 will be found in the transaction printout.

As a second possibility, Transaction 100 may be brought back into the model as a new transaction, either in a GENERATE block or in a SPLIT block. A statistical printout will now show that Facility 10 is being SEIZED or PREEMPTed by Transaction 100, and that there is also a transaction with the number 100 in the Transaction printout. However, this transaction is not really the one which originally SEIZED or PREEMPTed Facility 10. It is merely using the same set of four 360 words, which store the transaction attributes.

The third and most remote possibility is that the second Transaction 100 will actually enter a RELEASE or RETURN block which references Facility 10. The GPSS/360 program obviously cannot distinguish the second Transaction 100 from the first one. Consequently, the wrong transaction with the right number will successfully RELEASE or RETURN Facility 10. This will undoubtedly have an adverse effect on the logic of the model and on the statistics relating to the Facility involved.

Status Change Flag and Reactivation of SEIZE-GATE NU Delay Chain

When a transaction enters a RELEASE block there may be a delay chain of transactions blocked at SEIZE or GATE NU blocks for the Facility to be not in use. This is indicated by a transaction number in F6. Each delayed transaction is reactivated; i. e., their scan status indicators are reset to zero. The status change flag is automatically set to "on" whenever a transaction enters a RELEASE block. Consequently, the overall GPSS/360 scan will automatically transfer back to the start of the current events chain when it finishes moving the transaction which entered the RELEASE block. This ensures that all of the reactivated transactions in the former SEIZE-GATE NU delay chain will be processed by the overall GPSS/360 scan. All, none, or just some of the delayed transactions (including those in TRANSFER BOTH or ALL blocks) may actually be able to enter the SEIZE or GATE NU block(s).

Cumulative Time Integral of Facility Utilization

When a transaction RELEASES a Facility there is generally a status change; i. e. , the Facility changes from being in use to not in use (see "PREEMPT Block" below, which explains how, when a Facility has been PREEMPTed, the SEIZEing transaction is still able to move into a RELEASE block). When such a status change occurs, the following quantity is added to the cumulative time integral stored in word F3:

$$\text{Additional Clock Units in Use} = \text{Current Clock Time} - \text{Absolute Time of Last Change (Word F4)}$$

Word F3, therefore, accumulates the total number of clock units during which the Facility was in use, i. e. , SEIZED and/or PREEMPTed. The current absolute time is then stored in word F4 as the time of the last status change.

PREEMPT BLOCK

2	LOC	7	8 OPERATION	19 FACILITY NUMBER	PRIORITY OPTION	OPTIONAL BLOCK FOR PREEMPTED TRANSACTION	PARAMETER NO. OF PREEMPTED TRANSACTION	REMOVE OPTION
			PREEMPT	SNAj, SNA*n *n,k	PR	SNAj, SNA*n *n,k	SNAj, SNA*n	RE

The PREEMPT block bears the same relation to the preempting of a Facility that the SEIZE block bears to the normal seizing of a Facility. The important exception is that while a transaction may preempt a Facility which is currently being seized by another transaction, a transaction may not seize a Facility which is currently being seized or preempted by another transaction. The PREEMPTing of a Facility is not terminated until the same transaction which PREEMPTed the Facility enters a RETURN block which references the same Facility.

A transaction entering a PREEMPT block may have SEIZED any number of other Facilities, and may have PREEMPTed any number of other transactions on Facilities which they have SEIZED. It is also subject to being PREEMPTed on any of the Facilities which it has SEIZED.

The simplest and most common usage of the PREEMPT block is to specify the Facility number in field A. In previous versions of the GPSS Program this was the only usage possible. However, the operation of the PREEMPT block has been expanded in GPSS/360 to provide multilevels

of preempting as well as other options. These optional features will be discussed later in this section. The current discussion assumes only a field A is specified in the PREEMPT block.

Under this condition a transaction is not permitted to PREEMPT a Facility which has already been PREEMPTed by another transaction. It does, in this sense, operate in the same manner as the SEIZE block.

When a transaction attempts to enter a PREEMPT block, the PREEMPT block subroutine tests the word containing F1 and F2 of the Facility which is referenced by the field A argument of the PREEMPT block. There are three possible conditions: Case 1. The Facility is not in use; i. e. , it is neither being SEIZED nor PREEMPTed. (F1 and F2 are both zero.) Case 2. The Facility is already being PREEMPTed. (F2 contains the number of the PREEMPTing transaction.) There is no difference whether the Facility is being SEIZED or not. Case 3. The Facility is being SEIZED but is not PREEMPTed. (F2 is zero, while F1 contains the number of the SEIZEing transaction.)

Case 1. The facility is being neither SEIZED nor PREEMPTed

The PREEMPT block behaves much like the SEIZE block. The PREEMPT block subroutine places the number of the new PREEMPTing transaction in F2. Subsequent transactions can neither PREEMPT nor SEIZE the Facility until the PREEMPTing transaction enters a RETURN block. The current absolute clock time is placed in word F4 to record the time of the last status change, i. e. , from not in use to in use. The total entry count (word F9) is incremented by one. The PREEMPTing transaction then attempts to enter the next, sequential block following the PREEMPT block.

Case 2. The Facility is already being PREEMPTed

The PREEMPT block again behaves much like a SEIZE block. The transaction is not permitted to enter the PREEMPT block. Instead, it is linked into a pushdown delay chain of transactions which are blocked by PREEMPT or GATE NI blocks that reference the particular Facility (as usual, the transaction is not chained if it is in a TRANSFER BOTH or ALL block). The number of this latest delayed transaction is placed in F8. The scan status indicator of the transaction - is set to one, thereby deactivating the transaction from being processed by the overall GPSS/360 scan.

Case 3. The Facility is being SEIZED but is not PREEMPTed

This is the most complicated and important case. The number of the SEIZEing transaction is contained in F1. The transaction entering the PREEMPT block is first processed in the same manner as in Case 1. The operations performed on the SEIZEing transaction, however, involve four alternatives, namely 3A, 3B, 3C, or 3D, which are described below.

Case 3A. The SEIZEing transaction has already been PREEMPTed on another Facility which it has SEIZED, and is in an interrupt status

Bit 6 of T10 (see Table 9 in Chapter 7) is set to one whenever a transaction is PREEMPTed on a Facility which it has SEIZED. These PREEMPTed transactions are in an individual interrupt status and belong to neither the current events nor the future events chain. They can only be in ADVANCE, MATCH, ASSEMBLE, or GATHER blocks. T8 maintains a count of the number of Facilities on which the transaction has been PREEMPTed. Consequently, for CASE 3A the preempt count of the SEIZEing transaction will subsequently be decremented by one when the PREEMPTing transaction enters a RETURN block. When the preempt count is finally decremented to zero, the interrupted transaction is linked back into the current events chain. Execution error 474 will occur if the preempt count for SEIZEing transaction exceeds 127. This means that a transaction can be PREEMPTed on no more than 127 Facilities which it has SEIZED.

Case 3B. The SEIZEing transaction has not been PREEMPTed on any other Facility, and is in the future events chain, i. e. , in an ADVANCE block

If the SEIZEing transaction has not been PREEMPTed on any other Facility, Bit 6 of T10 is zero. The PREEMPT block subroutine then tests T9 to determine whether the transaction is in the current events or the future events chain.

If bit 5 of word T9 is one, the SEIZEing transaction is in the future events chain, i. e. , in an ADVANCE block. Word T4, therefore, contains the block departure time from the ADVANCE block, which must be greater than the current absolute clock time. The following operations are now performed on the SEIZEing transaction:

1. The remaining time in the ADVANCE block is stored in word T4, where

Remaining	Block Departure	Current
ADVANCE =	Time	-Absolute
Block Time	(Word T4)	Clock Time

2. Bit 6 of T10 is set to one, indicating that the transaction is being PREEMPTed on a Facility.

3. The preempt count (T8) is set to one.

4. Bit 7 of T9 is set to one to indicate that the transaction is in an interrupt status.

5. The SEIZEing transaction is unlinked from the future events chain.

"RETURN Block" later in this chapter, describes how these transactions are removed from a PREEMPTed interrupt status and are merged back into the future events chain.

BDT column printout for PREEMPTed Transactions

The BDT column in the transaction printout (see Table 10 in Chapter 7) generally lists the most recent block departure time for all transactions. However, in the case of PREEMPTed transactions in ADVANCE blocks, the BDT values are the remaining block times. Therefore, these values may be extremely low in comparison to the absolute BDT times of other transactions.

Case 3C. The SEIZEing transaction has not been PREEMPTed on any other Facility, and is in a matching condition in a MATCH, ASSEMBLE or GATHER block

For these transactions, the preempt status indicator (Bit 6 of T10) is zero. However, the matching status indicator (Bit 5 of T10) is one indicating that the transaction is in a matching condition at a MATCH, GATHER or ASSEMBLE block. Such transactions are, in a sense, already in an interrupt status; i. e. , they belong to neither the current or the future events chain. The following operations are performed on the SEIZEing transaction.

1. The preempt flag (Bit 7 of T10) is set to one to indicate to the MATCH, GATHER or ASSEMBLE block subroutine that this transaction is to be put in a preempt status when the matching condition is met.

2. The preempt count (T8) is incremented by one. Error 474 will again occur if this count exceeds 127.

Transactions in a MATCH, ASSEMBLE, or GATHER block, which are both being PREEMPTed and are in a matching condition, will be removed from an interrupt status and returned to the current events chain only when both the preempt count is decremented to zero, and the necessary matching, assembly, or gathering operation is successfully completed.

Case 3D. The SEIZEing Transaction is in the Current Events Chain

For these transactions, bit 6 of T9 is one indicating that they are in the current events chain. Also, the preempt status indicator (Bit 6 of T10) is zero.

These transactions are in two general states:

1. They have been blocked in entering a next block. (They may be linked in a pushdown delay chain, with their scan status indicator set to on.)

2. They have just been transferred to the current events chain, but have not as yet been processed by the overall GPSS/360 scan at the current clock time. The GPSS/360 program applies the following very important rule with regard to transactions in the current events chain which are to be PREEMPTed on a Facility which they have SEIZED:

Rule: Current events chain transactions, which have been PREEMPTed on one or more Facilities, will not be placed immediately in an interrupt status. They will be removed from the current events chain only when they enter:

1. A positive time ADVANCE block (Case 3B above)
2. A MATCH, ASSEMBLE, or GATHER block in which they are put into a matching condition (Case 3C above)

Consequently, these transactions can move through an unlimited number of other block types, and can be delayed an unlimited number of times before they are finally put into an interrupt status in one of the above four block types. It is possible that the SEIZING transaction will enter a RELEASE block and thereby completely avoid being put into an interrupt status.

Alternately, these transactions may never be placed in an interrupt status if the PREEMPTing transaction(s) enter RETURN block(s) before the SEIZING transaction enters an ADVANCE, MATCH, ASSEMBLE, or GATHER block.

The following operations are performed on Case 3D transactions:

1. The preempt count is incremented by one.
2. The preempt flag is set to one to indicate that the transaction is to be put onto a PREEMPTed interrupt status when it next enters an ADVANCE, MATCH, ASSEMBLE, or GATHER block. The value of the preempt flag (0 or 1) is printed in the PF column of the transaction printout.

The preempt flag is tested whenever a transaction enters one of the above four block types. If it is set to one, the following steps occur:

ADVANCE BLOCK

- a. The nonzero ADVANCE block time is stored in the T4 block departure time word. The transaction will spend this time in the ADVANCE block only after the preempt count of the transaction has been decremented to zero. The transaction is then finally merged into the future events chain, as described in "RETURN Block" later in this chapter.

- b. The preempt flag is reset to zero.

- c. The preempt status bit is set to one.
- d. Bit 7 of word T9 is set to one to indicate that the transaction is in interrupt status.
- e. The transaction is unlinked from the current events chain.

MATCH, ASSEMBLE, OR GATHER BLOCK

- a. The matching condition bit (Bit 5 of T10) is set to one according to the procedures described for these three block types in Chapter 7.

Steps b, c, d, and e are the same as those for the ADVANCE block above.

RELEASEING OF A PREEMPTED FACILITY BY A CURRENT EVENTS CHAIN TRANSACTION

Because of the rules which apply to current events chain transactions (Case 3D), it is possible for a transaction to move into a RELEASE block which refers to a Facility that has also been PREEMPTed by another transaction. The transaction entering the RELEASE block, therefore, succeeds in RELEASEing the facility. The RELEASE block subroutine decrements the preempt count by one.

If the preempt count is decremented to zero (i. e., the current Facility is the only one which the transaction has SEIZED that has been PREEMPTed), the preempt flag is reset to zero. If the preempt count is still one or more, the preemption flag remains set to one. This means that the transaction is still subject to being put into a PREEMPTed interrupt status if it enters an ADVANCE, MATCH, ASSEMBLE, or GATHER block.

STATUS CHANGE FLAG AND REACTIVATION OF GATE I AND GATE U DELAY CHAINS

When a transaction succeeds in entering a PREEMPT block, there may be GATE I and/or GATE U delay chains waiting for the Facility to be interrupted and/or to be in use, respectively (this is indicated by transaction numbers in F5 and/or F7). Each transaction in the GATE I and GATE U delay chains is reactivated; i. e., its scan status indicator is reset to zero. The status change flag is also automatically set "on" whenever a transaction enters a PREEMPT block. Consequently, the overall GPSS/360 scan will automatically transfer back to the start of the current events chain when it finishes moving the transaction which entered the PREEMPT block. This ensures that all of the reactivated transactions in the former GATE I and GATE U delay chains will be processed by the overall GPSS/360 scan.

Extension of the GPSS/360 PREEMPT Block

As was previously mentioned, the operation of the PREEMPT block has been expanded in GPSS/360 to provide multilevels of preempting as well as other options. The most significant of these options is the ability to preempt solely on the priority of the transactions. This, of course, allows 128 levels of preempting. This option is specified by placing the mnemonic PR in field B of the PREEMPT block. This means that if a Facility is currently being PREEMPTed by a transaction with a priority of n, then any other transaction of priority n + 1 or greater will be allowed to PREEMPT the preempting transaction in much the same way that it would if the preempting transaction had only been SEIZing the Facility. The second transaction could then be preempted by any transaction of priority n + 2 or greater, and so on.

Field C of the PREEMPT block may optionally specify a block number to which the preempted transaction will be sent. The preempted transaction will remain in contention for the Facility.

Field D may specify a parameter number associated with the preempted transaction. When preempted, the remaining time the transaction is scheduled to remain in the future events chain is calculated and placed in the parameter specified by the D argument.

The mnemonic RE in field E indicates that the preempted transaction should be completely removed from the chain waiting for the Facility not to be preempted, i. e., when the preempting transaction returns the Facility, the preempted transaction will not be in contention for the Facility. The preempted transaction will not have its preempt count incremented.

The only restriction on the use of field C, D, and E options is that if D and/or E are specified field C must also be specified.

Another fact which should be noted is that if field B (PR) is omitted, then C, D and/or E are ignored.

Also, if the preempted transaction is not on the future events chain when the preempt takes place, the field C, D and E options are ignored.

The remainder of this section lists several possible combinations of arguments for the PREEMPT block and explains the internal operation associated with each combination.

Case I Facility is neither being SEIZED or PREEMPTed

- a.) SNA*n
 - 1. Preempting transaction # is placed in FACILITY location F2

- 2. The current CLOCK time is placed in FACILITY location F4. (Time of last status change.)
- 3. FACILITY location F9 (entry count) is incremented by 1.
- b.) SNA*n, , SNA*n
Same as Ia; C argument is ignored.
- c.) SNA*n, , SNA*n, , RE
Same as Ia; C and E arguments are ignored.
- d.) SNA*n, , SNA*n, SNA*n
Same as Ia; C and D arguments are ignored.
- e.) SNA*n, , SNA*n, SNA*n, RE
Same as Ia; C, D, and E arguments are ignored.
- f.) SNA*n, PR
Same as Ia.
- g.) SNA*n, PR, SNA*n
Same as Ia; C argument is ignored.
- h.) SNA*n, PR, SNA*n, , RE
Same as Ia; C and E arguments are ignored.
- i.) SNA*n, PR, SNA*n, SNA*n
Same as Ia; C and D arguments are ignored.
- j.) SNA*n, PR, SNA*n, SNA*n, RE
Same as Ia; C, D, and E arguments are ignored.

Case II The Facility is already being PREEMPTed

- a.) SNA*n
 - 1. Transaction is not allowed to enter PREEMPT block. It is placed on a pushdown chain for the FACILITY "not preempted".
 - 2. The number of the transaction (attempting to PREEMPT) is placed in FACILITY location F8.
 - 3. The scan status delay indicator bit 0 T10 is set (Discontinue processing).
 - 4. The delay indicator (SIM mode) bit 1 T10 is set.
- b.) SNA*n, , SNA*n
Same as IIa; C argument is ignored.
- c.) SNA*n, , SNA*n, , RE
Same as IIa; C and E arguments are ignored.
- d.) SNA*n, , SNA*n, SNA*n
Same as IIa; C and D arguments are ignored.
- e.) SNA*n, , SNA*n, SNA*n, RE
Same as IIa; C, D, and E arguments are ignored.
- f.) SNA*n, PR
 - A. PREEMPTING TRANSACTION HAS A LOWER PRIORITY THAN TRANSACTION BEING PREEMPTED.
Same as IIa.
 - B. PREEMPTING TRANSACTION HAS A HIGHER PRIORITY THAN TRANSACTION BEING PREEMPTED.

1. Preempted transaction is placed in multiple preempt pushdown list (FACILITY location F10).
2. If preempted transaction is in current event chain (not at a GATHER, MATCH or ASSEMBLE block), the preempt flag is set (Bit 7-T10). If at a GATHER, MATCH or ASSEMBLE block preempt status indicator (Bit 6-T10) is set. In both cases, the preempt count is incremented by one.
3. If preempted transaction is on the future event chain, the time remaining is calculated and placed in T4 (BDT) of transaction. The preempt status indicator (Bit6-T10) is set. Preempted transaction preempt count is incremented by one.
4. In all instances (1-4) the number of the preempting transaction is set in FACILITY location F2.

g.) SNA*n, PR, SNA*n

A. PREEMPTING TRANSACTION HAS A LOWER PRIORITY THAN TRANSACTION BEING PREEMPTED.

Same as Iia; C argument is ignored.

B. PREEMPTING TRANSACTION HAS A HIGHER PRIORITY THAN TRANSACTION BEING PREEMPTED.

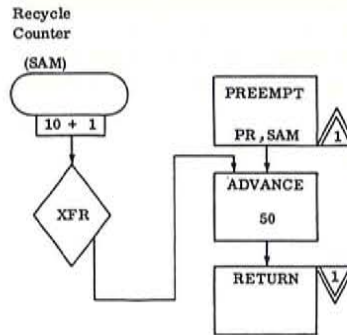
If preempted transaction is on the current event chain, same as Iif. 1, 2 and 4 above C argument is ignored.

If preempted transaction is on the future event chain it will be removed and placed on the current event chain scheduled to enter the block specified by the C argument. The preempt flag is set (Bit 7-T10). The preempted transaction will be placed in a multiple preempt pushdown list (FACILITY location F10). The preempt count of the preempted transaction will be incremented by 1. If the preempted transaction goes to the field C block. provision should be made for it to subsequently RETURN the FACILITY.

Example: A system where a complete recycle is necessary when an interruption occurs and a count of recycles is desired.

In the above example, assume transaction #2 with a priority of two has PREEMPTED Facility #1 and is in the ADVANCE block. Now transaction #5 with a priority of four enters the PREEMPT block. The following will occur:

- 1) Transaction #2 will be removed from



the future events chain and placed on the current events chain scheduled to enter block SAM.

- 2) The preempt flag, Bit 7-T10, of transaction #2 will be set.
- 3) The preempt count, T8, of transaction #2 will be incremented by one.
- 4) Transaction #2 will be placed in a pushdown list for Facility #1 (F10).
- 5) Transaction #5 will be placed in Facility #1 location F2.
- 6) Transaction #5 will then proceed to the ADVANCE block.
- 7) During the same scan of the current events chain, transaction #2 will enter the block SAM and increment Savevalue #10 by 1.
- 8) It would then enter the TRANSFER block and proceed to the ADVANCE block. When transaction #2 enters the ADVANCE block, the following will occur.
- 9) Since the preempt flag, Bit 7-T10, is set, the preempt status, Bit 6-T10, will be set.
- 10) The preempt flag, Bit 7-T10, will be reset.
- 11) Transaction #2 will then be removed from the current event chain and placed on the interrupt chain.
- 12) When transaction #5 subsequently leaves the ADVANCE block and enters the RETURN block the following will occur.
- 13) The first transaction on the pushdown chain will be obtained; in this case transaction #2.
- 14) Transaction #2 will be removed from the interrupt chain and placed on the

current events chain scheduled to enter the ADVANCE block. The preempt status bit of transaction #2, Bit 6-T10, will be reset. The preempt count of transaction #2 will be decremented by one.

Note: If the preempt count is not reduced to zero, the transaction will remain in the interrupt state.

h.) SNA*n,PR,SNA*n,,RE

A. PREEMPTING TRANSACTION HAS A LOWER PRIORITY THAN TRANSACTION BEING PREEMPTED.

Same as IIa; C and E arguments are ignored.

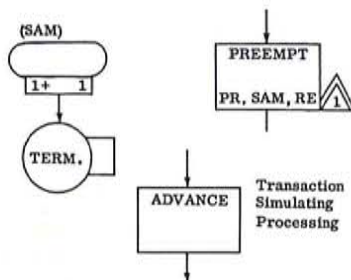
B. PREEMPTING TRANSACTION HAS A HIGHER PRIORITY THAN TRANSACTION BEING PREEMPTED.

If preempted transaction is on the current event chain; same as III. 1, 2 & 4. C and E arguments will be ignored.

Preempted transaction is on the future event chain.

- 1) Preempted transaction will be removed from the future event chain and placed on the current event chain scheduled to enter the block specified by the C argument.
- 2) All indications of the preempted transaction using the Facility will be removed. i. e., STEPS IIg, 2-4 will not be done.
- 3) The preempting transaction number will be placed in Facility location F2.
- 4) The preempting transaction will then proceed to the next sequential block.

Example: A system where an I/O error causes the complete termination of the job which issued the I/O command.



Assume in this example that transaction #10 with a priority of 10 is currently preempting Facility 1. At this time Transaction #127 enters the PREEMPT block with a priority of 127 to simulate an I/O error. Assuming further that Transaction 10 is in an ADVANCE block somewhere in the system, the following will occur:

1. Transaction 10 will be removed from the future events chain and placed in the current events chain scheduled to enter the SAVEVALUE block (SAM).
2. Transaction 10 will not be placed in any pushdown chain associated with Facility 1. The preempt count will not be incremented and neither the preempt status indicator nor the preempt flag will be set.
3. Transaction #127 will be placed in Facility location F2 (transaction preempting facility).
4. Transaction #127 will then proceed to a RETURN block to simulate the I/O recovery.
5. During the same scan of the current events chain, transaction #10 will enter the SAVEVALUE block and increment the count of I/O failures and then be removed from the system.

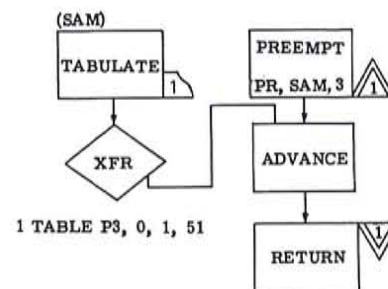
Note: When Transaction #127 RETURNS Facility 1 there will be no record of Transaction #10 being in contention for the Facility.

i.) SNA*n,PR,SNA*n,SNA*n

Similar to IIg as previously described with the following exception.

When the preempted transaction is removed from the future events chain, the time "left in the future event chain" is calculated and placed in the preempted transaction parameter specified by field D.

Example: In this system when a job is PREEMPTed the amount of time remaining before completion is of interest.



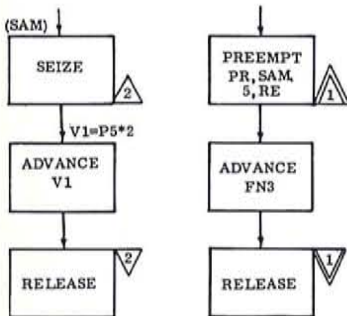
This example is very similar to Iig. above with the exception that the time the pre-empted transaction has remaining in the ADVANCE block is computed and placed in Parameter 3 before the transaction is sent to the tabulate block (SAM). This way a frequency distribution of the "remaining-time" will be obtained.

j.) SNA*n, PR, SNA*n, SNA*n, RE

Similar to Iih as previously described with the following exception:

When the preempted transaction is removed from the future event chain, the time "left in the future event chain" is calculated and placed in the preempted transaction parameter specified by field D.

Example: The example below might represent a portion of a job shop in which Facility 1 represents a high-speed machine. Facility 2 represents a similar machine which performs the same function as the first but is slower.



In this shop, if a job is using (PREEMPTing) Facility 1 when a higher priority job arrives, the first job is removed from the machine and finished on the slower machine. In this example, the remaining time on the high-speed machine is computed and placed in Parameter 5 of the PREEMPTed transaction. This value is then used to compute the ADVANCE time on the slower machine. As was the case in Iih. above, the PREEMPTed transaction is no longer in contention for Facility 1 and need not enter the RETURN block for Facility 1.

RETURN BLOCK

2	LOC	7	8	OPERATION	Facility Number 19 A
				RETURN	SNAj, SNA*n k, *n

The RETURN Block bears the same relation to the returning of a Facility that the RELEASE block bears to the normal releasing of a Facility. The RETURN block serves the function of removing from a Facility the transaction which has PREEMPTed it. The RETURNing operation is performed immediately upon entering the RETURN block. There is no further effect on the Facility if the transaction is subsequently prevented from entering the next sequential block following the RETURN block.

A transaction is never refused entry to a RETURN block. However, the Facility which is referred to by the field A argument of the RETURN block must have been PREEMPTed by the entering transaction. Otherwise, execution error 421 will occur: "Facility RETURNed by a Transaction not PREEMPTing it". No other transaction except the one that currently PREEMPTing a Facility is entitled to RETURN it.

If the Facility has also been SEIZED by another transaction, a variety of operations can be performed on the SEIZING transaction, depending on its current status.

Case 1. The SEIZING transaction has been PREEMPTed on more than one Facility

The preempt count of the SEIZING transaction is decremented by one to indicate that a Facility on which it has been PREEMPTed has been RETURNed. Since the preempt count is still greater than one, the SEIZING transaction remains in its current status until all preempts have been removed (preempt count equals zero).

Case 2. The SEIZING transaction is currently being PREEMPTed on only this one Facility, and is currently in an interrupt status in an ADVANCE block

Bit 7 of T9 is one, which indicates an interrupt status. The preempt count is first decremented to zero. The preempt status bit is, therefore, reset to zero to indicate that the transaction is no longer in a preempt status. The remaining time in the ADVANCE block has been stored in T4 block

departure time word of the SEIZEing transaction. The SEIZEing transaction is now merged into the future events chain according to the following departure time from the ADVANCE block (which is stored in word T4). The departure time is computed as follows:

$$\text{New ADVANCE Block Departure Time} = \text{Current Absolute Clock Time} + \text{Remaining Block Time (previously stored in word T4)}$$

Case 3. The SEIZEing transaction is currently being PREEMPTed on only this one Facility, and is in a matching condition in a MATCH, ASSEMBLE, or GATHER block

Bit 7 of T9 is one, which indicates an interrupt status. The preempt count is decremented to zero. The preempt status bit is reset to zero, which indicates that the transaction is no longer in a preempt status. However, the matching condition bit is still set to one. Consequently, the SEIZEing transaction remains in a matching status in its MATCH, ASSEMBLE, or GATHER block.

Case 4. The SEIZEing transaction is currently being PREEMPTed on only this one Facility and is still in the current events chain

Bit 6 of T9 is one, which indicates that the transaction is in the current events chain. The preempt count is decremented to zero. The preemption flag bit is reset to zero, so that the SEIZEing transaction can now enter ADVANCE, MATCH, ASSEMBLE, or GATHER blocks without being put into an interrupt status.

In case the Facility has also been PREEMPTed by another transaction the following would occur:

Case 5. The Facility is currently being PREEMPTed by other transactions of lower priority

Facility location F10 contains the address in COMMON of the list of transactions which were PREEMPTed by other transactions of a higher priority. The RETURN block subroutine obtains the last entry in this list (the number of the transaction with the next highest priority which was preempted by the RETURNing transaction) and sets this number in Facility location F2 as the PREEMPTing transaction. The preempt count of this transaction is

decremented by one and if the result is zero (only one preempt) the transaction is allowed to continue processing. If the preempt count is greater than zero, the transaction remains in a preempt status until all preempts are removed.

Cumulative Time Integral of Facility Utilization

When a transaction RETURNS a Facility, there may be a status change, i.e., the Facility changes from being in use to not in use. In many cases, however, the Facility may also be SEIZED, so that there is no status change. When a status change does occur, the following quantity is added to the cumulative time integral stored in word F3:

$$\text{Additional Clock Units in use} = \text{Current Absolute Clock Time} - \text{Time of last status change (word F4)}$$

Word F3, therefore, accumulates the total number of clock units during which the Facility was in use, i.e., SEIZED and/or PREEMPTed. The current absolute clock time is then stored in word F4 as the time of the last status change.

Status Change Flag and Reactivation of PREEMPT-GATE NI Delay Chain

When a transaction enters a RETURN block, there may be a PREEMPT-GATE NI delay chain waiting for the Facility not to be PREEMPTed (this is indicated by a transaction number in F8). If there are multiple preempts on this Facility at this time, this is the only delay chain which is reactivated. A SEIZE-GATE NU delay chain (indicated by a transaction number in F6) may also have been formed waiting for the Facility not to be in use. If this is the only preempt, each transaction on both of these delay chains is reactivated, i.e., their scan status indicators are reset to zero. The status change flag is automatically set to "on" whenever a transaction enters a RETURN block.

Consequently, the overall GPSS/360 scan will automatically transfer back to the start of the current events chain when it finishes moving the transaction which entered the RETURN block. This ensures that all of the reactivated transactions in former PREEMPT-GATE NI and/or SEIZE-GATE NU delay chains will be processed by the overall GPSS/360 scan. All, none, or just some of the delayed transactions (including those in TRANSFER BOTH or ALL blocks) may actually be able to enter the SEIZE, PREEMPT, GATE NU, or GATE NI blocks.

Observe that the relative position of the delayed transactions in the current events chain is critical in determining which transactions actually succeed in moving into their next blocks. For example, assume that a transaction RETURNS a Facility which is currently not being SEIZED. Assume further that there are transactions waiting to SEIZE and to PREEMPT the Facility. Depending on their relative position in the current events chain, it is possible for higher priority transactions to SEIZE the Facility. A lower priority transaction will then PREEMPT the Facility in the same instant of time, and probably put the SEIZING transaction into an interrupt status.

A SEIZEING OR PREEMPTING TRANSACTION ATTEMPTS TO SEIZE OR PREEMPT THE SAME FACILITY

What happens if a transaction which has SEIZED or PREEMPTed a Facility subsequently attempts to enter a SEIZE or PREEMPT block which references the same Facility? There are four possible situations:

Case 1. A SEIZEing transaction attempts to SEIZE the same Facility

The SEIZE block subroutine simply determines that the Facility is already in use (F1 is nonzero) and, therefore, denies entry to the SEIZING transaction. The transaction can, therefore, be hung up indefinitely, unless it is in a TRANSFER block with a BOTH or ALL selection mode that might allow exit by another block.

Case 2. A PREEMPTing transaction attempts to PREEMPT the same Facility

This is similar to Case 1, i.e., the PREEMPT block subroutine simply determines that the Facility is already being PREEMPTed (F2 contains a transaction number) and, therefore, denies entry to the PREEMPTing transaction.

Case 3. A PREEMPTing transaction attempts to SEIZE the same Facility

This is also similar to Case 1, i.e., the SEIZE block simply determines that the Facility is already in use (F2 is nonzero) and, therefore, denies entry to the PREEMPTing transaction.

Case 4. A SEIZING transaction attempts to PREEMPT the same Facility

This is the most interesting case, since the PREEMPT block subroutine determines that the SEIZING transaction is in the current events chain. This is obvious since the SEIZING transaction is

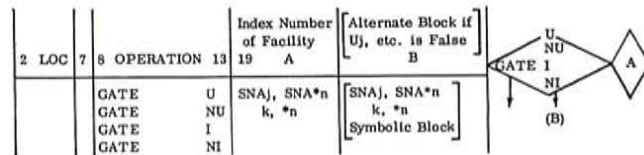
the same transaction which is attempting to move into the PREEMPT block. The operations that are performed are the same as if the SEIZEing and PREEMPTing transactions were different (see Case 3D in "PREEMPT Block" earlier in this chapter).

1. The preempt count (T8) is incremented by one.

2. The preempt flag is set to one to indicate that the transaction is to be put into an interrupt status when it next enters an ADVANCE, MATCH, ASSEMBLE or GATHER block.

A race against time now begins. If the transaction succeeds in entering a RELEASE or RETURN block, the preempt count is decremented by one, and the preempt flag is reset to zero. On the other hand, the transaction goes into a permanent limbo if it enters an ADVANCE, MATCH, ASSEMBLE, or GATHER block before it can get to a RELEASE or RETURN block. Any one of the first four blocks mentioned above detects that the preempt flag has been set to one and, therefore, removes the transaction from the current events chain and puts it into an interrupt status. The transaction is dead now since it can never enter a RETURN block which will remove it from an interrupt status and return it to the current events chain.

GATE U, GATE NU, GATE I AND GATE NI BLOCKS



If a field B alternate block is specified, the above GATE blocks operate in an unconditional entry mode, i.e., transactions can always enter the GATE block. If the Standard Logical Attribute is true, the transaction moves to the next sequential block following the GATE block. If Uj, NUj, Ij, or NIj is false, the transaction moves to the alternate block which is specified in field B of the GATE block.

If a field B alternate block is not specified, the GATE block operates in a conditional entry mode, i.e., transactions can enter the GATE block only if the Standard Logical Attribute is true. If Uj, NUj, Ij, or NIj is false, the transaction is placed in a pushdown delay chain and deactivated from the overall GPSS/360 scan (the usual exceptions are transactions in TRANSFER BOTH or ALL blocks).

Other transactions which pass through the following blocks can make the indicated Standard Logical Attributes true:

Standard Logical Attribute	Block(s) Which Can Change the Value of the Standard Logical Attribute	
	Make True	Make False
Uj	SEIZE, PREEMPT	RELEASE, RETURN
NUj	RELEASE, RETURN	SEIZE, PREEMPT
Ij	PREEMPT	RETURN
NIj	RETURN	PREEMPT

All transactions in the delay chains that are associated with the above Standard Logical Attributes will be reactivated by the appropriate block-type subroutine. The overall GPSS/360 scan may then be able to advance one or more of these transactions (as well as those in TRANSFER BOTH or ALL blocks) into the conditional entry GATE U, GATE NU, GATE I, and GATE NI blocks.

STATISTICAL PRINTOUT

The standard Facility statistical printout shown in Table 19 is written on the output device, under the following three conditions:

1. A simulation run is terminated normally after the termination count (which is specified in field A of the latest START card) is decremented to zero or less.
2. A simulation run is terminated by one of the running error stops described in Appendix A.
3. A transaction enters the following type of PRINT block:

2	LOC	7	8	OPERATION	19	A	B	C
				PRINT		k_L	k_U	F

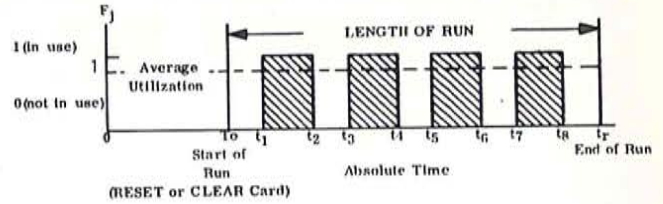
As described in "PRINT Block" in Chapter 14, the standard statistics for the Facilities numbered from the field A lower limit (k_L) up through the field B upper limit (k_U) are printed out. In all three cases statistics are printed out for only those Facilities which have had one or more entries (as recorded in word F9).

Average Utilization

The average utilization of each Facility printed at the end of each simulation run is equal to:

$$\text{Average Utilization} = \frac{\text{Cumulative time integral}}{\text{Relative clock time (C1) since last RESET or CLEAR card}}$$

The cumulative time integral can be portrayed graphically as the shaded area under the following usage profile:



The odd = number status change block times (t_1, t_3, \dots) represent times when the Facility changed from being not in use to in use, i.e., SEIZED and/or PREEMPTed. The even-number status change clock times (t_2, t_4, \dots) represent times when the Facility changed from being in use to not in use, i.e., by being RELEASED and/or RETURNed.

The quantities $(t_2 - t_1), (t_4 - t_3), \dots$ which are added to word F3, represent the total number of clock units during which the Facility was in use. The average utilization is therefore the average height of the above usage profile (shown by the dotted line), which is simply the number of clock units in use (word F3) divided by the relative clock time which has elapsed since the last RESET or CLEAR card.

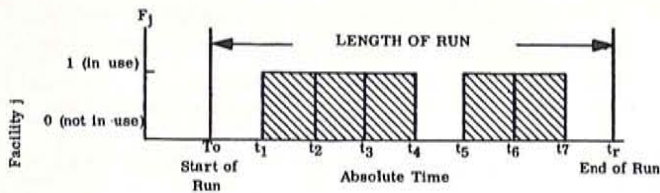
If RESET or CLEAR cards are not read between multiple STARTs, the cumulative time integral builds up the total number of clock units in use since the last RESET or CLEAR card. Consider the following series of three simulation runs, each of which is 1000 clock units long.

Run Number	Clock Units In Use During Run	Average Utilization (With RESET/CLEAR)	Cumulative Clock Units	Average Utilization (Without RESET/CLEAR)
1	536	.536	536	.536
2	600	.600	1136	.568
3	340	.340	1476	.492

If RESET or CLEAR cards are read between STARTs the three average utilization values (.536, .600, and .340) are the utilizations only during each run. If no RESET or CLEAR cards are read the average utilization values are the cumulative utilizations for the first run (.536), for the duration of the first two runs (.568), and for the duration of the first three runs (.492).

There is a statistical advantage in using RESET cards, since the three individual average utilizations (.536, .600, and .340) give a measure of the sampling variability around the grand average value of .492.

The usage profile more typically appears as shown below:



At time t_1 , Transaction 1 SEIZES Facility j . At t_2 , Transaction 2 PREEMPTS the Facility, and at t_3 , it RETURNS the Facility to Transaction 1, which then RELEASES the Facility at time t_4 . At time t_5 , Transaction 3 SEIZES Facility j and RELEASES it at time t_6 . Transaction 4 immediately SEIZES the Facility at the same time t_6 and RELEASES it at time t_7 .

Word F9 now contains an entry count of four at the end of the simulation run. The average time per transaction is then the number of clock units in use (word F3) divided by the number of entries (word F9). The average time per transaction could be considered as the average horizontal time distance spent by a transaction in the above usage profile.

Observe that no distinction is made between SEIZEing or PREEMPTing transactions in computing the average time per transactions.

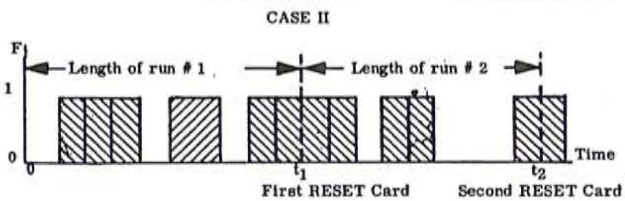
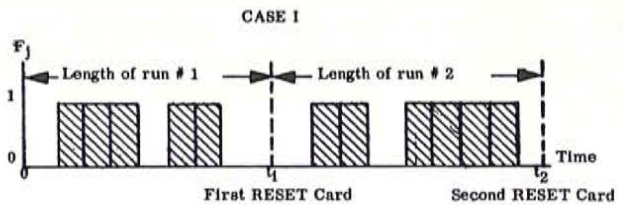
EFFECT OF RESET AND CLEAR CARDS

Table 20 summarizes the effects of RESET and CLEAR cards on the attributes of Facility entities. A RESET card zeros the cumulative time integral (word F3) and sets the clock time of the last status (word F4) equal to the current absolute clock time. The entry count (word F9) is then set equal to:

1. Zero, if the Facility is neither being SEIZED nor PREEMPTed.
2. One, if the Facility is either being SEIZED or PREEMPTed at the start of the new run.
3. The number of transactions currently PREEMPTing the Facility, if there are multiple PREEMPTs.
4. The number of transactions currently PREEMPTing the Facility plus 1, if the Facility is being SEIZED and there are multiple PREEMPTs.

The above entry count adjustments can have a significant effect on the computed average time per transaction. Consider the following two cases:

In Case I, the Facility is not in use at times t_1 and t_2 when RESET cards are read. The true average time per transaction is therefore computed and printed out. However, in Case II, the Facility is being both SEIZED and PREEMPTed at time t_1 . Consequently, the F6 entry count for run 2 is initially set equal to 2, even though the two trans-



actions were already counted in the entry count for run 1. At time t_2 in Case II, the Facility is being SEIZED when the second RESET card is processed. Consequently, the entry count for run 3 is initially set equal to 1, even though the SEIZEing transaction was already counted in the entry count for run 2.

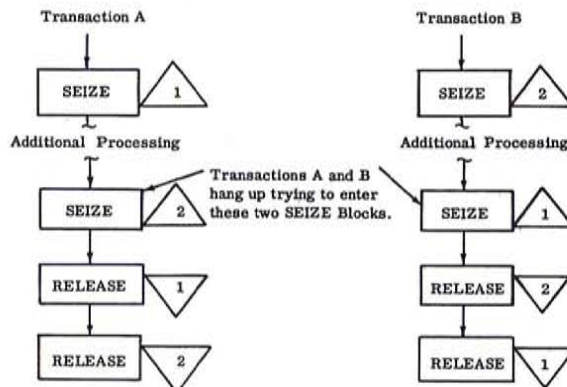
The net result is that the computed average time per transaction for both runs 1 and 2 in Case II is less than the true long-run average time per transaction. The following general rule can be stated for all Facilities:

The computed average time per transaction on each Facility is always less than or equal to the true long-run average time per transaction. This depends on whether the true number of entries has been overstated because of RESET card operations.

EXAMPLES OF FACILITY BLOCK TYPES

Example 1:

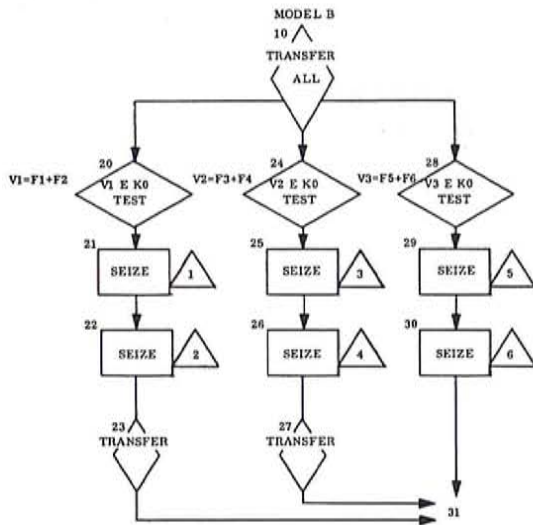
The analyst must be careful to avoid the following classic interlock when two transactions each SEIZE Facilities and then attempt to SEIZE the Facility that the other Transaction has SEIZED.



Neither Transaction A nor B can enter the second two SEIZE blocks and, therefore, neither can manage to RELEASE the Facility which it originally SEIZED.

Example 2:

In model B below, transactions will SEIZE Facilities 1 and 2, Facilities 3 and 4, or Facilities 5 and 6 whenever one pair of Facilities is not in use and can be SEIZED concurrently. The three variable statements 1, 2, and 3 are zero whenever the particular pair of Facilities is not in use, i.e., whenever Standard Numerical Attributes F_j and $F_{(j + 1)}$ are both zero.



Example 3:

Model C shows how a SEIZEing transaction can avoid being PREEMPTed on a Facility which it has SEIZED. The SEIZEing transaction will spend a fixed amount of time (T) before RELEASEing the Facility, whether or not the Facility might be PREEMPTed by another transaction.

The SEIZEing transaction subverts its preemption by the following methods:

1. Logic Switch 1 is put onto a set condition.
2. An offspring transaction is SPLIT off to spend the fixed time (T) in an ADVANCE block.

3. The SEIZEing transaction hangs up on a GATE LR block until the offspring transaction puts Logic Switch 1 back into a reset condition.

4. The SEIZEing transaction the RELEASES Facility 1 after the fixed time, T.

The time between the SEIZEing and RELEASEing of the Facility is always the fixed time T. Consider what happens when another transaction PREEMPTs Facility 1. Because the SEIZEing transaction is still in the current events chain and is blocked from entering the GATE LR block, it will not be PREEMPTed. Instead, the preemption flag (bit 6 of byte T9) is set to one to indicate that the transaction is to be put into an interrupt status when it next enters an ADVANCE, MATCH, ASSEMBLE, or GATHER block.

The SEIZEing transaction, of course, never reaches such a block. Instead, when its offspring transaction puts Logic Switch 1 back into a reset condition, the SEIZEing transaction succeeds in entering the GATE LR block and then RELEASEs Facility 1 after the fixed time, T.

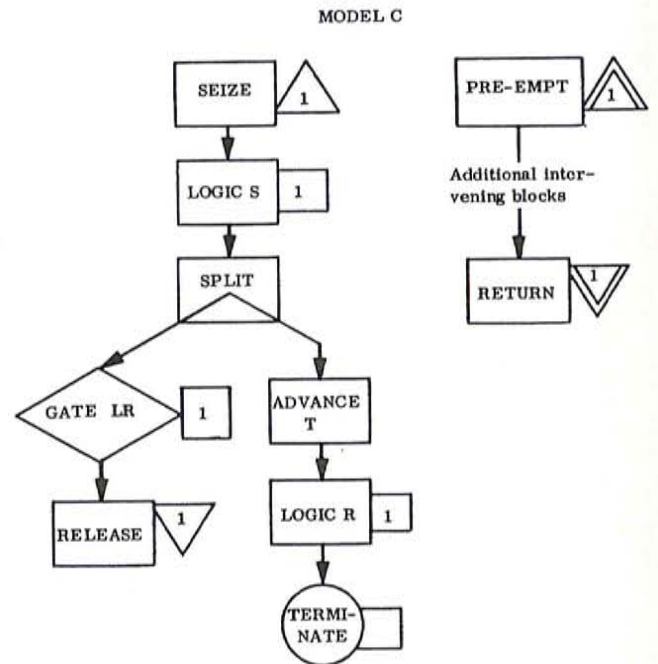


TABLE 19: STATISTICAL PRINTOUT FOR FACILITY ENTITIES

Facility	Average Utilization	Number Entries	Average Time/Trans.	SEIZEing Trans. No.	PREEMPTing Trans. No.
2	.131	9	5833.113	81	7
10	.309	8	15449.635	5	53
15	.273	6	18245.333		

<p>Facility Name or Number</p> <p>↑</p>	<p>$\frac{\text{Cumulative Time Integral (Word F3)}}{\text{Relative Clock Time Since Last RESET or CLEAR Card}}$</p> <p>↑</p>	<p>(Word F9)</p> <p>↑</p>	<p>$\frac{\text{Cumulative Time Integral (Word F3)}}{\text{Number of Entries (Word F9)}}$</p> <p>↑</p>	<p>Number of SEIZEing Transaction (F1)</p> <p>↑</p>	<p>Number of PREEMPTing Transaction (F2)</p> <p>↑</p>
---	--	---------------------------	---	---	---

TABLE 20: EFFECT OF RESET AND CLEAR CARDS ON FACILITY ATTRIBUTES

RESET Card			
Word	Length	Attribute Value Before RESET Card	Result of RESET Card on Attribute Value
F1	2 Bytes	Index number j of the SEIZEing Transaction (if any)	Unchanged
F2	2 Bytes	Index number j of the PREEMPTing Transaction (if any)	Unchanged
F3	4 Bytes	Cumulative time integral of Facility usage	Set to zero
F4	4 Bytes	Clock time of last status change	Set to current value of absolute clock
F5, F6, F7, F8	2 Bytes for each	Origins of four different pushdown delay chains	Unchanged
F9	4 Bytes	Entry count	<ol style="list-style-type: none"> 1. Set to zero if Facility is neither being SEIZED nor PREEMPTed 2. Set to one if Facility is either being SEIZED or PREEMPTed 3. Set to the number of transactions currently preempting if there are multiple preempts. 4. Set to the number of transactions currently PREEMPTing plus one, if Facility is being SEIZED and there are multiple PREEMPTs.
F10	4 Bytes	Address of multiple-preempt list	Unchanged

CLEAR Card

F1, F2, F3, F4, F5, F6, F7, F8, and F9 are all set equal to zero. If there has been a multiple preempt, the core for the pushdown list is returned to COMMON and the address in F10 is zeroed.

GENERAL NATURE

Storage entities are used in system simulations to represent multiple-capacity equipment, e.g., the core storage in computers and message exchanges. Such equipment is generally capable of simultaneously handling one or more transactions up to some finite capacity limit. The standard GPSS / 360 program for a 128k machine has core allocated for 150 storages, numbered 1, 2, . . . 150. Execution Error 499 will occur if a storage number greater than the quantity allocated is referenced. Table 21 describes the attributes that are stored in the ten 360 words required for each storage. These words accumulate various statistics on the usage of storages, which are printed as part of the standard GPSS/360 output, as shown in Table 22. Key storage statistics include:

1. the average contents of the storage;
2. the average utilization of the storage capacity, which is equal to the average storage contents divided by the storage capacity;
3. the total number of entries into the storage, i.e., into ENTER blocks;
4. the average time that each entry spends in the storage, i.e., between ENTERing and LEAVING the storage;
5. the current storage contents, at the end of the simulation run;
6. The maximum contents observed since the last RESET or CLEAR card.

Two block types, ENTER and LEAVE, are associated with storage entities. Four types of GATE blocks are also associated with storages: GATE SE, GATE SNE, GATE SF, and GATE SNF.

STANDARD NUMERICAL ATTRIBUTES

Each storage entity has seven Standard Numerical Attributes:

1. S_j = current contents of Storage j
 2. R_j = remaining available capacity of Storage j
- At any time $S_j + R_j =$ capacity of Storage j .
3. SR_j = utilization of Storage j in parts per thousand, i.e., if the utilization was .65 the computed value would be 650.
 4. SA_j = average contents of Storage j (truncated to an integer, i.e., $1.73 = 1$).
 5. SM_j = maximum contents of Storage j
 6. SC_j = number of entries for Storage j
 7. ST_j = average time each transaction used Storage j (truncated)

The capacity of each Storage may be defined by the user before the start of a simulation run. The Storage definition card is provided for this purpose. There are two formats for this card the use of which depends on the number of Storages which the user wishes to define.

The first format is that used in GPSS III. The location field contains the Storage number, the operation field contains STORAGE, and the operand field specifies the capacity (which must be less than or equal to $2^{31}-1$ or 2, 147, 483, 647).

Examples:

1	STORAGE	100
6	STORAGE	237485

The second format provides the user with the ability to define multiple Storages with one Storage definition card. The location field is blank, the operation field contains STORAGE, and the operand field specifies the Storages and/or range of Storages and their capacities using the following rules:

1. The Storage number must be preceded by the letter S, i.e., S1, S6, S10, etc.
 2. The Storage number or range of Storages must be separated from the capacity value by a comma, i.e., S1, 100 or S6, 237485.
 3. A range of Storages, which will be assigned the same capacity, must be separated by a dash (-), i.e., S1-S10, 100.
 4. Subsequent entries on the card must be separated by a slash (/), i.e., S1, 10/S2, 20.
- Examples:

STORAGE	S1,100
STORAGE	S1,100/S2,200
STORAGE	S1-S10,100/S11-S20,200

Each Storage definition card replaces the previous one, and the contents of the storage are unchanged by the definition. Any Storage of undefined capacity which is referenced in a model is assumed to have a maximum capacity, i.e., each storage is initialized to have $R_j = 2^{31}-1$ or 2, 147, 483, 647 and $S_j = 0$. (For further details see "Re-definition of Storage Capacity with Storage Card" later in this chapter.)

STANDARD LOGICAL ATTRIBUTES

Each storage entity has four Standard Logical

Attributes with two possible values (true and false).

1. SE_j is true if Storage j is empty, i. e., S_j = 0 and R_j = capacity
2. SNE_j is true if Storage j is not empty, i. e., S_j > 0 and R_j < capacity
3. SF_j is true if storage j is full, i. e., S_j = capacity and R_j = 0
4. SNF_j is true if Storage j is not full, i. e., S_j < capacity and R_j > 0

These Standard Logical Attributes are used in GATE SE, GATE SNE, GATE SF, and GATE SNF blocks, as described earlier, to control the flow of transactions.

When a Storage is defined to have a unit capacity, the following Standard Logical Attributes of the storage are equivalent:

Equipment Storage Standard Logical Attributes	Analogous Facility Standard Logical Attributes
SNE _j = SF _j	U
SNF _j = SE _j	NU

When the unit capacity Storage is not empty it is also full. When it is not full, it is also empty. A unit capacity storage is analogous in many ways to a facility, so that the Facility Standard Logical Attributes U_j and NU_j are analogous to the storage logical attributes shown above.

INTERNAL NONADDRESSABLE ATTRIBUTES

Table 20 describes the various internal attributes stored in the twelve storage locations (S1, S2, . . . , S12).

Those which are nonaddressable by the user include:

1. The clock time of the last status change (word S5), i. e., when the storage contents increased (ENTER block) or decreased (LEAVE block).
2. A 64-bit cumulative time integral of the Storage contents (words S3 and S4).
3. The number of the first transaction in each of the five pushdown delay chains associated with each Storage (S8, S9, S10, S11, S12).

ENTER BLOCK

2	LOC	7	8	OPERATION	Index Number of Storage	Number of Units To Enter Storage
				ENTER	19 A	B
					SNA _j , SNA* _n k, * _n	[SNA _j , SNA* _n] k, * _n

ENTER	A, B
-------	------

Examples:

```

ENTER 5
ENTER *8, P6
ENTER FN3, 100
ENTER 1, R1           Fills up Storage
ENTER FN*11
    
```

A transaction is not permitted to enter a Storage that is already full, or which does not have enough available space to accommodate the transaction. The ENTER block specifies a Storage number in field A, and the number of storage units which are to be occupied in field B. If field B is blank the number of Storage units to be occupied is assumed to be one. When a transaction attempts to enter an ENTER block the GPSS/360 Program compares the field B requested number of units with R_j, the space remaining in the Storage. If the field B number of units is less than or equal to the available space R_j, the steps in "Transaction Succeeds in Moving into ENTER Block" below, will be executed immediately. If the value of the field B argument is zero, the transaction will always succeed in entering the ENTER block but will not contribute to the Storage contents. The same transaction can ENTER an unlimited number of Storages and can, subsequently, LEAVE all or some of these Storages.

Transaction Cannot Move Into ENTER Block

The transaction will be delayed in the block preceding the ENTER block and will be placed in a pushdown delay chain as described in "LEAVE Block" later in this chapter. The blocked transaction may, however, be diverted if the preceding block was a TRANSFER block with a BOTH or ALL selection mode.

Transactions which move into an ENTER block attempt to move to the next sequential block.

When successive transactions attempt to ENTER the same storage with different field B quantities, it is possible for a later transaction to pass an earlier transaction. This would occur if an earlier transaction attempts to ENTER with a field B quantity which is greater than the remaining capacity, R_j, while a later transaction succeeds in ENTERING the Storage with a field B quantity which is less than the remaining capacity, R_j. Observe that if the field B argument of an ENTER block is one, then transactions will be refused entry to the ENTRY block only if the referenced Storage is full (a blank field B in an ENTER block is interpreted as one).

Transaction Succeeds in Moving Into ENTER Block

The current Storage contents, S_j (which are stored

in S1) is incremented by the number of field B units. The remaining Storage capacity, Rj (S2) is concurrently decremented by the same quantity. The new contents, Sj, are compared against the maximum contents (stored in word S7). If the new Storage contents are greater, this value is entered in word S7 as the new maximum contents. The entry count into the Storage (word S6) is incremented by the field B number of units, which can be one or more. Therefore, the number of transactions entering the Storage will always be less than or equal to the S6 entry count.

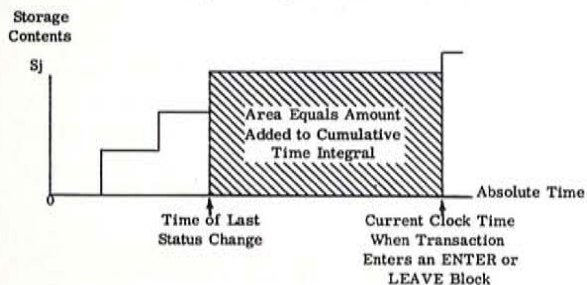
Observe that no record is kept on which transactions ENTER a Storage, as opposed to facilities which record the numbers of the transactions that are SEIZEing and PREEMPTing the facility. Therefore, a transaction may LEAVE a Storage which it had not previously ENTERed. A transaction need not remove the same number of units (in a LEAVE block) that it added to the Storage (in an ENTER block). Ultimately, the additions and removals of all transactions must balance out. Otherwise, the Storage will fill up if the additions (ENTERs) exceed the removals (LEAVEs); or Execution Error 425, "Transaction LEAVEing by More Than Storage Contents", will occur if the removals exceed the additions. In cases where the analyst cannot be sure that the same transaction that SEIZED a facility will eventually RELEASE it, he can instead use a unit capacity Storage. Different transactions can now ENTER (seize) and LEAVE (release) the storage.

The ENTER block subroutine also computes the length of time that the Storage was at its previous contents:

$$\text{Length of time} = \text{Current absolute clock time} - \text{Time of last status change (word S5)}$$

The product of this time interval, and the previous contents, is added to the 64-bit cumulative time integral (which is maintained in double word S3-S4). The current absolute clock time is then stored in word S5 as the time of the last status change.

The cumulative time integral of the Storage contents can be portrayed as follows:



At the end of each simulation run the cumulative time integral is divided by the value of the relative clock (C1) since the last RESET or CLEAR Card, to obtain the average contents of the storage. The time integral is also divided by the entry count (word S6) to obtain the average time per entry into the storage.

Status Change Flag and Reactivation of GATE SF and/or GATE SNE Delay Chains

When a transaction succeeds in moving into an ENTER block there may be:

1. A GATE SNE delay chain of transactions waiting for the Storage to be not empty, in order to enter a GATE SNE block.
2. A GATE SF delay chain of transactions waiting for the Storage to be full, in order to enter a GATE SF block.

(The existence of the above delay chains is indicated by the presence of transaction numbers in S11 and S8 respectively.) If a transaction ENTERs an empty Storage, each transaction in a possible GATE SNE delay chain is reactivated; i.e., their scan status indicators are reset to zero. The status change flag is also automatically set on. If the transaction ENTERs and fills the Storage, then each transaction in a possible GATE SF delay is likewise reactivated. The status change flag is also automatically set on.

Because the status change flag is set on, the overall GPSS/360 scan will automatically transfer back to the start of the current events chain when it finishes moving the transaction that ENTERed the particular Storage. This ensures that all of the reactivated transactions in the former GATE SNE and GATE SF delay chains will be processed by the overall GPSS/360 scan. All, none, or just some of the delayed transactions (including those in TRANSFER BOTH or ALL blocks) may actually be able to enter the GATE SNE and GATE SF blocks.

LEAVE BLOCK

2	LOC	7	8	OPERATION	Index Number of Storage A	Number of Units To Leave Storage B	LEAVE	A, B
				LEAVE	SNAj, SNA*n k, *n	[SNAj, SNA*n] k, *n		

Examples:

LEAVE	5	
LEAVE	1, S1	Empty Out Storage
LEAVE	6, FN*11	
LEAVE	V3, V4	

The LEAVE block is used to make available some previously occupied contents in a Storage entity. A transaction is always permitted to enter a LEAVE block, but a restriction is made on the number of Storage units which may be made available. The LEAVE block specifies the storage number in field A, and the number of units made available in field B. If field B is blank, one unit is removed from the Storage contents. Transactions proceed to the next sequential block following the LEAVE block. Execution Error 425, "Transaction LEAVEing by more than storage contents", will occur if the number of units to be removed from the Storage (the value of the field B argument, or 1 if the field B is blank) is greater than the current contents of the Storage (Sj).

The current Storage contents, Sj (S1), is decremented by the number of field B units. The remaining Storage capacity, Rj (S2), is concurrently incremented by the same quantity. The entry count (word S6) is not affected by the LEAVE block.

The LEAVE block subroutine (just like the ENTER block) computes the length of time that the Storage was at its previous contents:

$$\text{Length of time} = \text{Current absolute clock time} - \text{Time of last status change (word S5)}$$

The product of this time interval, and the previous contents, is added to the 64-bit cumulative time integral. The current absolute clock time is then stored in word S5 as the time of the last status change.

Status Change Flag and Reactivation of GATE SNF, GATE SE, and ENTER Delay Chains

When a transaction enters a LEAVE block the following three pushdown delay chains of transactions may be associated with the referenced Storage:

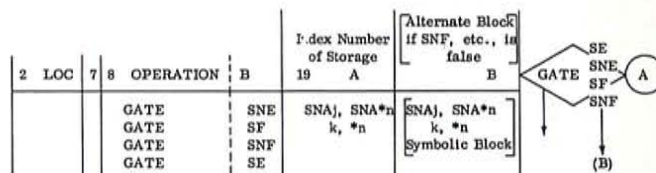
1. GATE SNF delay chain of transactions waiting for the Storage to be not full, in order to enter a GATE SNF block
2. GATE SE delay chain of transactions waiting for the Storage to be empty, in order to enter a GATE SE block
3. ENTER delay chain of transactions waiting to ENTER the Storage whenever the current Storage contents are decremented by transactions entering LEAVE blocks

The existence of the above delay chains is indicated by the presence of transaction numbers in the halfword Storage locations S9, S10 and S12 respectively. If a transaction LEAVEs a full Storage, each transaction in a possible GATE SNF delay chain is reactivated; i.e., their scan status

indicators are reset to zero. Similarly, each transaction in a possible ENTER delay chain is also reactivated. If the transaction at the LEAVE block reduces the Storage contents to zero, each transaction in a possible GATE SE delay chain is also reactivated.

In all three cases, the status change flag is set to on, so that the overall GPSS/360 scan will automatically transfer back to the start of the current events chain when it finishes moving the transaction that left the particular Storage. This ensures that all of the reactivated transactions in the former GATE SNF, GATE SE, and ENTER delay chains will be processed by the overall GPSS/360 scan. All, none, or just some of the delayed transactions (including those in TRANSFER BOTH or ALL blocks) may actually be able to enter the GATE SNF, GATE SE, and ENTER blocks.

GATE SNE, GATE SF, GATE SNF, AND GATE SE BLOCKS



If a field B alternate block is specified, the above GATE blocks operate in an unconditional entry mode; i.e., transactions can always enter the GATE blocks. If the Standard Logical Attribute is true, the transaction moves to the next sequential block following the GATE block. If it is false, the transaction moves to the field B alternate block.

If a field B alternate block is not specified, the GATE block operates in a conditional entry mode; i.e., transactions can enter the GATE block only if the Standard Logical Attribute is true. If SNEj, SFj, SNFj, or SEj is false, the transaction is placed in a pushdown delay chain and deactivated from the overall GPSS/360 scan. The usual exceptions are transactions in TRANSFER BOTH or ALL blocks which are not deactivated in pushdown delay chains.

Other transactions, which pass through the following blocks, can make the indicated Standard Logical Attributes true:

Standard Logical Attribute	Block(s) Which Can Change the value of the Standard Logical Attribute	
	Make True	Make False
SNEj	ENTER	LEAVE
SFj	ENTER	LEAVE

Standard Logical Attribute	Block(s) Which Can Change the value of the Standard Logical Attribute	
	Make True	Make False
SNFj	LEAVE	ENTER
SEj	LEAVE	ENTER

All transactions in the delay chains associated with the above Standard Logical Attributes will be reactivated by the appropriate block-type subroutines. The overall GPSS/360 scan may then be able to advance one or more of these transactions (as well as those in TRANSFER BOTH or ALL blocks) into the conditional entry GATE SNE, GATE SF, GATE SNF, and GATE SE blocks.

STATISTICAL PRINTOUT

The standard storage statistical printout shown in Table 22 is written on the output device under the following three conditions:

1. A simulation run is terminated normally after the termination count (specified in field A of the latest START card) is decremented to zero or less.
2. A simulation run is terminated by one of the execution errors described in Appendix A.
3. A transaction enters the following type of PRINT block:

```
PRINT ki, ku, S
```

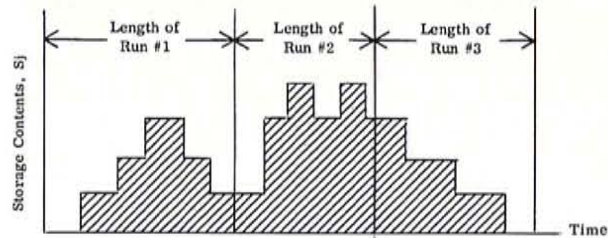
As described in "PRINT Block" in Chapter 14, the standard statistics for the storages numbered from the field A lower limit (k_i) up through the field B upper limit (k_u) are printed. In all three cases, statistics are printed only for those storages which have had one or more entries (as recorded in word S6).

Average Contents of Storage

The average contents of each Storage, which is printed at the end of each simulation run, is equal to:

$$\text{Average Contents} = \frac{\text{Cumulative time integral of contents}}{\text{Relative Clock Time (C1) since last RESET or CLEAR Card}}$$

The 64-bit cumulative integral is simply the area under the following storage content profile:



The average contents are simply the average height of the above profile.

Average Utilization of Storage Capacity

The average utilization of the capacity of each Storage, printed at the end of each simulation run, is equal to:

$$\text{Average Utilization} = \frac{\text{Average Contents}}{\text{Storage Capacity}} = \frac{\text{Cumulative Time Integral}}{\text{Relative Clock Time (C1) x Capacity}}$$

The Storage capacity is computed at any time as the sum of the current contents (S1) and the remaining capacity (S2).

RESETEing and CLEARing of Cumulative Time Integral

If RESET or CLEAR cards are not read between multiple STARTs, the 64-bit cumulative time integral continues to build up the area under the Storage contents profile.

Consider the following series of three simulation runs, each 1000 clock units long, where the Storage has a capacity of 200 units:

Run No.	Time Integral During Run	With RESET/CLEAR Cards		Without RESET/CLEAR Cards	
		Avg. Contents	Avg. Utiliz.	Avg. Contents	Avg. Utiliz.
1	7840	78.40	.394	78.40	.394
2	6930	69.30	.347	72.39	.369
3	5482	54.82	.274	67.51	.338

If RESET or CLEAR cards are read between multiple STARTs, the three average contents values (78.40, 69.30, and 54.82) are the statistics only during each run. If no RESET or CLEAR cards are read, the average contents values are the cumulative contents for the first run (78.40), for the duration of the first two runs (72.39), and for the duration of the first three runs (67.51). The same reasoning applies to the three individual

run average utilizations (.394, .347, .274) versus the three cumulative average utilizations (.394, .369, .338). There is a statistical advantage in using the RESET cards since the three individual average contents (78.40, 69.30, 54.82) give a measure of the sampling variability around the grand average value of 67.51.

Average Time per Transaction

The average time per transaction in the Storage is computed as follows:

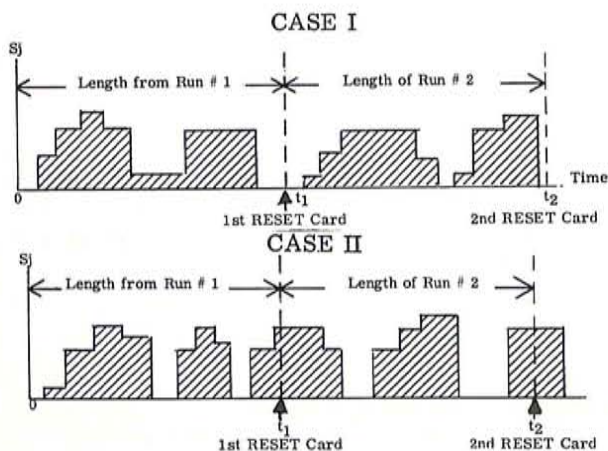
$$\text{Average Time} = \frac{\text{Cumulative Time Integral}}{\text{per Transaction Number of Entries (Word S6)}}$$

This statistic is really the average time per unit entered into the storage, since the field B argument of the ENTER block determines the number of units entered into the storage by each transaction. Word S6 accumulates the number of units, and not necessarily the number of transactions that entered ENTER blocks which reference the storage. The total number of units and the number of transactions are equal only when each transaction contributes a unit entry, e.g., when field B of every ENTER block is blank or specifies one.

EFFECT OF RESET AND CLEAR CARDS ON STORAGE STATISTICS

Table 23 summarizes the effects of RESET and CLEAR cards on the attributes of storage entities. A RESET card zeroes the 64-bit cumulative time integral and sets the clock time of the last status change equal to the current absolute clock time. The entry count is then set equal to the Storage contents (Sj) whose value will be in the range between zero and the Storage capacity.

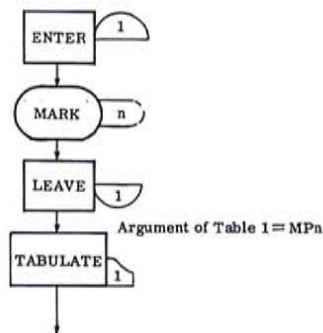
The above entry count adjustments can have a significant effect on the computed average time per transaction. Consider the following two cases:



In Case I, the Storage is empty at times t_1 and t_2 when RESET cards are read. The true average time per transaction is, therefore, computed and printed out. However, in Case II, the storage is not empty at time t_1 , i.e., $S_j > 0$. Consequently, the S6 entry count for run 2 is initially set equal to the storage contents at time t_1 , even though these units were already counted in the entry count for run 1. At time t_2 in Case II, the Storage contents are again greater than zero when the second RESET card is read. Consequently, the entry count for run 3 is initially set equal to the Storage contents at time t_2 , even though these units were already counted in the entry count for run 2.

The net result is that the computed average time per transaction for both runs 1 and 2 in Case II is less than the true long-run average time per transaction. The following general rule can be stated for all storages: The computed average time per transaction in each Storage is always less than or equal to the true long-run average time per transaction. This depends on whether the true number of entries has been overstated because of RESET card operations.

The only way in which the true long-run average time per Storage entity can be definitely obtained is with a model of the following type:



The transit time of each transaction is TABULATED in table 1, whose argument is MPn (the transaction transit time in Parameter n). The transaction transit time is obtained by first MARKING the absolute clock time in Parameter n, when the transaction ENTERs Storage 1. The above model is effective, however, only if the maximum value of the clock does not exceed the maximum value which can be stored in the specified parameter. For this reason, it is suggested that fullword parameters be used when MARKING transaction parameters since the maximum range of a halfword parameter is 32,767 and the absolute clock might easily exceed this value.

REDEFINITION OF STORAGE CAPACITY WITH STORAGE CARD

"Storage Definition Card", earlier in this chapter, stated that each of the 150 storages in the standard GPSS/360 program is initialized to have a capacity of 2, 147, 483, 647= $(2^{31}-1)$. This is accomplished by simply setting $R_j=(2^{31}-1)$, while S_j remains zero. When a STORAGE definition card is read during the initial input phase, R_j (the remaining storage capacity) is set equal to the storage capacity specified in field A of the STORAGE card.

The initial reading is simply a special case of reading a STORAGE definition card during a sequence of runs. There are two possible cases:

1. The new Storage capacity is greater than the current Storage contents (S_j).
 S_j remains the same, while the remaining storage capacity (R_j) is set equal to the difference between the new Storage capacity and the current Storage contents (S_j).
2. The new Storage capacity is less than the current Storage contents (S_j).
 An appropriate error message is given and the run will be terminated when the next START card is read.

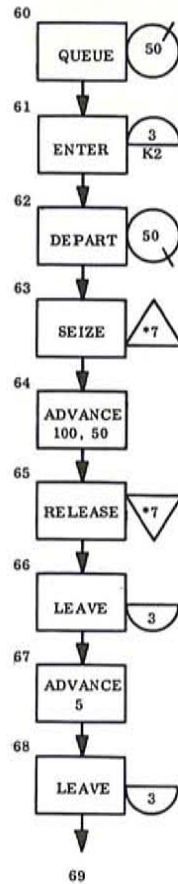
EXAMPLES OF ENTER AND LEAVE BLOCKS

Example 1:

In model A, transactions wait in Queue 50 until they can ENTER Storage 3. Each transaction that ENTERs block 61 occupies two storage units. After a transaction ENTERs Storage 3, it immediately attempts to SEIZE Facility *7. The facility number is specified indirectly by Parameter 7 of the transaction. When its action time (100 ± 50 clock units) at ADVANCE block 64 has been completed, a transaction makes available one of its two storage units at LEAVE block 66. Five clock units later, the second Storage unit is made available at LEAVE block 68.

Example 2:

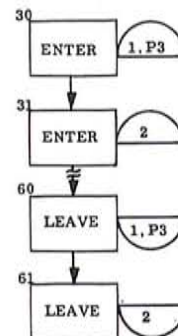
Transactions ENTER Storage 1 in model B with varying numbers of units specified by Parameter 3, which is the field B argument of ENTER block 30. The average time per transaction in the Storage statistics is, therefore, the average time per Storage unit. However, it is also desired to obtain the average time that each transaction spends in the Storage, regardless of how many units the transaction might have added or removed from the Storage.



MODEL A

LOC	OPERATION			A	B	C
	1	2	7			
60	QUEUE			50		
61	ENTER			3	K2	
62	DEPART			50		
63	SEIZE			*7		
64	ADVANCE			100	50	
65	RELEASE			*7		
66	LEAVE			3		
67	ADVANCE			5		
68	LEAVE			3		
3	STORAGE			10		

The average time per transaction may be obtained by concurrently ENTERing a companion statistical Storage 2 with a unit transaction count. Field B of ENTER block 31 is blank, indicating a unit entry count.

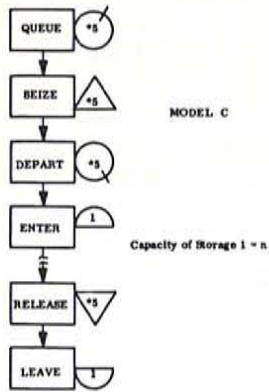


MODEL B

Storage 2 now provides the desired statistics on the average time per transaction, while Storage 1 provides the average time per Storage entry.

Example 3:

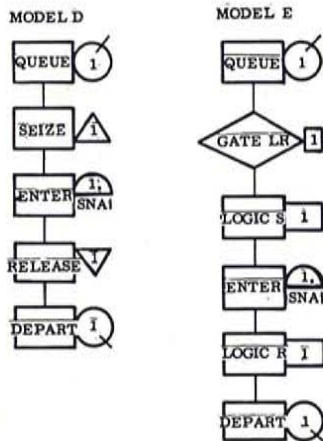
Storages can be used as important statistics-gathering devices. In Model C, a communication network has n lines numbered 1, 2, . . . , n . Each transaction carries one of the transmission line numbers (1, 2, . . . , n) in Parameter 5.



The standard GPSS/360 output provides the average utilization and average time per transaction on each of the line Facilities 1, 2, . . . , n. Storage 1, however, also provides the grand average utilization of all n line facilities, and the grand average time of all transactions.

Example 4:

When the field B argument of an ENTER block specifies a variable number of units to be entered into a storage, it is possible for a later transaction to ENTER the storage ahead of an earlier transaction. This can happen when a later transaction specifies a field B entry count which is less than the remaining Storage capacity (Rj), while an earlier transaction has specified a field B entry count which is greater than the remaining Storage capacity (Rj). The following block diagrams show two ways to ensure that later transactions cannot enter ahead of earlier transactions:



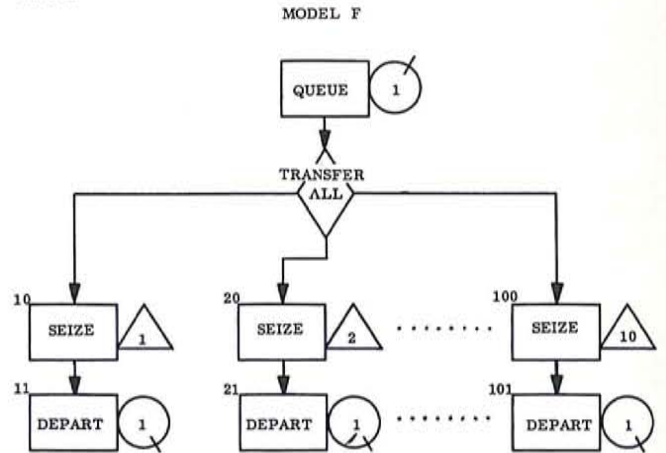
Either Facility 1 in model D, or Logic Switch 1 in model E, permits only one transaction at a time to try to ENTER Storage 1.

Example 5:

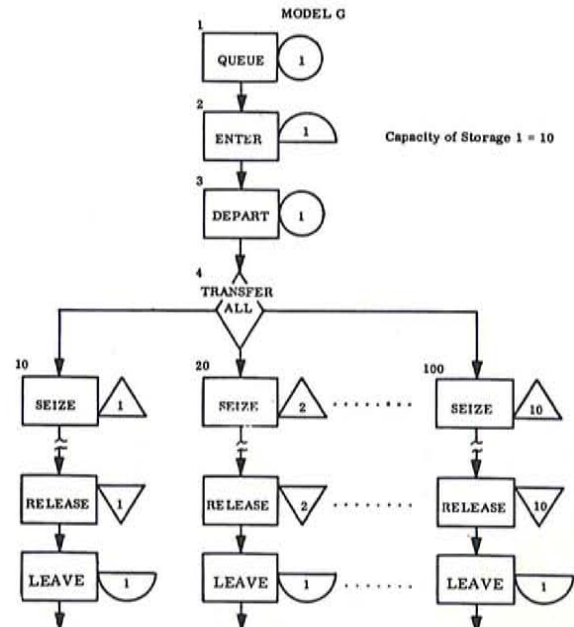
In model F, transactions attempt to SEIZE one of the 10 Facilities 1, 2, . . . 10. Facility 1 is always tried first, Facility 2 is tried next if

Facility 1 is already being used, and finally Facility 10, only if all of the other nine Facilities are being used.

Model F can run quite slowly, however, since the transactions, which are blocked in the TRANSFER ALL block, because all ten facilities are busy, are not deactivated from the overall GPSS/360 scan by being put into pushdown delay chains. Consequently, each time that the overall GPSS/360 scan encounters these transactions, it attempts to move them into each one of the 10 SEIZE blocks, even if all 10 Facilities are currently being used.



Model G, which follows, can be considerably faster. Transactions will be able to ENTER Storage 1 (which has a capacity of ten), only when one or more of the ten facilities are free to be SEIZED. Whenever all ten facilities are in use,



Storage 1 is also full. The delayed transactions in QUEUE block 1 are now linked in a unique delay chain of transactions which are waiting to ENTER Storage 1.

These transactions are thereby deactivated from being processed by the overall GPSS/360 scan.

Whenever a transaction RELEASEs one of the facilities, it also LEAVEs Storage 1. If all ten facilities were being used, i. e. , Storage 1 was full, the delay chain would be reactivated so that a blocked transaction could now ENTER Storage 1 and be sure of SEIZEing a free facility.

TABLE 21: S/360 CORE ALLOCATION FOR STORAGE ENTITIES

<u>SYMBOL</u>	<u>LENGTH</u>	<u>QUANTITY</u>	<u>STORAGE</u>
S1	4 bytes	Current contents of the Storage. This is the Standard Numerical Attribute Sj.	This field is incremented whenever a transaction enters an ENTER block that refers to the Storage. The magnitude of the increment is specified by the ENTER block field B argument.
S2	4 bytes	Number of available units in the Storage. This is the Standard Numerical Attribute Rj.	This field is decremented when a transaction enters an ENTER block which refers to the Storage. The magnitude of the decrease is specified by the ENTER block field B. Note that the sum on this field and S ₁ equals the capacity of the storage. When a transaction enters a LEAVE block, the number of units specified by the LEAVE block field B is added to this field, and subtracted from S ₁ .
S3-S4	8 bytes	Cumulative time integral	The cumulative time integral is a 64-bit, floating-point sum of the number of clock units that the Storage was occupied, weighted by the number of units in the Storage during each interval that the occupancy was constant. The integral is updated each time the Storage content changes status.
S5	4 bytes	Last clock time the Storage changed status.	This word contains the clock time at which the time integral was last updated. The clock time is placed in this word each time the Storage changes status, and when a RESET or CLEAR card is read.
S6	4 bytes	Entry count. This is the Standard Numerical Attribute SCj.	This word is incremented every time a transaction enters an ENTER block. The magnitude of the increment is specified at the ENTER block.
S7	4 bytes	Maximum Storage contents. This is the Standard Numerical Attribute SMj.	Each time a transaction enters an ENTER block which refers to the Storage, the new total contents of the Storage is compared against this field. If the new contents is greater than the previous maximum, it is entered in this word.
S8	2 bytes	Delay chain origin, for transactions waiting for the Storage to be "full."	Each time a transaction at a GATE SF block fails to advance because the Storage is not full, the number of that transaction is put in this field (unless the transaction is in a TRANSFER block with a BOTH or ALL selection mode). The previous transaction number in the field is placed in the first two bytes of transaction word T1, thus forming a pushdown delay

TABLE 21 (CONTINUED)

<u>SYMBOL</u>	<u>LENGTH</u>	<u>QUANTITY</u>	<u>STORAGE</u>
S8 (Cont)			chain of transactions waiting for the Storage to be full. The delay chain is reactivated whenever Rj (S2) becomes zero as a result of a transaction entering an ENTER block.
S9	2 bytes	Delay chain origin, for transactions waiting for the storage to be "not full."	The pushdown delay chain is formed in the usual manner, and is cleared whenever S2 becomes nonzero (after previously being zero) as a result of a transaction entering a LEAVE block. These transactions are blocked by a GATE SNF block.
S10	2 bytes	Delay chain origin, for transactions waiting for the storage to be "empty."	This pushdown delay chain is formed in the usual manner and is cleared whenever S1 becomes zero as a result of a transaction entering a LEAVE block. These transactions are blocked by a GATE SE block.
S11	2 bytes	Delay chain origin, for transactions waiting for the storage to be "not empty."	This pushdown delay chain is formed in the usual manner and is cleared whenever S1 becomes nonzero (after previously being zero) as a result of a transaction entering an ENTER block. These transactions are blocked by a GATE SNE block.
S12	2 bytes	Delay chain origin, waiting for a reduction in the contents of the Storage in order to enter the Storage.	This pushdown delay chain is formed in the usual manner, and is reactivated whenever a transaction enters a LEAVE block. These transactions are blocked by an ENTER block.
S13	2 bytes	Unused	

TABLE 22: STATISTICAL PRINTOUT FOR STORAGE ENTITIES

Storage	Capacity	Average Contents	Average Utilization	Entries	Average Time/Trans.	Current Contents	Maximum Contents
1	1000	555.21	.5552	4640	47862.90	634	1000
CPU	1	.60	.5987	212	1129.55	0	1
10	32767	9.42	.0003	308	12233.24	41	41

<p>Storage name or number</p> <p>↑</p>	<p>Originally field A of STORAGE definition card, subsequently the sum of $S_j + R_j$ ($S_1 + S_2$)</p> <p>↑</p>	<p>Cumulative Time Integral (S_3, S_4) Relative clock time since last RESET or CLEAR card</p> <p>↑</p>	<p>Entry Count (Word 6)</p> <p>↑</p>	<p>Cumulative Time Integral (S_3, S_4) Number of entries (Word S6)</p> <p>↑</p>	<p>Word S1</p> <p>↑</p>	<p>Word S7</p> <p>↑</p>
--	--	---	--------------------------------------	--	-------------------------	-------------------------

<p>Cumulative Time Integral (S_3, S_4) $\frac{\text{Relative clock time since last } x \text{ (} S_1 + S_2 \text{)}}{\text{Capacity}}$ RESET or CLEAR card</p>
--

TABLE 23: EFFECT OF RESET AND CLEAR CARDS ON STORAGE ATTRIBUTES

RESET Card			
Word	Length	Attribute Value Before RESET Card	Result of RESET Card on Attribute Value
S1	4 bytes	Current contents of Storage	Unchanged
S2	4 bytes	Number of available units	Unchanged
S3-S4	8 bytes	Cumulative Time Integral	Set to zero
S5	4 bytes	Clock time of last status change	Set to current value of absolute clock.
S6	4 bytes	Entry count	Set equal to current contents (S1 word)
S7	4 bytes	Maximum contents	Set equal to current contents (S1 word)
S8, S9 S10, S11 S12	2 bytes each	Origin of five different pushdown delay chains	Unchanged

CLEAR Card

All words are set to zero except word S2 (number of available units) which is set equal to the total Storage capacity (S1+S2) or (Rj+Sj).

CHAPTER 12: QUEUE ENTITIES

GENERAL NATURE

Queue entities are incorporated into simulation models for the purpose of gathering statistics on transactions which are delayed by a common cause or set of causes. The standard GPSS/360 program for a 128k machine has core allocated for 150 Queues, numbered 1, 2, . . . 150. Execution Error 500 will occur if a Queue number greater than the quantity allocated is referenced. Table 24 describes the attributes stored in the eight words required for each Queue.

These words accumulate various statistics on the contents of Queues, which are printed as part of the standard GPSS/360 output. Key Queue statistics include:

1. the average contents of the Queue;
2. the total number of entries into the Queue, i.e., into QUEUE blocks;
3. The number of entries which spent zero time in the Queue, i.e., which entered a DEPART block at the same clock time as they entered a QUEUE block;
4. The percentage of transactions which spend zero time in the Queue;
5. The average time that all entries spend in the Queue, between entering QUEUE and DEPART blocks;
6. The average time that delayed entries spend in the QUEUE, between entering QUEUE and DEPART blocks;
7. The current Queue contents, and the maximum contents observed since the last RESET or CLEAR card.

Two block types, QUEUE and DEPART, are associated with Queue entities.

STANDARD NUMERICAL ATTRIBUTES

Each Queue entity has seven Standard Numerical Attributes:

- Qj = current contents of Queue j
- QMj = maximum contents of Queue j
- QAj = average contents of Queue j
- QCj = total entries into Queue j
- QZj = zero entries into Queue j
- QTj = average time/transaction in Queue j
- QXj = average time/transaction in Queue j excluding zero entries

The range of values is zero through $2^{31}-1$.

STANDARD LOGICAL ATTRIBUTES

There are no Standard Logical Attributes associated with Queues. TEST blocks may, however, be used to define various relations between Queue SNA's and constants. These TEST blocks can then control the flow of transactions as a function of these Queue SNA's.

INTERNAL ATTRIBUTES

Table 24 describes the various internal attributes which are stored in the eight Queue words (Q1, Q2, . . . Q8). These include:

1. the current contents of the Queue (Q6 word). This is the Standard Numerical Attribute Qj.
2. the maximum Queue contents, QMj, observed since the last RESET or CLEAR card (Q7 word).
3. the total number of entries into the Queue, QCj, i.e., into QUEUE blocks (Q2 word).
4. the number of zero-delay entries into the Queue, QZj, (Q3 word).
5. the clock time of the last status change (Q1 word), i.e., when the Queue contents either increased (QUEUE block) or decreased (DEPART block).
6. a long-precision floating-point cumulative time integral of the Queue contents (Q4 and Q5 words).

QUEUE BLOCK

2	LOC	7	8	OPERATION	19	Index Number of Queue	A	[Number of Units To Enter Queue]	B	QUEUE	A, B
				QUEUE		SNAj, SNA*n k, *n		[SNAj, SNA*n] k, *n			

Examples:

```

QUEUE      3
QUEUE      *10, P1
QUEUE      FN3, 10
QUEUE      FN*11
    
```

QUEUE blocks never refuse entry to a transaction. Transactions attempt to enter the next sequential block following the QUEUE block. This is generally a block which can refuse entry to a transaction, i.e., SEIZE, PREEMPT, and ENTER blocks, or GATE and TEST blocks which are operating in a conditional entry mode. The field A argument of the QUEUE block specifies the index number of a Queue. The field B argument specifies the number

TABLE 24: S/360 CORE ALLOCATION FOR QUEUE ENTITIES

<u>Symbolic Core Location</u>	<u>Quantity</u>	<u>Source</u>
Q1	Last clock time the Queue changed status (4 bytes)	This word contains the last clock time at which the cumulative time integral was computed. The clock time is placed in the word each time the Queue changes status and when a RESET or CLEAR card is encountered.
Q2	Total entry count into Queue. (4 bytes)	This word is incremented each time a transaction enters a QUEUE block which refers to the Queue. The magni- tude of the increment is specified by the field B argument of the QUEUE block.
Q3	Number of zero-delay entries. (4 bytes)	This field is incremented whenever a transaction enters and leaves the Queue with no time delay. The magnitude of the increment is given by the field B argument of the QUEUE block.
Q4-Q5	Cumulative time integral. (8 bytes)	The cumulative time integral is a long- precision, floating-point number which represents the sum of clock units the Queue was occupied, weighted by the Queue content during each interval in which the content was constant.
Q6	Current contents of the Queue. (4 bytes)	This field is incremented whenever a transaction enters a QUEUE block which refers to the Queue, and is decremented whenever a transaction enters a corresponding DEPART block. The magnitude of the change is specified by field B of the QUEUE and DEPART blocks.
Q7	Maximum contents of the Queue. (4 bytes)	Each time a transaction enters a QUEUE block which refers to the Queue, the new contents, Q6, are compared against this field. If the new contents are greater, they replace this field.
Q8	QTABLE number. (4 bytes)	This field is set up at input time when a QTABLE card is read which refers to the Queue. The field contains a table number in which Queue delay times are entered.

of units which are to be added to the current QUEUE contents. If field-B is blank, it is assumed to be one.

The QUEUE block, therefore, behaves much like an ENTER block, with the important exception that a Queue does not have a capacity limit, i.e., transactions can always enter a QUEUE block and add to its contents.

The same transaction can add to the contents of an unlimited number of Queues, and can subsequently, in DEPART blocks, remove the same or different number of units from all, or just some, of these Queues.

Operations When a Transaction Enters a QUEUE Block

The current QUEUE contents, Qj, are incremented by the value of the field B argument. In GPSS III the current Queue contents were maintained modulo 2¹⁵. However, in GPSS/360, the current Queue contents may range from zero to 2³¹-1.

The new Queue contents, (Qj), are compared against the maximum contents, (QMj). If the new Queue contents are greater, this value becomes the new maximum contents.

The entry count into the Queue, (Q2 word), is incremented by the field B number of units, which can be one or more. Therefore, the number of transactions which enter the Queue will always be less than or equal to the Q2 entry count (QCj).

Observe that no record is maintained of which transactions enter a Queue, as opposed to facilities which record the numbers of the transactions that are SEIZEing and PREEMPTing the facility. Therefore, completely different transactions may DEPART from a QUEUE than those that entered it in QUEUE blocks. A transaction does not have to remove the same number of units from the Queue (in a DEPART block) that is added (in a QUEUE block). Ultimately, the additions and removals of all transactions must balance out. Otherwise, execution error 428 may occur (transaction DEPARTing Queue by more than Queue contents).

The QUEUE block subroutine also computes the length of time that the Queue was at its previous contents:

$$\text{Length of time} = \frac{\text{Current absolute clock time} - \text{Time of last status change (word Q1)}}{\text{Current absolute clock time}}$$

The product of this time interval and the previous contents is added to the cumulative time integral (which is maintained in words Q4 and Q5). The current absolute clock time is then stored in word Q1 as the time of the last status change.

The cumulative time integral of the Queue contents can be portrayed as shown in Figure 31.

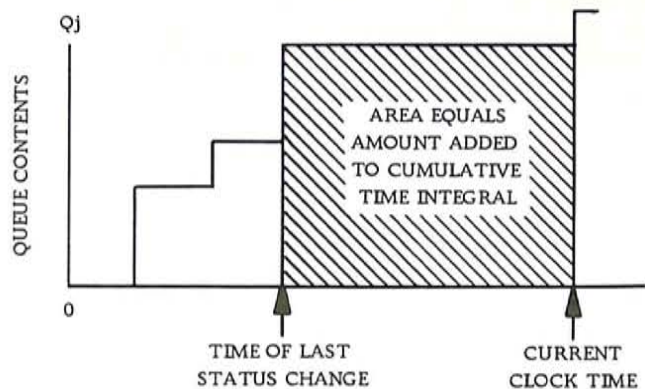


Figure 31. Cumulative Time Integral of Queue Contents

At the end of the simulation run, the cumulative time integral is divided by the value of the relative clock (C1) since the last RESET or CLEAR card to obtain the average contents of the Queue (QAj).

The time integral is also divided by the entry count to obtain the average time per entry into the Queue (QTj), and by the number of nonzero delay entries to obtain the average time per delayed entry into the Queue (QXj).

The QUEUE block subroutine performs one additional operation which is not performed by the ENTER block subroutine. The current absolute clock time is placed in transaction word T14, so that the time in the Queue may be subsequently tabulated in a QTABLE (see "QTABLE Card", later in this chapter, for further details).

DEPART BLOCK

2	LOC	7	8	OPERATION	Index Number of Queue 19 A	[Number of Units from Queue] B	DEPART	A, B
				DEPART	SNAj, SNA*n k, *n	[SNAj, SNA*n] k, *n		

Examples:

```
DEPART 5
DEPART 1, Q1      Empty the Queue
DEPART 6, FN*11
DEPART V3, V4
```

The DEPART block is used to reduce the contents in a Queue entity. A transaction is always permitted to enter a DEPART block, but a restriction is made on the amount of Queue contents which may be removed. The DEPART block specifies the Queue number in field A and the number of

units to be removed in field B in exactly the same way as the QUEUE block. If field B is blank, one unit is removed from the Queue contents. Transactions proceed to the next sequential block following the DEPART block.

Execution Error 428 (transaction DEPARTing Queue by more than Queue contents) will occur if the number of units to be removed from the Queue (the value of the field B argument, or one if field B is blank) is greater than the current contents of the Queue (Qj). Execution Error 428 will obviously occur if a transaction enters a DEPART block when the referenced Queue is empty.

Each time a transaction enters a DEPART block, the current contents (Qj) of the referenced Queue is decremented by the number of field B units. The entry count is not affected by the DEPART block.

The DEPART block subroutine (just like the QUEUE block) computes the length of time that the Queue was at its previous contents:

$$\text{Length of time} = \text{Current absolute clock time} - \text{Time of last status change (word Q1)}$$

The product of this time interval and the previous contents is added to the double-word cumulative time integral. The current absolute clock time is then stored in word Q1 as the time of the last status change.

The DEPART block subroutine checks whether the transaction has spent zero time in the Queue. If the current absolute clock time is the same as the time stored in transaction word T14, it is assumed that the transaction has spent zero time in the Queue. This is because the QUEUE block subroutine places the entry clock time in word T14. If there was a zero delay time (word T14 = current clock time), the number of zero delay time entries in word Q3 is incremented by the number of units specified by the field B argument of the DEPART block.

The DEPART block subroutine finally checks if a QTABLE number has been stored in word Q8. If a QTABLE is referenced, the time spent in the Queue is tabulated in the table. The Queue delay time is computed as follows:

$$\text{Time spent in queue} = \text{Current absolute clock time} - \text{Time in transaction word T14}$$

QTABLE CARD

Queue entities provide, as part of their standard statistical output, the average time per transaction (see Table 25). This is equal to the cumulative

time integral of the Queue contents divided by the number of entries into the Queue. As discussed in "Examples of QUEUE and DEPART Blocks" later in this chapter, this statistic may be less than the true long-run average Queue delay time.

Model A shows one way to obtain the complete distribution of true Queue delay times. Table 5, which is referenced by TABULATE block 44, provides the Queue delay times measured on Parameter 2 (argument MP2).

Queue entities, however, provide a simpler and more powerful means of tabulating the complete distribution of true Queue delay times. As shown in model B, a QTABLE definition card has the following arguments:

1. Field A specifies a Queue number which may be a constant or a symbol.

2. The location field specifies a table number, which is stored in Q8 word of the Queue specified by field A.

3. Fields B, C, and D are the same as for a normal TABLE definition card.

These fields set up a table entity as described in Chapter 13:

Field B: Upper limit of lowest table frequency interval

Field C: Width of table frequency interval

Field D: Number of frequency intervals

Whenever a transaction enters a DEPART block, the Q8 word is tested for a table number. If a table number has been entered via a QTABLE definition card, the following Queue delay time for the transaction is tabulated in the specified table:

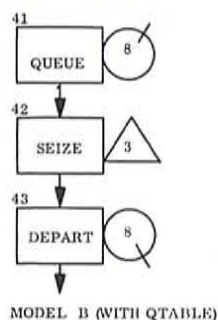
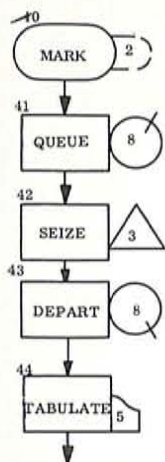
$$\text{Queue delay time} = \text{Current absolute clock time} - \text{Time stored in transaction word T14}$$

Zero delay times are included in the tabulated values.

The time in word T14 should be the absolute clock time when the transaction entered a QUEUE block which references the same Queue as the DEPART block. (The QUEUE block subroutine automatically places the entry clock time in word T14.) This simple mechanism subsequently permits the true Queue delay times to be tabulated in QTABLEs.

MULTIPLE QUEUES

In GPSS/360, a transaction may be a member of up to five Queues at any one time. This enables the analyst to obtain meaningful Queue statistics for transactions which enter multiple Queues at different points of time. Thus, intermediate delay times may be measured along with overall delay



MODEL A

LOC	OPERATION	19	A	B	C	D
40	MARK	2				
41	QUEUE	8				
42	SEIZE	3				
43	DEPART	8				
44	TABULATE	5				
5	TABLE	MP2	0	5	40	DELAY TIME
41	QUEUE	8				
42	SEIZE	3				
43	DEPART	8				
5	QTABLE	8	0	5	40	Same results as above

times. Transaction word T14 contains an address in COMMON of multi-Queue table which contains the Queue numbers and QUEUE block entry times.

Execution Warning Messages

An attempt by a transaction enter more than five Queues at any one time will result in the following warning message being printed:

WARNING EXECUTION ERROR NUMBER 853.
BLOCK NUMBER XXXX. CLOCK YYYY. SIMULATION CONTINUES.

where XXXX = QUEUE block at which the error occurred.

YYYY = Clock time at which the error occurred.

The contents of the Queue specified by field A will be incremented by the amount specified in field B. Testing for maximum contents will be made and updating will be done accordingly. Cumulative time integral information will not be calculated. Therefore, statistics on average time per transaction in Queue, number of zero entries, percent zeros, and average time per transaction excluding zero entries will be in error. An attempt by a transaction to depart from a Queue of which it is not a member will result in the following warning message being printed:

WARNING EXECUTION ERROR NUMBER 854.
BLOCK NUMBER XXXX. CLOCK YYYY. SIMULATION CONTINUES.

where XXXX = DEPART block at which the error occurred.

YYYY = Clock time at which error occurred.

The transaction will proceed to the next sequential block, and because it may be the intention of the analyst to allow one transaction to remove units from a Queue which were added by another transaction, the number of units specified in field B of the DEPART block will be removed from the Queue.

Each warning message will be printed only the first time the specified error occurs.

INCLUSION OF ADVANCE BLOCKS BETWEEN QUEUE AND DEPART BLOCKS

In GPSS III, transaction word T4 was used to store the QUEUE block entry time as well as the time the transaction was scheduled to leave the future events chain. Consequently, inclusion of ADVANCE blocks between QUEUE and DEPART blocks often resulted in meaningless output statistics.

In GPSS/360, since each QUEUE block entry time (for up to five Queues) is stored separately in a table associated with each transaction, the user may now insert ADVANCE blocks between QUEUE and DEPART blocks without destroying Queue statistics.

STATISTICAL PRINTOUT

The standard Queue statistical printout shown in Table 25 is printed under the following three conditions:

1. A simulation run is terminated normally after the termination count (specified in field A of the latest START card) is decremented to zero or less.
2. A simulation run is terminated by one of the running errors described in Appendix A.
3. A transaction enters the following type of PRINT block:

PRINT k_l, k_u, Q

As described in "PRINT Block" in Chapter.14, the standard statistics are printed for the Queues from the field A lower limit (k_l) up through the field B upper limit (k_u). In all three cases, statistics are printed only for those Queues which have had one or more entries (as recorded in word Q2).

Average Contents of Queue

The average contents of each Queue, which is printed at the end of each simulation run, is equal to:

TABLE 25: STATISTICAL PRINTOUT FOR QUEUE ENTITIES

Queue	(QMj) Maximum Contents	(QAj) Average Contents	(QCj) Total Entries	(QZi) Zero Entries	Percent Zeros	(QTj) Average Time/Trans	(QXi) \$ Average Time/Trans	Qtable Number	(Qj) Current Contents
1	2	.02	10	6	60.0	900.30	2250.75	10	1
CPU	72	41.55	83	1	1.2	200253.61	202695.73	0	72
8	63	23.24	476	289	60.7	19531.91	49717.58	3	50
	Q7		Q2	Q3	Q3 x 100	Excluding Zero Entries		Q8	Q6
		\$ Average	Time/Trans	Average	Time/Trans	Cumulative Time			
Queue name or number		Integral (Q4 and Q5) Relative Clock (C1)				Integral (Q4 and Q5) Total Entries (Q2)			Cumulative Time Integral (Q4 and Q5) Number of nonzero delay entries (Q2-Q3)

$$\text{Average Contents} = \frac{\text{Cumulative time integral of contents (Q4 and Q5 word)}}{\text{Relative clock time (C1) since last RESET or CLEAR card}}$$

The 64-bit cumulative time integral is the area under the Queue contents profile shown in Figure 32. The average Queue contents is simply the average height of the profile.

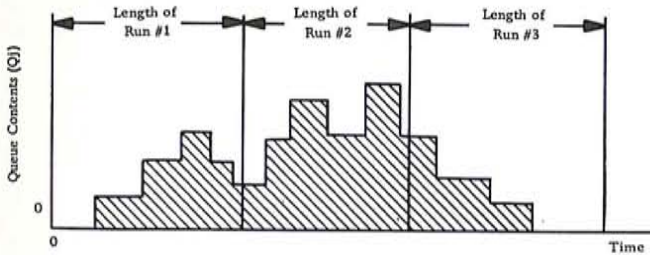


Figure 32. Queue Contents Profile

RESETing and CLEARing of Cumulative Time Integral

If RESET or CLEAR cards are not used between multiple STARTs, the cumulative time integral continues to build up the area under the Queue contents profile. "Effect of RESET and CLEAR Cards on Storage Statistics", in Chapter 11, described how RESET cards provide individual run statistics on storage entities. The same considerations apply to Queue entities. Once again, there is a statistical advantage in using RESET cards since the individual run statistics provide a measure of the sampling variability around the grand average values. Without the RESET cards, only the grand average values would be obtained.

Average Time per Transaction

Two average time per transaction statistics are printed for each Queue:

$$\text{Average Time/Trans} = \frac{\text{Cumulative Time Integral}}{\text{Total entries (word Q2)}}$$

$$\text{\$Average Time/Trans} = \frac{\text{Cumulative Time Integral}}{\text{Number of delayed entries (word Q2-Q3)}}$$

The first average time is based on all transactions, including zero time delay transactions. The second average time excludes zero-time delay

transactions, i.e., it is the average time spent in the Queue by only those transactions which were actually delayed.

EFFECT OF RESET AND CLEAR CARDS ON QUEUE STATISTICS

Table 26 summarizes the effects of RESET and CLEAR cards on the attributes of Queue entities. A RESET card zeros the cumulative time integral (Q4 and Q5 words), and sets the clock time of the last status change (Q1 word) equal to the current absolute clock time. The entry count (Q2 word) is then set equal to the current Queue contents (Qj) which may be in the range between 0 and $2^{31}-1$. "Selective Reset" in Chapter 15 describes the method by which the user may specify certain Queues which the RESET operations will not affect.

A CLEAR card zeros out all the Queue statistics, including the current Queue contents.

The above entry count adjustments can have the same significant effect on the computed average time per transaction that was discussed in "Average Time per Transaction" in Chapter 11.

The following general rule can be stated for all Queues:

The computed average time per transaction on each Queue is always less than or equal to the true long-run average time per transaction. This depends on whether the true number of entries has been overstated because of RESET card operations.

The above also applies to the percentage of zero-delay entries.

QTABLES provide a solution to this problem as discussed in "QTABLE Card" earlier in this chapter. Such tables will provide the complete distribution of true Queue delay times.

The two average time per transaction statistics are really the average time per unit entered into the Queue, since the field B argument of the QUEUE block determines the number of units entered into the Queue by each transaction. Words Q2 and Q3 accumulate the number of units, which is not necessarily the number of transactions which entered QUEUE and DEPART blocks which reference the Queue. The entry counts and the number of transactions are equal only when each transaction contributes a unit entry, e.g., when fields B of all QUEUE and DEPART blocks are blank.

EXAMPLES OF QUEUE AND DEPART BLOCKS

Example 1: Grand Average Statistics for a Set of Queues

TABLE 26: EFFECT OF RESET AND CLEAR CARDS ON QUEUE ATTRIBUTES

RESET Card

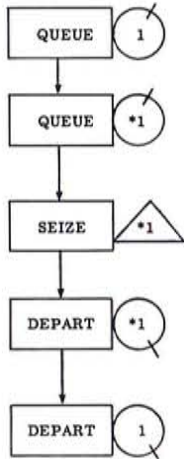
Word	Attribute Value before RESET Card	Result of RESET Card on Attribute Value
Q1	Last clock time the Queue changed status	Set equal to current value of absolute clock
Q2	Total entry count	Set equal to the current contents of the Queue
Q3	Number of zero delay entries	Set to zero
Q4-Q5	Cumulative time integral	Set to zero
Q6	Current contents of the Queue	Unchanged
Q7	Maximum contents of the Queue	Set equal to the current contents of the Queue
Q8	Qtable number	Unchanged

CLEAR Card

All eight words and their fields are set to zero except:

Q8	Q table number	Unchanged
----	----------------	-----------

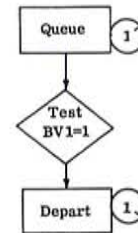
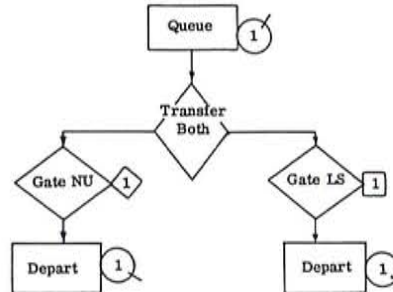
In the following model, Transaction Parameter 1 contains the numbers 2, 3, . . . n:



Consequently, transactions which attempt to SEIZE Facility *1 (=2, 3, . . . n) first enter one of the Queues *1 (=2, 3, . . . ,n). Each transaction, however, also enters Queue 1, which provides the grand average for all the Queues 2, 3, . . . n.

Example 2:

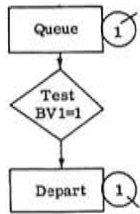
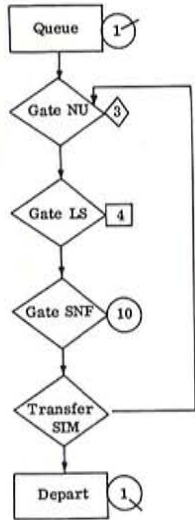
In each of the following models, transactions will be delayed in Queue 1 only when both Facility 1 is in use and Logic Switch 1 is in a reset condition:



BV1 = FNUI+LS1

Example 3:

In each of the following models, transactions will be delayed in Queue 1 until Facility 3 is "not-in-use", Logic Switch 4 is "set," and Storage 10 is "not full".



$BV1 = FNU3 * LS4 * SNF10$

CHAPTER 13: DISTRIBUTION TABLE ENTITIES

GENERAL NATURE

Distribution Table entities are used in simulation models to obtain the frequency distribution of a specified argument, which can be any one of the Standard Numerical Attributes. The standard GPSS/360 program for a 128k machine has core allocated for 30 Tables. Execution Error 435 will occur if an illegal table number is referenced during a run, either in a TABULATE block or as the index *j* of the Standard Numerical Attribute TB_{*j*}, TD_{*j*}, or TC_{*j*}. Table 27 describes the attributes stored in the twelve words required for each Table.

TABLE STATISTICS

Table 28 illustrates the standard set of statistics which are printed out for each Table at the end of a simulation run, or after an Execution Error. These statistics, which are described in detail in "Table Statistical Printout" later in this chapter, include:

1. The sum of the Table arguments which have been TABULATED since the last RESET or CLEAR card (or during the first simulation run). The sum of the arguments is accumulated in word D1.
2. The number of entries into the Table, i.e., the number of times that a transaction entered a TABULATE block which referenced the particular Table and thereby resulted in tabulation of the value of the Table argument. The number of entries is accumulated in word D6.
3. The mean value of the Table arguments - which is simply,

$$\begin{array}{l} \text{Mean} \\ \text{Value} \\ \text{of} \\ \text{Argu-} \\ \text{ments} \end{array} = \frac{\text{Sum of Table arguments (word D1)}}{\text{Number of entries into Table (word D6)}}$$

4. The standard deviation of the Table arguments whose computation is described later in "Table Statistical Printout."
5. The average value of overflow argument values in the upper (last) frequency interval. This will be printed at the bottom of the Table printout if any overflow values were observed.
6. One output line of statistics for each frequency interval of the Table, including the following:

- a. Number of times the value of the Table argument was within each frequency interval.

- b. The percentage of the total observations in each frequency interval.
- c. The cumulative percentage of the total observations which are equal to or less than the upper limit of each frequency interval.
- d. The cumulative remaining percentage of the total observations which are greater than the upper limit of each frequency interval.
- e. The upper limit of the frequency interval expressed as a multiple of the mean value of the Table argument values.
- f. The upper limit of the frequency interval expressed as a multiple of the standard deviation from the mean value of the Table argument values.

STANDARD NUMERICAL ATTRIBUTES

Each Table entity has three Standard Numerical Attributes associated with it; these can be addressed at any time by the mnemonic TB_{*j*}, TD_{*j*}, or TC_{*j*}:

TB_{*j*} = mean of the argument values accumulated in the Table since the last RESET or CLEAR card (or since the start of the simulation job if no RESET or CLEAR cards have as yet been encountered).

$$= \frac{\text{Sum of arguments (D1)}}{\text{Number of entries (D6)}}$$

TD_{*j*} = Table Standard Deviation, square root of the variance

$$= \sqrt{\frac{D2 - (D1)^2/D6}{D6-1}}$$

TC_{*j*} = number of entries in Table

$$= D6$$

STANDARD LOGICAL ATTRIBUTES

There are no Standard Logical Attributes associated with Table entities.

TABULATE BLOCK

2	LOC	7	8	OPERATION	19	A	Index number of Table	Weighting B Factor	TABULATE	A
				TABULATE	SNA _{<i>j</i>} , SNA* _{<i>n</i>}	k, * _{<i>n</i>}		SNA _{<i>j</i>} , SNA* _{<i>n</i>}		
								* _{<i>n</i>} , k		

TABLE 27: S/360 CORE ALLOCATION FOR TABLE ENTITIES

Word	Field	Quantity	Source
D1	Full Word	Sum of the arguments entered in the Table.	This value (signed, floating-point) is incremented by each argument value that is entered in the Table. The sum is used in the computation of the mean argument.
D2	Full Word	Sum of the squared values of the arguments entered in the Table.	This floating-point value is incremented by the square of each argument value that is entered in the Table. The sum is used in the computation of the standard deviation of the Table argument.
D3	Full Word	Sum of the weighted values of the arguments entered in the Table	This floating-point value is incremented by the product of the argument and the number of units specified by the field B weighting-factor of the TABULATE block which is initiating the tabulation. The sum is used in the computation of a weighted mean.
D4	Full Word	Sum of the weighted, squared values of the arguments entered in the Table.	This floating-point value is incremented by the product of the square of the argument and the number of units specified by the field B weighting factor of the TABULATE block which is initiating the tabulation. The sum is used in the computation of a standard deviation of the weighted argument.
D5	Full Word	Temporary storage for the Table.	This fixed-point value contains either: <ol style="list-style-type: none"> 1. The last value of the argument for a Table which is operating in the difference mode or IA mode. 2. The sum of the number of units that have arrived for a Table which is operating in the RT mode. In the latter case, the number of units contributing to the sum is specified by field B of the TABULATE block.
D6	Full Word	Number of entries in Table.	This value is incremented each time a transaction enters a TABULATE block referring to the Table (except for RT mode). The magnitude of the increment is specified by the field B argument of the TABULATE block.
D7	1 Byte	Mode of tabulating Table arguments.	Set up at input time from field A of the TABLE card which specifies the argument and field D which specifies weighted or nonweighted mode. Bit 5 - Weighted mode Bit 4 - IA mode Bit 3 - RT mode Bit 2 - Difference mode Bit 1 - Normal mode Bits 0, 6, 7 - Not used
	3 Bytes	Width of the Table frequency classes.	This constant is set up at input time from field C of the TABLE definition card.

TABLE 27: (Continued)

Word	Field	Quantity	Source
D8	Full Word	Address of COMMON where frequency classes are located.	This field is set up at input time when the number of classes is requested by the TABLE card field D. The block of storage is obtained from the chain of COMMON storage.
D9	Full Word	Table argument or arrival rate time interval for RT mode.	This field is set up at input time from field A or field E of the TABLE definition card.
D10	Full Word	Sum of overflow argument values in last frequency interval of Table.	This value (signed, floating-point) is incremented by each argument value which lies in the last frequency interval of Table. This sum is divided by the number of entities in the last interval to obtain "AVERAGE VALUE OF OVERFLOW."
D11	Half Word	Number of frequency classes in Table.	This constant is set up at input time from field D of the TABLE definition card.
D12	Half Word	Temporary storage for RT mode Table.	Nonweighted sum of the number of units that have arrived for Table operating in the RT mode.
D13	Full Word	Upper limit of the lowest interval of the Table.	This constant is set up at input time from field B of the TABLE definition card.

The current argument value of a Table is TABULATED whenever a transaction enters a TABULATE block. An appropriate entry will be made in the Table, whose number is specified by field A of the TABULATE block. It should be noted that the type of argument tabulation requested depends on the mode of operation of the Table. For example, the Table may take for its argument M1, the transit time of the current transaction (the one entering the TABULATE block). In contrast, the Table may use the contents of a queue (Qj) as its argument. In this case, Qj does not depend on any property of the transaction entering the TABULATE block. The TABULATE block should thus be regarded as a general request for an appropriate Table operation, with the specific request determined by the TABLE definition card (see below).

A TABULATE block may also specify the number of units to be added to the class of the distribution in which the argument is placed. Field B is used to specify this quantity. If the field is blank, the number is assumed to be equal to one. If a weighting factor is used (field B), field D of the associated TABLE definition card must begin with any alphabetic character.

TABLE DEFINITION CARD

Table number 2 LOC	7	8 OPERATION	Table argument 19 A	Upper limit of lowest interval B	Internal width C	Number of intervals D	Arrival rate time interval E
j		TABLE	SNAj SNA*n SNAj-, SNA*n-, RT IA	k _B	k _C	k _D	k _E

Examples:

2 LOC	7	8 OPERATION	19 A	B	C	D	E
10		TABLE	M1	500	100	25	
20		TABLE	RT	0	1	60	1500
33		TABLE	IA	100	100	20	
3		TABLE	N20	0	1	12	

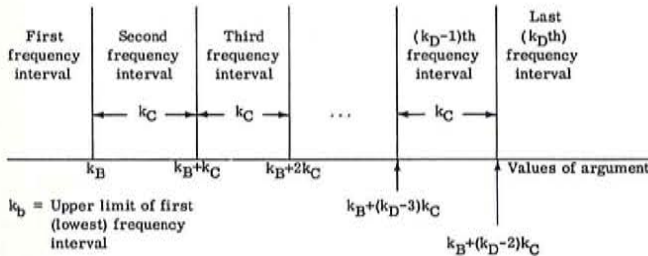
The TABLE definition card provides the user with a means of specifying the independent argument as well as the range and width of frequency classes. When a minus sign (-) follows the Standard Numerical Attribute used as the Table argument, the Table operates in a difference mode. The difference between the current argument value and its previously TABULATED value is recorded in the

appropriate Table frequency interval (the most recently TABULATED value is stored in word D5 of Tables operating in the difference mode). The special Table argument RT indicates that the Table operates in an arrival rate mode (see Example 3 in "Table Arguments" below). The special Table argument IA indicates that the Table operates in an interarrival time mode (see Example 3 in "Table Arguments" below.)

If the field A argument of a TABULATE block references an undefined Table, Execution Error 436 occurs.

When a TABLE definition card is encountered the field D argument (k_D) specifies the number of frequency intervals in the Table. The GPSS/360 program obtains k_D words from COMMON storage and associates them with the Table entity, whose index number j is specified in the location field of the TABLE card. The field C argument of the TABLE definition card (k_C) specifies the width of each of the k_D frequency intervals of the Table.

The Table entity can be portrayed as follows in terms of the field B, C, and D arguments:



The first (lowest) frequency interval extends from -2^{31} up through and including k_B , the upper limit of the first interval which is specified in field B of the TABLE definition card. The first of the k_D COMMON storage words accumulates the number of times that the value of the Table argument falls within the lowest (first) frequency interval. Next, there are $(k_D - 2)$ inner frequency intervals, each with a width k_C . There are $(k_D - 2)$ corresponding COMMON storage words, each of which accumulates the number of times that the value of the Table argument falls within the particular frequency interval. The last (k_D th) frequency interval extends from $k_B+(k_D-2)k_C$ (which is the upper limit of the next to last frequency interval) up through $+ (2^{31}-1)$. The last (k_D th) COMMON storage word accumulates the number of times that the value of the Table argument falls within the last frequency interval. This last frequency class is identified as the overflow in the Table printout.

TABLE ARGUMENTS

Field A of the TABLE definition card specifies the Table argument. The value of the argument is

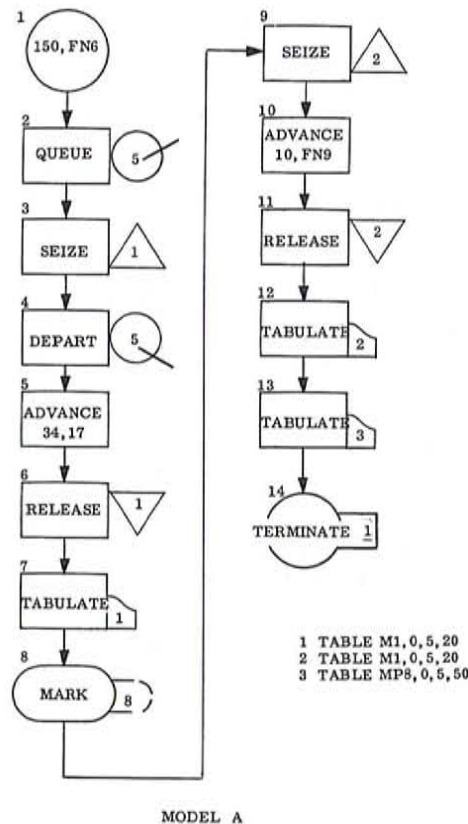
TABULATED each time that a transaction enters a TABULATE block which references the particular Table (the Table argument is stored in word D₉ of the Table entity). The Table argument can be any Standard Numerical Attribute, except a constant or a matrix savevalue (MX or MH).

In GPSS III, *n was not permitted as a Table argument, but in GPSS/360 *n as well as P_n can be used as a Table argument to reference the value of Parameter n of the transaction in the TABULATE block.

Example 1:

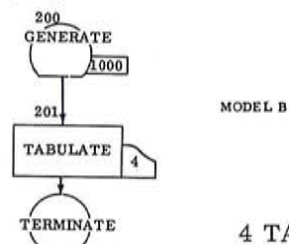
Model A, which is shown and described below, shows how transaction transit times may be TABULATED.

Table 1 will TABULATE M1, the transaction transit time from block 1 through block 6. Table 2 TABULATEs the time between block 1 and 11, while Table 3 TABULATEs an intermediate transit time, MP8, between blocks 8 and 11. The MARK block 8 stores the low-order 15 bits of the current absolute clock time in Transaction Parameter 8 (assuming that parameter 8 is a halfword parameter. If parameter 8 is a fullword parameter, the absolute clock time is stored in parameter 8.) The intermediate transit time MP 8 can then be TABULATED in Table 3.



MODEL A

Example 2: Difference Mode of Table Arguments Model B, which follows, may be used in conjunction with the above example to TABULATE the number of Transactions that arrive at block 2 during intervals of 1000 clock units. Note that the minus sign (-) following the N2 indicates that not N2, but the difference between N2 Standard Numerical Attribute argument and its previously tabulated value, is to be recorded in the appropriate frequency interval of the Table. This mode of operation is termed the "difference mode". When operating in the difference mode, the first tabulation is automatically omitted by the GPSS/360 program.



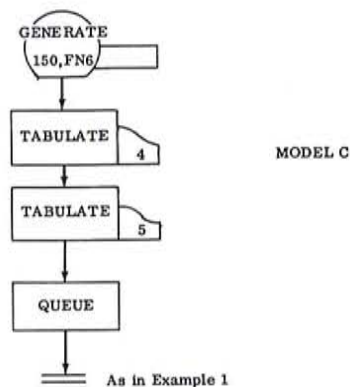
Example 3: Arrival Rate (RT) and Interarrival Time (IA) Table Modes

The same results as those in Example 2 may be achieved more simply by the following Model C. The mnemonic RT is not a Standard Numerical Attribute but, instead, designates that Table 4 will operate in the arrival rate mode. When a transaction enters a TABULATE block that refers to a Table operating in the arrival rate mode, an entry is not made in the frequency distribution. Instead, the temporary storage word D12 of the Table is incremented by one and the word D5 is incremented by the number of units specified at the TABULATE block (weighting factor). The TABLE definition card which specifies RT in field A must also specify an arrival rate time interval in field E. If this time interval is 1000, as in Table 4, for example, the arrival count accumulated in word D5 will be recorded in the appropriate frequency interval of the Table every 1000 clock units (TABLE cards with RT in field A are the only ones which require a field E argument). Words D12 and D5 will then be set to zero to begin accumulating the number of arrivals into the Table during the next time interval (the use of arrival rate Transaction operators is described in "Transaction Operators for RT Arrival Rate Tables" later in this chapter).

Table 5 in the following diagram also specifies a special mode, which is termed the "interarrival time mode" (IA). When a transaction enters a TABULATE block that refers to a Table operating in the interarrival time mode, the GPSS/360

program computes the time arrival since the previous reference to the Table. This value is recorded in the appropriate frequency interval of the Table. As in the difference mode, the first tabulation is omitted. The absolute clock time when the previous tabulation occurred is stored in Table word D5. Whenever a Transaction enters a TABULATE block, which is referencing an IA Table, the time interval is very simply computed as:

$$\text{Interarrival time interval} = \text{Current absolute clock time} - \text{Clock time of previous tabulation (word D5)}$$



4 TABLE RT, 0, 1, 60, 1000
5 TABLE IA, 10, 10, 25

Example 4: Weighted Table

In Example 3, if the arrival rate mode table is weighted by XH1, the D5 word would be incremented by the current value of Halfword Savevalue 1 when a transaction enters TABULATE block 2. Note that whenever a weighting factor is used, field D of a TABLE card must begin with an alphabetic character, followed by the number of intervals.

2 TABULATE 4, XH1
4 TABLE RT, 0, 1, A12, 1000

The other cards are the same as in Example 3.

INTERNAL OPERATION OF TABLES AND TABULATE BLOCKS

Whenever a transaction enters a TABULATE block, the following steps occur (for nonweighted normal mode):

1. The first byte of D7 word is tested to determine in which mode the Table is operating (see Table 27):

- Bit 5 = weighted mode
- Bit 4 = IA mode
- Bit 3 = RT mode
- Bit 2 = Difference mode
- Bit 1 = Normal mode

2. The value of the Table argument is computed according to the Table mode, as described in "Table Arguments" earlier.

3. The argument value is added to the running sum of arguments, which is maintained in word D1.

4. The argument value is squared and added to the running sum of squared arguments, which is maintained in word D2.

5. The number of entries into the Table, which is maintained in word D6, is incremented by one.

6. The frequency interval in which the argument value lies is determined. The observation count for this interval is then incremented by one.

7. If the argument value lies in the last (upper) frequency interval of the Table, the argument value is added to the cumulative sum of overflow values, which is maintained in word D10.

TRANSACTION OPERATORS FOR RT ARRIVAL RATE TABLES

Each Table which operates in the RT arrival rate mode has a unique transaction operator associated with it. When a TABLE definition card is encountered with RT in field A, the GPSS/360 input program performs the following steps:

1. The field B and C arguments are stored in words D13 and D7 (as with all other TABLE definition cards).

2. kD COMMON storage words are associated with the Table (as with all other TABLE cards).

3. The RT arrival rate mode is indicated by setting bit 3 of D7 word on.

4. The rate-time interval, which is specified in field E, is stored in word D9.

5. The most unusual step now occurs. An unused transaction is obtained from the internal chain of inactive transactions. The table number is stored in the lower half of Transaction word T6. This transaction operator can be distinguished from the normal transaction by the fact that its word T5 is zero; i. e., it is not associated with any assembly set (lower half of T5 word is zero). The rate-time interval is added to the current absolute clock time and stored in Transaction word T4 just as if the transaction operator were in an ADVANCE block. The arrival rate (dummy) transaction is then merged, according to the T4 time, into the future events chain (again, just as

if it were in an ADVANCE block). The transaction will be removed from the future events chain at the time stored in word T4 (just as if it were in an ADVANCE block).

Subsequent Processing of the Arrival Rate Transaction

Other transactions now begin to enter TABULATE blocks which reference a Table that is operating in the RT arrival rate mode. Each time that a transaction references such a Table, the arrival count in word D12 is incremented by one and the arrival count in word D5 is incremented by weighting factor. For nonweighted case, contents of D12 and D5 should be equal.

Eventually, the absolute clock time is updated to the time stored in word T4 of the arrival rate transaction. Normal transactions with the same time in word T4 will be transferred to the current events chain (these normal transactions have been in ADVANCE blocks, or they have been incipient transactions waiting to enter the system at a GENERATE block).

The GPSS/360 program checks each of these transactions to determine if it is an arrival rate operator, i. e., if its word T5 is zero. When such a transaction is encountered, the GPSS/360 program performs the following steps:

1. The table with which the arrival rate transaction is associated is determined from the lower half of Transaction word T6. The number of units which have been accumulated in Table word D5 during the most recent time interval is tabulated in the appropriate frequency interval of the Table.

2. Words D5 and D12 are zeroed so that it can begin again to accumulate the number of transactions, which arrive in TABULATE blocks, referencing the RT Table, during the next time interval.

3. The next tabulation time is computed and stored in word T4 of the transaction operator as follows:

$$\begin{array}{rcl} \text{Next} & & \text{Current} & & \text{Arrival rate time} \\ \text{tabulation} & = & \text{absolute} & + & \text{interval (stored in} \\ \text{time} & & \text{clock time} & & \text{Table word D9)} \end{array}$$

The arrival rate transaction is then merged into the future events chain according to its T4 time (just as if it were in an ADVANCE block).

The net result of the above steps is that the arrival rate transaction goes through a repetitive cycle. It spends the arrival-rate time interval (the TABLE card field E argument) in the future

events chain. It emerges briefly to activate the tabulation of the number of arrivals into the Table during the most recent interval. The transaction is then merged back into the future events chain.

Whenever a CLEAR card is encountered, the dummy transaction for each RT Table is destroyed. Before beginning the next simulation run, the GPSS/360 program will, however, recreate a new transaction for each RT Table in the model.

TABLE STATISTICAL PRINTOUT

The standard Table statistical printout is shown in Table 28. The computations that are required for the various statistics are outlined below. The following basic notation is used:

x_i = The value of an i th Table argument, where $i = 1, 2, \dots, N$. N is the total number of entries into the Table, which is stored in word D_6 .

n_j = The number of argument values which fall into the j th frequency interval, where $j = 1, 2, \dots, k_D$. k_D is the number of Table intervals as specified by field D of the TABLE definition card. We have

$$\sum_{j=1}^{k_D} n_j = N.$$

k_B = Upper limit of the lowest frequency i interval (specified in field B of the TABLE definition card).

k_C = Width of each frequency interval (specified in field C of the TABLE definition card).

Mean Value*

$$\bar{X} = \frac{\sum_{i=1}^N x_i}{N} = \frac{\text{Word D1}}{\text{Word D6}}$$

$$\bar{X}_w = \frac{\sum_{i=1}^N (x_i \times \text{W.F. } i)}{\sum_{j=1}^{k_D} (n_j \times \text{W.F. } j)} = \frac{\text{Word D3}}{\text{Word (D6)}_w}$$

Standard Deviation of Sample*

$$S = \sqrt{\frac{D_2 - D_1^2/D_6}{D_6 - 1}}$$

$$S_w = \sqrt{\frac{D_4 - D_3^2/(D_6)_w}{(D_6)_w - 1}}$$

*Subscript W denotes weighted mode. $W.F.$ is an abbreviation for weighting factor.

Percentage of Total Argument Values Which Lie within Each j th Frequency Interval

$$P_j \% = \frac{n_j}{N} \times 100\%$$

Cumulative Percentage of Arguments Less than or Equal to the Upper Limit of Each Frequency Interval

For each j th frequency interval

$$\text{Cumulative percentage} = c_j \% = \frac{\sum_{i=1}^j n_i}{N} \times 100\%$$

Cumulative Remainder of Argument Values Greater than the Upper Limit of Each Frequency Interval

For each j th frequency interval

$$\text{Cumulative remainder} = r_j \% = \frac{\sum_{i=j+1}^N n_i}{N} \times 100\%$$

Upper Limit of Each Frequency Interval as a Multiple of Mean Argument Value

For the upper limit of each j th frequency interval

$$\text{Multiple of mean} = m_j = \frac{\text{Upper limit of } j\text{th frequency interval}}{\text{Mean value of Table arguments}} = \frac{k_B + (j-1)k_C}{\sum_{i=1}^N x_i / N}$$

The above normalization of the argument values as multiples of the mean value permits the analyst to determine whether the probability distribution belongs to the exponential-Erlang family of distributions. These analytic distributions are themselves generally normalized as multiples of a mean value of 1.0.

Upper Limit of Each Frequency Interval as a Deviation from Mean Argument Value

For the upper limit of each j^{th} frequency interval

$$\begin{aligned} \text{Deviation from mean} &= dj = \frac{\text{Upper limit of } j^{\text{th}} \text{ frequency interval} - \text{Mean value of Table arguments}}{\text{Standard deviation of argument values}} \\ &= \frac{[k_B + (j-1)k_C] - \bar{X}}{s} \end{aligned}$$

The above normalization of the argument values as multiples of the standard deviation from the mean value permits the analyst to determine whether the probability distribution is like a normal distribution. The analyst can compare a normal distribution, which has a mean \bar{x} and a standard deviation s , with the standard deviation of 1. This is due to the fact that, if $x_i \sim N(\bar{x}, s)$, then

$$\frac{(x_i - \bar{X})}{s} \sim N(0, 1),$$

where the general form $RV \sim N(a, b)$ means "a random variable RV is distributed normally with mean a and standard deviation b ."

Average Value of Overflow

Whenever an argument value falls in the last (k_D^{th}) frequency interval of a Table two steps occur:

1. The entry count in the last additional storage word is incremented by one.
2. The argument value is added to the sum of overflow values which is maintained in Table word D10.

Whenever one or more argument values do fall into this last frequency interval, the following line is printed at the bottom of the Table statistics: AVERAGE VALUE OF OVERFLOW, where:

$$\text{Average Value of Overflow} = \frac{\text{Sum of overflow argument values}}{\text{Number of overflow entries}}$$

EFFECT OF RESET AND CLEAR CARD ON TABLE STATISTICS

Table 29 summarizes the effects of RESET and CLEAR cards on the statistics which are gathered for each Table entity. A RESET card zeros out the sum of argument values (word D1), the sum of squared argument values (word D2), and the number of entries (word D6). The temporary storage words D5 and D12, which are used by Tables operating in the RT, IA, and difference modes, are left unchanged.

A CLEAR card also zeros out words D1, D2, and D6, and in addition it zeros out words D5 and D12. It should be observed that RESET and CLEAR cards do not affect the true long-run average values of any of the TABLE statistics. Tables differ in this respect from facilities, storages, and queues whose printed "Average Time per Transaction" statistics may be less than the true long-run average times because of RESET Card operations.

REDEFINITION OF TABLES

If a Table is redefined by another TABLE definition card during a simulation run, the GPSS/360 COMMON core blocks that were originally allocated to the Table are freed and the necessary words to store the Table frequency classes will be obtained from the words in COMMON storage.

TABLE 28: STATISTICAL PRINTOUT FOR TABLE ENTITIES

TABLE XTIME ENTRIES IN TABLE 2000		MEAN ARGUMENT 842.645	STANDARD DEVIATION 553.000	SUM OF ARGUMENTS 1685291.000	NON-WEIGHTED	
UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIATION FROM MEAN
0	0	.00	.0	100.0	-.000	-1.523
100	0	.00	.0	100.0	.118	-1.342
200	0	.00	.0	100.0	.237	-1.162
300	1	.04	.0	99.9	.356	-.981
400	218	10.89	10.9	89.0	.474	-.800
500	268	13.39	24.3	75.6	.593	-.619
600	391	19.54	43.8	56.1	.712	-.438
700	202	10.09	53.9	46.0	.830	-.257
800	162	8.09	62.0	37.9	.949	-.077
900	157	7.84	69.9	30.0	1.068	.103
1000	104	5.19	75.1	24.8	1.186	.284
1100	92	4.59	79.7	20.2	1.305	.465
1200	75	3.74	83.4	16.5	1.424	.646
1300	42	2.09	85.5	14.4	1.542	.827
1400	47	2.34	87.9	12.0	1.661	1.007
1500	27	1.34	89.2	10.7	1.780	1.188
1600	39	1.94	91.2	8.7	1.898	1.369
1700	33	1.64	92.8	7.1	2.017	1.550
1800	25	1.24	94.1	5.8	2.136	1.731
1900	20	.99	95.1	4.8	2.254	1.912
2000	15	.74	95.8	4.1	2.373	2.092
2100	12	.59	96.4	3.5	2.492	2.273
2200	10	.49	96.9	3.0	2.610	2.454
2300	8	.39	97.3	2.6	2.729	2.635
2400	4	.19	97.5	2.4	2.848	2.816
2500	5	.24	97.8	2.1	2.966	2.997
2600	3	.14	97.9	2.0	3.085	3.177
2700	5	.24	98.2	1.7	3.204	3.358
2800	1	.04	98.2	1.7	3.322	3.539
2900	4	.19	98.4	1.5	3.441	3.720
3000	6	.29	98.7	1.2	3.560	3.901
3100	2	.09	98.8	1.1	3.678	4.082
3200	3	.14	99.0	.9	3.797	4.262
3300	3	.14	99.1	.8	3.916	4.443
3400	1	.04	99.2	.7	4.034	4.624
3500	2	.09	99.3	.6	4.153	4.805
3600	3	.14	99.4	.5	4.272	4.986
3700	2	.09	99.5	.4	4.390	5.167
3800	3	.14	99.7	.2	4.509	5.347
3900	0	.00	99.7	.2	4.628	5.528
4000	1	.04	99.7	.2	4.746	5.709
4100	0	.00	99.7	.2	4.865	5.890
4200	2	.09	99.8	.1	4.984	6.071
4300	0	.00	99.8	.1	5.102	6.251
4400	1	.04	99.9	.0	5.221	6.432
4500	0	.00	99.9	.0	5.340	6.613
4600	0	.00	99.9	.0	5.458	6.794
4700	0	.00	99.9	.0	5.577	6.975
4800	0	.00	99.9	.0	5.696	7.156
4900	0	.00	99.9	.0	5.815	7.336
5000	0	.00	99.9	.0	5.933	7.517
5100	0	.00	99.9	.0	6.052	7.698
5200	0	.00	99.9	.0	6.171	7.879
5300	0	.00	99.9	.0	6.289	8.060
5400	0	.00	99.9	.0	6.408	8.241
5500	1	.04	100.0	.0	6.527	8.421

TABLE 29: EFFECT OF RESET AND CLEAR CARDS ON TABLE ATTRIBUTES

RESET Card

Word	Field	Attribute Value Before RESET Card	Result of RESET CARD on Attribute Value	
		Each frequency interval word in COMMON storage.	Number of argument values which fall in the frequency interval	Set to zero
D1	Fullword	Sum of argument values	Set to zero	
D2	Fullword	Sum of squared values of argument.	Set to zero	
D3	Fullword	Sum of weighted values of argument.	Set to zero	
D4	Fullword	Sum of squared weighted values of argument.	Set to zero	
D5	Fullword	Temporary storage for RT, IA and difference mode Tables.	Following values are unchanged: 1. Last value of argument for Table operating in difference or IA mode. 2. Sum of the number of units that have arrived for Table operating in the RT mode.	
D6	Fullword	Number of entries.	Set to zero	
D7	Fullword	Mode indicators and width of Table frequency classes.	Unchanged	
D8	Fullword	Address of COMMON where frequency classes are located.	Unchanged	
D9	Fullword	Table argument or arrival rate time in RT mode.	Unchanged	
D10	Fullword	Sum of overflow in last frequency class.	Unchanged	
D11	Halfword	Number of frequency classes.	Unchanged	
D12	Halfword	Nonweighted sum of units arrived for RT mode.	Set to zero	
D13	Fullword	Upper limit of lowest frequency class.	Unchanged	

CLEAR Card

The CLEAR card is the same as the RESET card (D1, D2, D3, D4, and D6 are set to zero; D7, D8, D9, D10, D11, and D13 are unchanged) except for:

Word	Field	Attribute Value before RESET Card	Result of RESET CARD on attribute Value
D5	Fullword	Temporary storage for RT, IA, and difference mode Tables.	Set to zero
D12	Halfword	Nonweighted sum of units arrived for RT mode.	Set to zero

CHAPTER 14: STATISTICAL PRINTOUT BLOCKS

GPSS/360 has three block types whose functions are to provide additional statistical output on the normal output device. The first is the PRINT block, which can initiate a partial output of the standard GPSS/360 statistics. The other two are the TRACE and UNTRACE blocks, which flag and unflag transactions so that data on each individual block move is printed out.

PRINT BLOCK

2 LOC	7	8 OPERATION	19 A	25 B	31 C	37 D	A - B
		PRINT	Lower index number j k _l	Upper index number j k _u	Entity Mnemonic	Paging indicator	PRINT C

The PRINT block serves to initiate a partial output of the standard GPSS/360 simulation statistics. Field C contains a mnemonic code which specifies the output desired:

Field C Mnemonic	Output
MOV*	Current events chain
FUT*	Future events chain
I*	Interrupt chain
MAT*	Matching status chain
C*	Relative and absolute clock time
B or N or W*	Block counts
S	Storage statistics
Q	Queue statistics
F	Facility statistics
U	User chain statistics
T	Table statistics
X or blank	Contents of fullword savevalue
XH	Contents of halfword savevalue
MX	Contents of fullword matrix savevalue
MH	Contents of halfword matrix savevalue
LG	Status of logic switches
CHA	User chain listing


The options not marked by '*' should have fields A and B specified. If not, the entire range of the particular entity involved will be printed. A single set of statistics may be printed by placing the same number in both fields, e.g., PRINT 106, 106, S.

Any Standard Numerical Attributes or indirect addressing is permitted at the PRINT block in fields A and B.

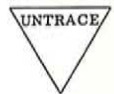
Field D contains a paging indicator. If this field contains any alphameric character, a page will not be skipped preceding each PRINT operation. If the field is left blank, a page will be skipped before each PRINT operation.

TRACE AND UNTRACE BLOCKS

2 LOC	7	8 OPERATION	19 A
		TRACE UNTRACE	



TRACE



UNTRACE

In order to debug and test the validity of a block diagram, it is necessary to verify that the desired paths are being followed by transactions. The TRACE block has been provided so that the progress of selected transactions may be recorded. The UNTRACE block serves to stop the tracing of a transaction.

When a transaction enters a TRACE block, a tracing indicator is set in the transaction. (Internally, bit 2 of Transaction byte T9 is set to one.) The UNTRACE block unconditionally removes the tracing indicator (the bit is reset to zero). Whenever a flagged transaction succeeds in entering a new block, two lines of output are produced. The first line indicates:

1. The transaction number
2. The previous block number
3. The new block number
4. The clock time
5. The remaining run termination count

The following is a sample of the first line of the TRACE output:

```
TRANS 1 FROM 1 TO 2 CLOCK 2 TERMINATIONS TO GO 5
```

The second line of output is the normal transaction printout for the TRACed transaction as described in Table 10 in Chapter 7.

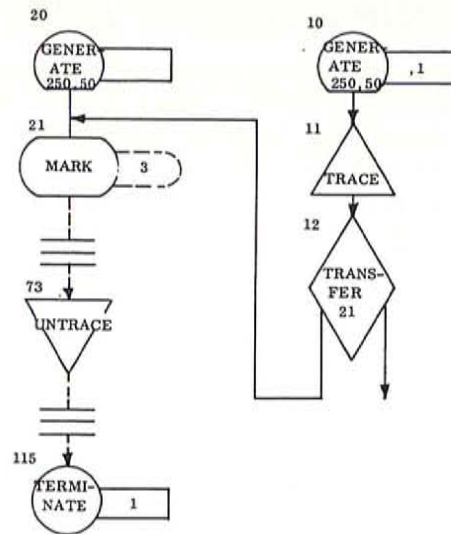
It is permissible to enter several TRACE blocks before an UNTRACE, and also to enter an

UNTRACE when the tracing flag is not set. Entries into a TRACE block are traced, while entries into an UNTRACE block are not. The TERMINATE block is the only other block that cannot cause a tracing line to be printed. All output will appear on the output device in chronological order, so that a complete trace for a single transaction may be printed in several disconnected sections. It is even possible for several traces which are taken for one transaction at one clock instant to be separated by other output.

TRACE and UNTRACE blocks should be used with great caution as each block move involves writing on the output device. The write operation may, however, be overlapped with the execution times for untraced transactions.

Example:

The following procedure will permit a single transaction to be traced through a path of the block diagram that contains many transactions. The UNTRACE block in the main stream will have no effect on the transactions which have not been flagged for tracing.



1	2	7	8	13	19	25	31	37	
*	LOC		OPERATION		A	B	C	D	
*	FORM SINGLE TRACING TRANSACTION								
	10		GENERATE		250	50		1	
	11		TRACE						
	12		TRANSFER			21			
*	MAIN DIAGRAM								
	20		GENERATE		250	50			
	21		MARK		3				
*	AND SO ON -- ALL BLOCKS IN SEQUENCE WILL BE TRACED								
	73		UNTRACE						
*	TRACING OF ANY FLAGGED TRANSACTIONS WILL BE ENDED								
	115		TERMINATE		1				

CHAPTER 15: CONTROL CARDS

GPSS/360 simulation models are controlled by the nine control cards listed below:

1. **START Card** -- indicates to the GPSS/360 program that the current set of input cards has been received and that the next simulation run should start. The START Card also specifies the length of the next simulation run.
2. **RESET Card** -- causes various accumulated statistics to be set to zero.
3. **CLEAR Card** -- causes various accumulated statistics to be set to zero and removes all transactions from the model.
4. **JOB Card** -- performs all the functions of a CLEAR Card; it also removes all block, function, arithmetic variable, table, and storage definitions.
5. **END Card** -- should be the last card of an input deck, which may contain one or more jobs.
6. **JOBTAPE Card** -- allows the user to read in transactions from a previously prepared transaction tape.
7. **REWIND Card** -- allows the user to rewind the transaction tape. It is used in conjunction with the WRITE block and JOBTAPE Card.
8. **LIST/UNLIST Cards** -- are used to delete strings of blocks from the output listing.
9. **READ/SAVE Cards** -- enable the user to save the current state of a model and then continue processing at this point at some later time.

START CARD

2	LOC	7	8	OPERATION	19	A	B	C	D
				START		Run Termination count	NP = Suppress printout.	Snap Interval K_C	1 = Standard Transaction printout also

The START Card has two primary functions:

1. It signals the GPSS/360 program that the current set of input cards has been received, and that the running phase should begin. The overall GPSS/360 scan will return to the start of the current events chain and once again begin to move transactions through blocks.
2. The total run termination count for the next simulation run is supplied by the field A constant. During the next simulation run, transactions should enter one or more TERMINATE blocks whose field A arguments specify a run termination count. If this does not happen, the simulation model will run indefinitely until removed from the computer by the operators. These values are subtracted from the total count. When the remaining termination

count has been reduced to zero or less, the simulation run ends and the GPSS/360 program proceeds into two phases:

- a. Statistical Output Phase

Fields B and D of the START Card control the type of statistical output.

- b. Input Phase

Further definition, control, and remarks cards are read in until either another START Card or an END Card is encountered.

The run termination count may be decremented to zero in the middle of the overall scan. The GPSS/360 program immediately transfers into the output and input phases. If another START Card but no CLEAR Card is read in, the GPSS/360 transactions continue processing, just as if the output and input phases had never occurred.

Print Suppression (field B)

NP in field B suppresses the printout of statistics at the end of the run. This is often desirable at the end of an initialization run when the model is being brought into a "steady state." Any entry other than NP will have no effect.

Snap Interval (field C)

Field C sets up a snap interval count, which is also decremented by the field A values of TERMINATE blocks. Whenever the snap interval count has been reduced to zero or less, a normal statistical printout occurs. The snap interval count will be reinitialized to its original value and the decrementing process will begin again. This process is repeated until the field A terminate count is decremented to zero. With the snap feature, statistics can be obtained during a simulation run as well as at the end.

Examples:

```
START 500,,100
```

Output statistics will be given every 100 terminations. There will be five such sets of statistics labeled SNAP 1 OF 5, SNAP 2 OF 5, etc.

```
START 500,,150
```

Output statistics will be given after 150, 300, 450 and 500 terminations.

```
START 10,,1
```

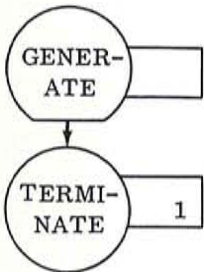
Output statistics will be given after every termination. There will be ten such sets of statistics.

Transaction Printout Option (field D)

A 1 in field D of the START Card indicates that each statistical printout (either at a snap interval or at the end of a run) will also include a transaction printout of the current events chain, future events chain, interrupt transactions, user chain, and transactions in matching status.

Control of Running Time of a Simulation Model

In many cases it is desirable to control the time duration of simulation runs. A simple two-block timer, shown below, will permit this. The TERMINATE Block has 1 coded in field A. All other TERMINATE blocks in the model have blank fields A. The run termination count (k_A) in field A of the START Card defines the length of the simulation as: $RUN\ TIME = k_A \times m$ (GENERATE block mean time).



When such a run timer is used, the number of actual transactions TERMINATED in the main model can be a random variable, while the run time is a constant. The alternative run control method is to have the actual transactions TERMINATE in blocks that contribute toward the run termination count, which is specified by field A of the START Card. In this case the number of transactions which are terminated during the simulation can be a constant, while the running time becomes a random variable.

Example:

```
GENERATE    100
TERMINATE   1
START       1
```

If all other TERMINATE blocks in the associated model had blank fields A, the run timer shown above would cause simulation to end at clock time 100. If the START Card had a 10 in field A, simulation would end at clock time 1000.

RESET CARD

The RESET card, first of all, sets the relative clock time to zero. Therefore, the Standard Numerical Attribute (C1) is equal to zero. The value of the absolute clock time remains unchanged. The seeds of the eight random number generators are not reset.

The RESET Card affects the attributes of six GPSS/360 entities, as outlined in the following Tables:

- | | |
|----------------------------------|---------------------|
| 1. Block Attributes | Table 17 Chapter 6 |
| 2. User Chain Attributes | Table 13 Chapter 7 |
| 3. Facility Attributes | Table 20 Chapter 10 |
| 4. Storage Attributes | Table 23 Chapter 11 |
| 5. Queue Attributes | Table 26 Chapter 12 |
| 6. Distribution Table Attributes | Table 29 Chapter 13 |

The values in savevalue locations are not altered, nor are the reset or set conditions of the logic switches.

Total block counts (N_j) are set to zero, but the current block counts (W_j) are unchanged. Facility cumulative time integrals are set to zero. The entry count is altered as described in Table 20, Chapter 10.

Storage cumulative time integrals are set to zero. The storage entry count and maximum contents are both set equal to the current storage contents. Queue cumulative time integrals are set to zero. The Queue total entry count and maximum contents are both set equal to the current Queue contents. The following accumulated table statistics are all set to zero: number of values in each frequency interval, total number of entries, sum of argument values, sum of squared values, sum of weighted values, sum of squared weighted values. User chain cumulative time integrals are set to zero. The chain entry count and maximum contents are both set equal to the current contents. The User Chain link indicators are reset to "off".

The RESET Card also allows the user to specify, beginning in column 19, certain entities or ranges of certain entities which the RESET operations will not affect. The following mnemonics must be used with the number of the entity indicated at the right:

- Fj - Facilities
- Qj - Queues
- Sj - Storages
- CHj - User Chains
- TBj - Tables

The selective reset option does not apply to block attributes.

The use of the selective reset feature can best be explained by a number of examples.

The card: RESET F1, F7, F10 would RESET all facilities except Facilities 1, 7, and 10.

The card: RESET F1-F10, F15 would RESET all facilities except Facilities 1 thru 10 (includes 1 and 10) and Facility 15.

The card: RESET F1, Q1, S1 would RESET all facilities, queues, and storages except Facility 1, Queue 1, and Storage 1. All tables and user chains would be RESET.

The card: RESET F1-F10, F15, Q1-Q10, Q15 would RESET all facilities and queues except Facilities and Queues numbered 1 through 10, and 15.

The following is a summary of the rules which govern the format of the Selective RESET Card.

1. All entries must be separated by commas.
2. If a range of an entity is specified, the upper and lower limits which are included in the range must be separated by a dash (-).
3. Entity types, i.e., facilities, storages, queues, etc., may not be intermixed. For example: RESET F1, Q1-Q5, F7, Q8
4. The entries for each entity must be in ascending order.
5. The last entry on the RESET Card must end by column 71. However, multiple RESET Cards may be used. If so, rules 1-4 apply just as if the second RESET Card were an extension of the first.
6. Most important: All the statistics mentioned at the beginning of this section will be RESET, except those for the entities specified in the operand field of the RESET Card(s).

Because the cumulative time integrals for facilities, storages, and queues are set to zero, the average time per transaction figure for queues, storages, and facilities is an approximation, since statistics may be RESET in the middle of a transaction's interval. This problem is discussed in further detail in the Chapters 10, 11, and 12 on facilities, storages, and queues. The printed average time result may be less than the "true" mean value. Average times which are obtained from a frequency distribution table or Qtable are, however, always exact, since incomplete intervals are not tabulated.

The RESET Card is effective immediately, and input cards which are encountered after the RESET Card are not subject to its effect.

CLEAR CARD

The CLEAR Card, first of all, sets both the relative clock time (Standard Numerical Attribute C1) and the absolute clock time to zero. The seeds of the eight random number generators are not reset. All transactions in the model are destroyed, including:

1. Incipient transactions which are scheduled to enter GENERATE blocks
2. The special RT table operators

The CLEAR Card affects the attributes of six GPSS/360 entities, as outlined in Tables 7, 13, 20, 23, 26, and 29 (see "RESET Card" above).

In addition, all of the savevalue locations are set to zero, and all the logic switches are put back into their original reset condition. The CLEAR Card is immediately effective, and input cards which are encountered after the CLEAR Card are not subject to its effect. A CLEAR Card is not required before the first START Card of a model.

The facility, storage, queue, and user chain cumulative time integrals are set to zero, just as with the RESET Card. The current counts or contents of blocks (Wj), storages (Sj), and queues (Qj) are all set to zero. The entry counts (words F9, U5, S6, and Q2) and maximum contents (words S7, Q7, and U4) are set to zero. The same table statistics are set to zero as for the RESET Card, as well as the temporary storage word which is used by RT, IA, and difference mode tables.

The CLEAR Card also allows the user to specify certain savevalues or ranges of savevalues which the CLEAR Card will not set to zero. The format for the Selective CLEAR is very similar to that of the Selective RESET described above. The only two mnemonics which are accepted on the CLEAR Card, beginning in column 19, are Xj (fullword savevalues) and XHj (halfword savevalues). The entries again must be in ascending order and fullword (Xj) and halfword (XHj) may not be intermixed.

Examples:

CLEAR X1, X3, X5-X10

All fullword savevalues will be set to zero except 1, 3, and 5 through 10. All halfword savevalues will be zeroed.

CLEAR X1-X10, XH1-XH10

All fullword and halfword Savevalues will be set to zero except fullword and halfword savevalues 1 through 10.

Recreating Transactions at GENERATE Blocks

After performing all of the above CLEARing operations, the GPSS/360 program examines the

model for GENERATE blocks. It creates a new transaction at each GENERATE block in the model, just as if the GENERATE block definition card had just been read in. The field C offset time is recomputed. The field D creation limit and the field E priority level of the GENERATE block card, are stored in the fourth and third words of COMMON for the GENERATE block, respectively. The creation limit is placed in the mark time word (T5) of the newly created transaction, while the priority level is stored in byte T7.

Recreating the Arrival Rate (RT) Table Operators

Each table which operates in the arrival rate (RT) mode has a special transaction operator associated with it. The CLEAR Card eliminates these operators. After all the CLEARing operations have been completed, the GPSS/360 program examines the model for RT mode tables. It recreates the transaction operators in the same manner that it recreates the GENERATE block transactions. The first byte of word D7 of each table is examined for the bit which indicates that the table is operating in the RT mode. A new transaction operator is recreated for each such RT table. The first arrival time of the operator is obtained from word D10, which contains the intertabulation time which was originally specified in field E of the TABLE Definition card.

JOB CARD

A JOB Card should be inserted between successive simulation models which are submitted as parts of the same run. It performs all the functions of the CLEAR Card, and it also removes all block, function, variable, table, and storage definitions. Different analysts can, therefore, run different models by simply including JOB Cards between their models. No JOB Card is necessary before the first job of a run. The format for the JOB Card is as follows:

1	2	8	19
\$		JOB	

The JOB Card is the only GPSS/360 input card which may contain a \$ in column 1. However, the \$ is optional and may be omitted.

END CARD

This card should be the last card of an input deck, which may of course contain many jobs. The END Card results in return of control to the system monitor after the execution of GPSS/360.

JOBTAPE CARD

The function of the JOBTAPE Card is to set up a transaction tape, previously prepared by WRITE blocks and/or other means, as input to a specified block in a subsequent simulation run. The JOBTAPE Card may also be used to skip files on transaction tapes. The format for the JOBTAPE card is as follows:

8	19
JOBTAPE	A,B,C,D

where:

A = Tape name (JOBTA1, JOBTA2, JOBTA3)

B = Next Block for Transactions

C = Offset before entry of first Transaction

D = Multiplier

Note: Fields C and D may contain constants of 1-999999.

Operation of the JOBTAPE Card

Fields B, C, and D are blank:

The tape specified by field A of the JOBTAPE Card will skip over a file until an EOF indicator is encountered.

Entries in fields B, C, and D:

1. The program obtains the tape name from field A of the JOBTAPE Card.

2. Transactions from the selected tape will be entered into a block in the model whose block symbol or number is given in field B of the JOBTAPE Card. Field B may refer to any legal block symbol or number which does not denote a GENERATE block. The interval of time separating the entry of each transaction from that of its predecessor will be the interarrival time recorded with the entering transaction.

3. Field C of the JOBTAPE Card specifies an offset time which will separate the start of simulation and the entry of the first transaction from the tape. If field C is left blank or coded zero, the first transaction will enter at time 1.

4. Field D of the JOBTAPE Card contains a scaling factor by which the interarrival time and transit time may be multiplied. Permissible factors are 1 to 999999, with zero and blanks being treated as 1.

5. Each transaction entered from a tape is treated as an independent assembly set, just as though it had come from a GENERATE block. This will be true even if some of the taped transactions constitute sets created by SPLIT blocks.

6. When an end-of-file indicator is encountered on a transaction tape, the program ceases to read the tape until a new JOBTAPE Card referring to the same tape drive is encountered.

GPSS/360 control cards which affect tapes that are being used as JOBTAPES are the END, JOB, and CLEAR Cards. The END and JOB Cards will cause all tapes which are being used as JOBTAPES to be rewound and unloaded. The CLEAR control card will cause all tapes being used as JOBTAPES which have not encountered an end-of-file indicator, to skip records until an end-of-file indicator is encountered. If an EOF was previously recognized, no skipping will take place.

REWIND CARD

The function of the REWIND Card is to rewind the tape specified in field A of the REWIND Card. This control card is used in conjunction with the WRITE block and the JOBTAPE Card. The format for the REWIND Card is as follows:

```

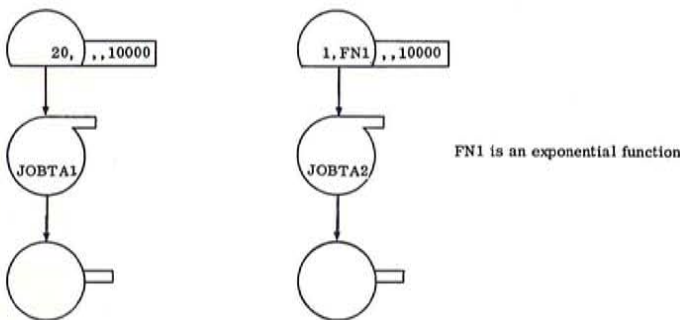
8           19
REWIND     JOBTA1
           JOBTA2
           JOBTA3
  
```

Operation of the REWIND Card

The tape specified by the field A of the REWIND Card will be unconditionally rewound to its load point regardless of its current status.

Example 1:

Write 10000 transactions on JOBTA1 and JOBTA2. The transactions on JOBTA1 will have a constant interarrival time of 20 time units. The transaction on JOBTA 2 will have interarrival times exponentially distributed.



Example 2:

The following sequence would skip two files on JOBTA1. The transaction of the third file would be sent to block ABC starting at time equal 1.

```

JOBTAPE   JOBTA1
JOBTAPE   JOBTA1
JOBTAPE   JOBTA1, ABC
  
```

LIST/UNLIST CARDS

The format for LIST/UNLIST Cards is as follows:

```

8
-----
LIST
UNLIST
  
```

Operation of the LIST/UNLIST Cards

The UNLIST Card is used to delete from the output listing all cards following the UNLIST Card. The LIST Card will resume the output listing of subsequent cards. The LIST/UNLIST Cards are applicable to the output of the GPSS/360 assembly program and the card listing of the GPSS/360 program. The LIST/UNLIST Cards do not affect the normal statistical output of the GPSS/360 program.

Example:

Block No.	Name	A
	8	19
	GENERATE	1, FN1
	UNLIST	
ABC	GATE NU	1
	SEIZE	1
	ADVANCE	10
DEF	RELEASE	1
	LIST	
	TERMINATE	

In the above model, blocks ABC through DEF inclusive will not be listed in the output of the GPSS/360 assembly or in the card listing of the GPSS/360 program.

GPSS /360 READ/SAVE FEATURE

The READ/SAVE feature of GPSS/360 allows the user to save the existing model and current statistics of a simulation run at a predetermined point, and then restart the simulation from that point at some subsequent time. To use the SAVE feature, the SAVE Control Card will be necessary. The SAVE Card has the following format:

```

8
-----
SAVE
  
```

When the SAVE Card is read by the GPSS/360 program, the model in its current state, along with all accumulated statistics, will be written on the device specified by the user (see GPSS/360 Operator's Guide) and the standard statistical output will be written on the normal output device.

When the SAVE Card is read in by the GPSS/360 program, two alternatives are possible:

1. Field A of the SAVE Card is blank

The READ/SAVE tape will be rewound. The model in its current state, along with all accumulated statistics, will be written as a single (1) file on the READ/SAVE tape. The GPSS/360 program will then read the card which follows the SAVE Card.

2. Field A contains an alphanumeric character

The model in its current state, along with all accumulated statistics, will be written on the READ/SAVE tape. The GPSS/360 program will then read the card which follows the SAVE Card. In this alternative the READ/SAVE tape is not rewound before writing.

Example 1:

Save a model in its current state with the accumulated statistics at the termination of every 100th transaction for a total of 500 transactions. This could be accomplished by use of the following cards:

8	19
START	100
SAVE	
START	100
SAVE	2R
START	100
SAVE	3R
START	100
SAVE	4R
START	100

Note: In this example, the READ/SAVE tape would contain five files.

Example 2:

If in the above example it was desired to have the latest SAVE always the first file on tape, the same sequence of control cards could be used except that field A of the SAVE Cards would remain blank:

8	19
START	100
SAVE	
START	100
SAVE	

START	100
SAVE	
START	100
SAVE	
START	100
SAVE	

To continue the simulation run at some later time, it is necessary to use the READ Control Card. The format for the READ Card is:

8	19
READ	blank or some numeric character

The entry in column 19 of the READ Card depends upon the contents of the READ/SAVE device that is being used as input. If the user wishes to READ the first (or only) file on the READ/SAVE tape the field should be left blank. If, however, a subsequent file is desired the user must specify the number of files to be skipped until the appropriate file is reached.

Example 3:

The user wishes to READ the fifth file on the READ/SAVE tape created in Example 1. The following card sequence could be used:

8	19
SIMULATE	
READ	4
START	100
END	

Note: The READ Card must always be followed by a START Card with the desired transaction termination count in column 19 since the termination count was reduced to zero when the tape was created.

If the GPSS/360 reallocation feature was used in the original run (when the model was SAVED), the same reallocation must be specified before the SIMULATE Card in the input stream.

When using the READ Card to read in an input tape which was previously created using the SAVE feature, it is possible to redefine blocks, functions, variables, etc., before execution of the model begins. These input cards must follow the READ Card and precede the START Card and be coded in the absolute GPSS/360 format. An ABS mode - type run should be specified with ABS and ENDABS Cards rather than a normal GPSS/360 assembly run.

CHAPTER 16: DIFFERENCE BETWEEN GPSS III AND GPSS/360

The first section of this chapter lists and describes the modifications required to run a GPSS III model using GPSS/360 operating under OS/360. The second section summarizes and describes in general terms the additional features provided by GPSS/360. The reader is referred to the specific chapter(s) in this manual for a complete description of the entity and/or addition of interest.

In general an existing model written in the GPSS III language can be run under GPSS/360 by exchanging the GPSS III 7040 or 7090 IBSYS control cards for the appropriate OS/360 control cards. The OS/360 control cards required for the operation of GPSS/360 under OS/360 are listed and described in the GPSS/360 Operator's Guides (OS and DOS).

The only GPSS III block and associated GPSS III control cards which will require modification are the WRITE block, JOBTAPE and REWIND control cards. Each of these requires one of the following for the "A" argument; JOBTA1, JOBTA2, or JOBTA3.

If random numbers are referenced, the user should not expect exact equivalent results for a model run with GPSS III and the same model run with GPSS/360. For a satisfactory sample size, results should compare plus or minus a few percentage points. The discrepancy between the results is due to different random number seeds which are used in GPSS III and GPSS/360.

MODIFICATION OF GPSS III BLOCK TYPES

GENERATE Block

Field: A

Legal Arguments: X, XH, FN, V, and K
Interpretation: Mean interarrival time for transactions created at the GENERATE block.

Field: B

Legal Arguments: X, XH, FN, V, and K
Interpretation: Modifier of the mean time for transactions created at the GENERATE block. If FN is specified, the floating-point value of the mean will be multiplied by the floating-point value of the modifier and then truncated. This value will be the interarrival time for the transaction created at the GENERATE block. Any other legal SNA will be evaluated and interpreted as a spread, that is, \pm the mean value.

Field: C

Legal Arguments: X, XH, FN, V, and K
Interpretation: OFFSET -- This argument may be used to determine when the first created transaction will leave the GENERATE block. When a CLEAR card is encountered the SNA specifying the offset will be reevaluated to determine the new OFFSET (if any) to be associated with the first transaction created at the GENERATE block for the subsequent run.

Field: D

Legal Arguments: X, XH, FN, V, and K
Interpretation: COUNT -- This argument specifies the number of transactions which are to be created at the GENERATE block. As transactions leave the GENERATE block this count is decremented. When the count reaches zero no further transactions will be created at the GENERATE block. When a CLEAR card is encountered the SNA specifying the count will be reevaluated to determine the count for the subsequent run. If the D argument is blank, transactions will be created indefinitely at the GENERATE block.

Field: E

Legal Arguments: X, XH, FN, V, and K
Interpretation: PRIORITY -- This argument specifies the priority to be assigned to transactions created at the GENERATE blocks. The priority may be from 0 to 127. The SNA specifying the priority is evaluated each time a transaction is created, thereby enabling the user to dynamically assign priorities at a GENERATE block by modifying the SNA which determines the priority.

Field: F

Legal Arguments: X, XH, FN, V, and K
Interpretation: Number of parameters to be associated with transactions created at the GENERATE block. The SNA specifying the number of parameters is evaluated each time a transaction is created. This enables the user to change dynamically the number of parameters to be associated with a transaction by modifying the SNA which determines the number of parameters. If the F argument is blank, 12 halfword (16 bits) parameters will be associated with the transaction created at the GENERATE block.

Field: G

Legal Arguments: F, H, or Blank
Interpretation: The G argument specifies the type of parameters to be associated with transactions

created at the GENERATE block. H or blank will associate halfword (16 bits) parameters to the transaction. F will associate fullword (32 bits) parameters to the transaction.

Note: In the above description for the A, B, C, D, E, and F arguments, constants may be specified without using the character K. That is, 100 is equivalent to K100.

PREEMPT Block

The operation of the PREEMPT block has been extended so that it is now possible to interrupt a facility based on the priority of the transaction currently SEIZING or PREEMPTING the facility and the priority of the transaction entering the PREEMPT block.

Preemption based on the priority of a transaction is specified by placing PR in the B argument of the PREEMPT block.

To account for the many processing alternatives the preempted transaction could take, the following options are available to the preempted transaction:

1. Field C may specify some other block to which the preempted transaction will be sent. The preempted transaction will continue to be in contention for use of the facility.
2. If the preempted transaction is at an ADVANCE block, the remaining time the preempted transaction has at the ADVANCE block is calculated and placed in the parameter specified in field D of the PREEMPT block. The preempted transaction will be sent to some other block specified in field C. The preempted transaction will continue to be in contention for the facility.
3. If RE is coded in field E of the PREEMPT block, the operation will be the same as (1) above, except that the preempted transaction will no longer be in contention for the facility.

PRINT Block

The external format of the PRINT block will now allow the upper and lower limits of the desired entity statistics to be specified by SNA's. These are the A and B arguments of the PRINT block.

SAVEVALUE Block

When referencing halfword SAVEVALUE's at a SAVEVALUE block the user must specify an H in field C.

UNLINK Block

The operation of the UNLINK block has been extended so it is now possible to remove transactions from USER CHAINS based on the user chain transaction's ability to satisfy a Boolean variable. This mode of operation is specified by placing BVj in the "D" argument of the UNLINK block where j is the Boolean variable number.

MODIFICATION OF GPSS III CONTROL CARDS

STORAGE Definition Card

The operation of the STORAGE definition card has been expanded so that it is now possible to define the capacity of several STORAGES and/or range of STORAGES on a single STORAGE definition card.

Selective RESET Card

The operation of the RESET card has been expanded so that it is now possible to specify those entities which should not have their statistical words and areas RESET.

Selective CLEAR Card

The operation of the CLEAR card has been expanded so that the user may specify the SAVEVALUES which should not be set to zero.

GPSS/360 FEATURES

This section lists and describes in general terms the additional features provided by GPSS/360. The additional features are categorized into three areas:

1. GPSS/360 Assembly Program
2. New Entities and Block types
3. GPSS/360 Output Editor

GPSS/360 Assembly Program Extensions (See Appendix B for detailed descriptions)

a) The GPSS/360 assembly program provides symbolic addressing of entities. In general, wherever a specific entity type is implied by a block field, a symbol may now be used to reference the entity. In fields where the entity type is not implied, it will be necessary to precede the symbol with a character(s) which identify the entity type and a \$. The assembly program will assign

numbers to the entities, transmit the converted information to the simulation program, and maintain a cross-reference dictionary for each entity.

b) The GPSS/360 assembly program block symbol table includes a cross-reference dictionary. Following each symbol will be a list of the blocks in which the symbol was referenced. The output will also include a table of entities which were defined symbolically within a model. The symbols will be grouped according to entity type and the numerical value which was assigned to the symbol by the assembly program will appear following the symbol itself.

c) A GPSS/360 Assembly Macro Operation is provided which will permit parallel or nearly identical parts of programs to be coded only once and then used at any point in the model by inserting a card calling the desired macro definition. Each macro instruction may have up to 10 arguments associated with it. This will enable the user to vary the function of the macros each time the macro is called.

d) An UPDATE feature is provided which will enable the user to place symbolic models on tape and subsequently make modifications to the model by updating the master tape.

New Entities, Block Types, and Extended Features of GPSS/360

This section is subdivided into two parts. The first part deals with extensions of existing features of GPSS III. The second part lists and explains new entities and block types which are available in GPSS/360.

1. Extension of Existing Features:

- a. Arithmetic Entity and SNA Limitations.
Since the System/360 core consists of 32 bit words, the upward limit of entity attributes and System Numerical Attributes (SNA) is $2^{31}-1$.
- b. Parameters.
In GPSS III, parameters are unsigned quantities with a maximum value of $2^{15}-1$. In GPSS/360 the user has the ability to specify the type of parameters which will be associated with transactions

created at a GENERATE block. The parameters are signed and of two types, which are 2 bytes ($\pm 2^{15}-1$) (H) or 4 bytes ($\pm 2^{31}-1$) (F).

Only one type of parameter can be associated with a given transaction. The number and type of parameter to be associated with a transaction is specified by the F and G fields of the GENERATE block.

- c. Random Number Generators.
Eight random number generators completely independent of each other are provided. The SNA's associated with the generators are RN1, RN2, . . . RN8. This will allow the user to reference independent sources or random numbers from various sections of the model, thereby eliminating the dependence on a single source of numbers.
- d. Function Input Format.
GPSS/360 provides the user with the ability to specify the X and Y coordinates of functions in a free form format. This enables the user to specify more than six characters for any point, thereby increasing the maximum input coordinate values to $2^{31}-1$.
The fixed format specified by GPSS III will still be accepted.
- e. The following output statistics are provided for each user chain.
Maximum number of transactions to appear on the user chain.
Average number of transactions to appear on the user chain.
Total number of transactions which were placed on the user chain.
Average time a transaction was on the user chain.
- f. Variable Statements.
Variable statements allow the use of parentheses. Elements within the parentheses are evaluated before other arithmetic operations outside the parentheses take place (similar to FORTRAN).
- g. A transaction is allowed to enter multiple QUEUE blocks. A single transaction

is able to gather multiple QUEUE statistics by passing the transaction through a sequence of up to five QUEUE blocks and subsequent DEPART blocks. ADVANCE blocks are also allowed between QUEUE and DEPART blocks without destroying queue statistics. This provides a means of obtaining multiple queue statistics, i. e. , queuing up for several items of equipment simultaneously.

- h. To provide further convenience and versatility, additional statistical System Numerical Attributes (SNA) are provided.

TRANSACTIONS

PR Priority of the transaction currently active.

FACILITIES

FRn Utilization of Facility n in parts per thousand. For example, if the utilization were .88, the value of FRn would be 880.

FCn Number of entries for Facility n.

FTn Average time each transaction used facility n. When referenced, the computed value will be truncated. For example, if FTn were 1.23, the computed value would be 1.

STORAGES

SRn Utilization of Storage n in parts per thousand. For example, if the utilization were .65, the value of SRn would be 650.

SA n Average contents of Storage n (truncated).

SMn Maximum contents of Storage n.

SCn Number of entries for Storage n.

STn Average time each transaction used Storage n. When referenced, the computed value will be truncated.

GROUPS

Gn Number of items in Group n.

USER'S CHAINS

CAn Average number of transactions on User Chain n (truncated).

CHn Current number of transactions on User Chain n.

CMn Maximum number of transactions on User Chain n at any one time.

CCn Total number of transactions which were on User Chain n.

CTn Average time each transaction was on User Chain n. When referenced, the computed time will be truncated.

QUEUES

QAn Average contents of Queue n (truncated).

QMn Maximum contents of Queue n.

QCn Number of entries in Queue n.

QZn Number of zero entries in Queue n.

QTn Average time each Transaction was on Queue n (including zero entries). When referenced, the computed time will be truncated.

TABLES

TCn Number of entries in Table n.

TDn Standard deviation of Table n.

2. New Entities:

- a. To increase the logical power and capabilities of GPSS, Boolean variable statements are provided. This makes it possible to make decisions at a single GPSS block based on the status and value of many GPSS entities.

The elements which make up the Boolean variable will be interpreted as a one if nonzero and zero if zero. Conditional statements will be allowed as elements of the Boolean variable. For example, (X10'G'500) would be interpreted as a one if the contents of SAVEVALUE 10 was greater than 500 and a zero if less than or equal to 500.

Parentheses, indirect addressing, and and the Boolean operations AND (*) and OR (+) will be allowed in Boolean variable statements.

- b. MATRIX SAVEVALUES provide the ability for the user to associate additional attributes to GPSS entities such as facilities, storages, logic switches, user chains, etc.

Matrix Savevalues may be fullwords (32 bits) or halfwords (16 bits). The Matrix Savevalue will be expanded, as specified by the user, to a M x N matrix where M is the number of rows and N is the number of columns.

- c. To establish a means of grouping transactions and/or other entities, a new entity type, GROUP, is provided. A GROUP is basically a list of numbers. The interpretation and meaning of a Group is dependent on what elements constitute a Group and how the analyst creates, manipulates and removes members of a Group within the model.

A Group and reference to members of a Group is completely independent of the status of the members which make up the Group. If transactions made up a Group, they could all be referenced whether they were on the future, current, interrupt, or user chain.

- d. Two new block types provide a means for COUNTing and SELECTing based on the status of GPSS entities. This eliminates the necessity for many complicated block sequences formerly required to perform a similar function. The COUNT block has the ability to count the quantity of a given entity type which meets a specified condition, for instance the number of facilities not in use. The SELECT block will test the same entities and status as those specified for the COUNT block. The operation of the SELECT block is such that it will select the first entity which meets the specified condition.

GPSS/360 Output Editor

The GPSS/360 output editor provides a means of editing and specifying a simulation output format which will be most meaningful to the user. The output editor will provide three different options:

- Standard Output
- User-Specified Output
- Graphical Output

With all three options, all entities defined by symbolics, except blocks, will have the symbolic name listed in the GPSS/360 output rather than the numeric value assigned by the GPSS/360 assembly program.

The standard output is similar to that provided by GPSS III. The second option allows the user to include titles in various portions of the output, to select the entity statistics which should be listed, or to completely determine the format and contents of the output by specifying the alphameric information and entity statistics which are desired.

The third option enables the user to obtain graphical output which provides a pictorial representation of GPSS/360 entity statistics on the system printer. Each graph may be considered as consisting of a 60 row by 132 column matrix. By use of the appropriate control cards the user will have the ability to:

- a. specify the origin for the X and Y axis.
- b. specify the entity SNA to be plotted.
- c. specify the range of the entity type, for example, Facility 1 through Facility 10.
- d. specify the values to be assigned to the X and Y ordinate points.
- e. list alphameric information on the graph.

CHAPTER 17: PRACTICAL SUGGESTIONS ON THE USE OF GPSS/360

This chapter includes a series of practical suggestions on how to use GPSS/360. These hints are presented in the typical time order of developing a model: hand coding, initial debugging runs, classic running errors, analysis of GPSS statistics, and statistical problems in simulation.

HAND CODING

1. The following characters should be carefully hand coded to avoid confusing them with each other:

- a. Number 4 (not 4) vs Number 9
- b. Number 0 vs Letter O
- c. Number 1 vs Letter I

2. The most common assembly program error is number 11 (this card contains an undefined symbol). The following two common causes should be avoided:

- a. The analyst forgets to code a block with a symbolic location that he uses elsewhere in a block variable field; as a Yi value of a function follower card; or as the symbolic block index of the two Standard Numerical Attributes N\$xxxxx and W\$xxxxx.
- b. The analyst codes a block symbolic location (columns 2-6) differently from the symbolic location that he uses elsewhere. The second symbol results in assembly program error number 11. The first symbol is printed out in the assembly program symbol table which follows the assembly program input listing. However, the cross-reference dictionary will contain no references to the symbol. The analyst should, therefore, be as alert in discovering such unreferenced symbols as he is in discovering undefined symbols.

3. Each important part of a simulation model should be identified and headed by a remarks card: an asterisk* in column 1. Individual cards should also be liberally commented. The input card listing will be much neater if these comments always begin in the same card column, e.g., column 49.

4. Left-justify all card names in the operation field to column 8.

5. The successive Xi values of function follower cards must be monotonically increasing, i.e., $X_i < X_{i+1}$. A very common coding error, when

using the random number RN1 as a function argument, is to incorrectly calculate or hand code the fractional values between 0 and 1. A later Xi value may then be less than an earlier one.

6. The number of GPSS/360 COMMON storage words used in a model should not exceed the available words. The following GPSS/360 COMMON storage requirements should be remembered.

- a. One word per table frequency interval.
- b. List-type functions — one word per function point. Discrete-type Functions — two words per Function point. Continuous-type functions — three words per Function point.

7. An END card should be placed at the end of the assembly program input deck.

INITIAL DEBUGGING RUNS

1. It is extremely desirable to provide about six RESET and START cards during initial debugging.

	A,	B,	C,	D,
START	n_1 ,	,	n_2 ,	1
RESET				
START	n_1 ,	,	n_2 ,	1
RESET				
START	n_1 ,	,	n_2 ,	1
RESET				
START	n_1 ,	,	n_2 ,	1
RESET				
START	n_1 ,	,	n_2 ,	1
END				

The termination count n_1 in field A of the START card should be relatively low, so that each simulation run covers a short period of time. The development of error conditions can therefore be traced through a series of statistical printouts covering short time periods. Remember that each simulation run ends as soon as an error stop occurs. The RESET cards which precede each START card (except the first) wipe out the statistics from the preceding run. By thus avoiding a cumulative build-up of statistics it may be possible to spot unusual trends in the series of individual run statistics. Perhaps a facility suddenly has zero utilization or 100 percent.

utilization in a series of runs. This may mean that transactions are somehow failing to enter SEIZE or PREEMPT blocks referencing the facility; or, conversely, a transaction which has SEIZED or PREEMPTed the facility is blocked from moving into a RELEASE or RETURN block.

2. The START card snap option should be liberally employed during debugging runs. The C field of the START card specifies the snap interval in termination count units. During debugging the number of units, n_2 , should be less than the field A run termination count, n_1 . The standard GPSS statistics can, therefore, be obtained several times within each START card simulation run.

3. Field D of each START card should contain a 1. This means that a complete transaction printout will be obtained with each set of statistics. It is imperative for the analyst to justify that each and every transaction is in the proper block with the proper set of attribute values. Efficient debugging of GPSS/360 models depends most importantly on a complete understanding of the transaction printout.

4. An incomplete model can be inputted to the assembly program simply to catch coding and format errors, and possibly undefined or unreferenced symbols.

5. It may be possible to decompose a model into independent parts which can be debugged separately.

CLASSIC ERRORS IN GPSS/360 III MODELS

Appendix A lists all possible running errors which can occur in GPSS models. The following is a summary of the most common errors:

1. Although indirect addressing *n of entity indices is one of the most powerful features in GPSS/360 models, several types of errors may occur. First, transaction parameter n may have a zero value, or its value may exceed the maximum index value for the particular type of entity being addressed. Secondly, if arithmetic variable or function values (V*n or FN*n) are referenced, then each possible entity must have been defined by a VARIABLE or FUNCTION card.

2. A transaction entering a LEAVE block may remove more storage units than the current storage contents (Execution error 425). A transaction entering a DEPART block may similarly remove more queue units than the current queue contents (Execution error 428).

3. A transaction which did not SEIZE or PREEMPT a facility attempts to RELEASE or

RETURN the facility (Execution Errors #415 and #421).

4. A transaction can get trapped in a zero-time loop of blocks. The transaction will continue to move around endlessly until an operator terminates the simulation.

ANALYSIS OF GPSS/360 STATISTICS

The most ominous moment in the debugging of a GPSS/360 model occurs when no further input errors or running error stops occur. At this point, each simulation run yields the standard set of GPSS/360 statistics. The big question now is whether the model is correctly simulating the real world or whether there are some logical flaws in the model. The following are some clues for which the analyst should look in the GPSS statistics as signs of possible logical errors in the model. Assume that RESET cards are used before each START card, so that an individual set of statistics is obtained for each simulation run, rather than building up cumulative statistics.

1. When the current queue contents or current storage contents at the end of a simulation run are equal to the maximum contents observed during the run, be suspicious of a monotonic build-up caused by some undesired blocking condition. If this condition occurs in a series of runs, with the current contents increasing at a relatively constant rate, then definitely look for a logical error which is causing undesired blocking. Possible errors are:

- a. Transactions are failing to RELEASE or RETURN facilities which they have SEIZED or PREEMPTed.
- b. Transactions are failing to LEAVE storage which they have ENTERED.
- c. Transactions are failing to set or reset logic switches in LOGIC S or LOGIC R blocks.

Each of the logical errors may cause infinite queues to build up at SEIZE, PREEMPT, ENTER, and conditional entry GATE blocks.

2. Another sign of wrong blocking delays are transactions in the current events chain which have very low block departure times in the BDT column in relation to the current value of the absolute clock. These transactions will occur as the first transactions printed out in each of the eight priority classes. The priority level is identified by the PR column of the transaction printout. Why have these transactions been delayed so long at their current block, which is printed out in the

BLOCK column? Why have they not been able to enter their next block, which is printed out in the NBA column? The S column shows the type of next block selection:

- 0 - only the one next block will be tried
- 1 - a BOTH selection mode is being used in a TRANSFER block
- 2 - an ALL selection mode is being used in a TRANSFER block

For the BOTH and ALL modes, the NBA column lists the first next block to be tried, i.e., the field B argument of the TRANSFER block.

3. Further major signals of wrong blocking delays are:

- a. Zero average facility utilization
- b. 1.000 average facility utilization
- c. Zero average storage utilization
- d. 1.000 average storage utilization
- e. Zero average queue contents

The zero values may indicate that transactions are being blocked at earlier points in a model and are failing to reach SEIZE, PREEMPT, ENTER, and QUEUE blocks.

The 1.000 values may indicate that transactions have entered the above block types but are now failing to enter RELEASE, RETURN, and LEAVE blocks.

4. Transactions may be wrongly delayed in ASSEMBLE, GATHER, and MATCH blocks because the required number of mate transactions in their assembly sets are failing to enter the above blocks. These delayed transactions can be identified in the interrupt selection of the transaction printout by:

- a. a 1 in the MC column, indicating that the transaction is in a matching condition
- b. A low block departure time in the BDT field relative to the current absolute clock time

A search should be made of their assembly sets to determine where their mate transactions are, and why they have not succeeded in entering the particular ASSEMBLE, GATHER, and MATCH blocks. The assembly set linkages are defined by the transaction numbers in the TRANS and SET columns. If the interrupted matching condition transaction is the only member of its assembly set, the transaction will never be able to leave the ASSEMBLE, GATHER, or MATCH block.

5. Whenever SPLIT transactions are created in a model it is important to identify the members of each assembly set by the assembly set

linkages and to justify that each member is in the proper block with the proper attributes.

6. One of the obvious consequences of incorrect blocking is that all available transactions may end up being used in the model. A GENERATE or SPLIT block will then attempt to obtain a transaction, and the classic Execution Error #468 (transaction space exceeded) will occur. It is possible, however, that Execution Error #468 does not really represent an error condition. The system being simulated may have more transactions in it as a normal state. The analyst must then revise his model so that the transaction limit is not exceeded or use reallocation.

7. The initial choice of the lower limit (field B) and the interval width (field C) of a TABLE card may not properly bracket the observed values of the table argument. If delay times or transit times ($M1, MP_n$) are being TABULATED, the following rule of thumb should be helpful: Determine the sum of the average service times, i.e., the average ADVANCE block times between the original MARKing and the TABULATE block. Values as high as ten times the average service time may be observed. Also determine the minimum possible sum of service times (which may be zero). This value, rounded to the nearest 100 or 1000, can be the field B upper limit of the lowest interval. Twenty intervals (field D) can be used at the start. The width of the frequency interval (field C) can then be set equal to the following value:

$$\text{Interval width} = \frac{10 \times \text{Average service time} - \text{Minimum time}}{20}$$

The interval width can be rounded to the nearest convenient figure, e.g., 50, 100, 250, 1000, etc.

8. A GENERATE block with a mean time of zero, and no field D creation limit, must be followed by a block which can eventually cause blocking: SEIZE, PREEMPT, ENTER, GATE, or TEST. Otherwise, the GENERATE block will continue creating transactions at clock time 1 until all allocated transactions are used up and an error 468 (transaction space exceeded) occurs.

9. When the analyst has trouble finding the cause of an error, he should always consider the possibility of a subtlety of the overall GPSS/360 scan. Many logical errors are caused by a false assumption of the order of transaction movement. All block sequences which involve SPLIT, MATCH, BUFFER, and ASSEMBLE blocks should be examined for possible errors in timing.

10. One of the major problems in the use of GPSS/360 is optimization of running time. This is

an individual problem, depending upon the system being modeled and the method of attack used by the analyst. However, one point to keep in mind is to have no more transactions "active" in the system than are absolutely necessary. The use of TRANSFER blocks with BOTH and ALL selection modes, and of the TEST blocks with no exit, i. e., in a conditional entry mode, must be restricted as much as possible. Often, when a BOTH or ALL selection mode is necessary in a TRANSFER block, the running time can be decreased by not allowing a transaction to get into the block until it is known that one or more exits are open. This can usually be done by the use of a GATE block or an ENTER block whose store capacity is equal to the number of exits from the TRANSFER block. The use of the TEST block can sometimes be eliminated by the use of a GATE block, but this is more difficult to model.

Another point to be remembered in optimization of running time is the fact that a little loss of reality in the model can sometimes produce a dramatic change in running time without affecting results significantly.

STATISTICAL PROBLEMS IN SIMULATION

The determination of the validity of computer simulation results is a very difficult problem. Two important questions to answer are:

1. How much simulator operation is needed with each system modeled to ensure that the system attains a steady state, assuming one is attainable?
2. Since simulator data is sequentially produced, the results are often sequentially dependent. How may one allow for the effects of this dependence?

Not all systems need to be evaluated in a steady state characterized by an average behavior which remains relatively constant, and which includes the effects of numerous small fluctuations as opposed to long-range trends. In applications which are cyclic or intermittent, the increase and decrease of system loading may be more important. Examples of this type of system are banks and stores, which only operate during part of a day, and may be simulated over a period of several days, and for which the effects of growth and decline of loading are important.

Examples of systems which do need to be evaluated at steady-state conditions are power plants, computer installations, and those whose operations are more or less continuous over the period simulated. Where steady-state evaluation

is important, it is desirable to accumulate simulation statistics which exclude transient effects. This may be effected in one of two basic ways: (1) the simulator may be operated long enough so that the error resulting from including the transient-period data is small, or (2) when the steady state is attained, the statistical data from the transient period may be wiped out with a RESET card, leaving the transactions in the system, after which the simulator may be run further to collect steady-state data only. The difficulty in either method is to know when the transient state has been passed. As yet, this problem has no definite solution.

It is known that different statistical elements within a single simulation problem have varying lengths of transient state and different rates of convergence toward average values in the steady state. Hence, the length of run required to pass through the transient state will be that length due to the maximum of all the transients in a given problem. At the present time, the amount of steady-state data to be collected for convergence is not known. Note that the term "steady-state" does not usually imply constant state, as a certain amount of erratic fluctuation is often seen, especially in queuing data.

Some of the statistical results whose trends should be observed in estimating transiency and convergence are the mean, the standard deviation or variance, the distribution, the cumulative distribution, and the extreme values.

The sequential nature of the Monte Carlo simulation process frequently results in a loss of randomness in the data compiled, especially in queuing data. In a simple queue, where the first unit in line is the next unit served, the amount of time spent in the queue by a transaction is a function of the time spent by preceding transactions. Similarly, if changes in queue lengths are restricted to unit arrival and unit departure, the number in queue after each change of queue length is clearly dependent on the previous queue length, rather than being random. Since nonrandomness invalidates certain standard statistical techniques, special methods must be used to offset the dependence of transactions on each other. One such method is the batching of data; another way is to derive the output statistics from separated values of the quantity to be tabulated, say every third or every sixth item.

APPENDIX A: GPSS/360 PROGRAM ERRORS

ASSEMBLY PROGRAM ERRORS

<u>ERROR NO.</u>	<u>SIGNIFICANCE</u>
1	Illegal selection mode specified in a field of TRANSFER block.
2	Illegal operation field.
3	Entity number to be reserved by EQU card has been reserved by previous EQU card.
4	This 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 EQU or MATRIX card.
8	Illegal entity indicator.
9	Field A 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 or/and columns specified in a MATRIX card is/are not constants.
14	Field E of MSAVEVALUE block is illegal.
15	TRANSFER block with ALL or PICK selection mode contains field C whose value is less than field B.
16	TRANSFER block with ALL selection mode contains field B and field C range which is not evenly divisible by the field D.
17	Card which must have entry in location field has a blank location field.
18	Illegal Halfword Matrix Savevalue.
19	Illegal mnemonic specified in operation field of GATE, LOGIC, TEST, COUNT or SELECT block.
20	Illegal field B in PRIORITY block.
21	A symbol in the above 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.

ERROR NO.	SIGNIFICANCE
22	Illegal Boolean Variable number.
23	Illegal report type specified in REPORT card.
24	Above card type is not permitted within the report type specified.
25	Storage defined with capacity greater than the maximum permissible value.
26	The Table must be specified numerically.
27	Illegal symbol.
28	Illegal FUNCTION type.
29	Modifier of GENERATE or ADVANCE block exceeds mean.
30	Field A omitted where it must be specified.
31	Field B 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.
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 above card.
45	Illegal SNA.
46	Field C omitted where it must be specified.
47	Illegal Matrix Savevalue number.

ERROR NO.	SIGNIFICANCE
48	Field D omitted where it must be specified.
49	Maximum number of MACROS already defined.
50	Undefined MACRO.
51	Illegal MACRO argument - argument must be alphabetic A-J.
52	MACRO card expanded past column 72.
53	More than 2 MACRO's nested within a MACRO.
54	More than 10 arguments specified in above MACRO card.
55	Field C of SAVEVALUE block is illegal.
56	Illegal Halfword Savevalue.
57	There is no legal entity number left to be assigned to the entity symbol.
58	Operand field extends into column 72.
59	There are more right parenthesis than left parenthesis in a VARIABLE card.
60	There are more left parenthesis than right parenthesis in a VARIABLE card.
61	An impossible module division has been 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 field A of Statement Card
66	Illegal row request in Statement Card
67	Illegal field B of 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

ERROR NO.	SIGNIFICANCE
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

INPUT ERRORS

ERROR NO.	SIGNIFICANCE
201	Number of transactions exceeded.
202	Referenced transaction not inactive .
203	Priority exceeds 127.
204	Limit count must be a constant.
205	Number of parameters exceeds 100.
206	GENERATE Block: Field F must be F, H, or blank
207	Preempt Block: Field B must be 'PR'
208	Field E must be 'RE'
209	Field C not specified with D and/or C Fields
210	Illegal mnemonic in operation field.
211	Illegal storage number.
212	Field D not necessary if MAX or MIN mode specified.
213	Illegal mnemonic in Field C of PRINT block.
214	Illegal format in Logic Switch Initial Card.
215	Amount of available GPSS/360 COMMON core exceeded.
216	Modifier cannot exceed mean.
217	Action time not ≥ 0 .
218	Illegal Fullword Matrix number.

ERROR NO.	SIGNIFICANCE
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: Field E 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 XH used in Savevalue Initial Card.
229	First index higher than second index in multi-initialization.
230	Illegal Logic Switch number.
231	VARIABLE DEFINITION CARD: Column 18 not blank.
232	Illegal variable number.
233	Number stated incorrectly.
234	Improper number of parentheses.
235	Too many sets of parentheses.
236	Impossible Modulo Division.
237	Illegal Boolean Variable number.
238	Modulo division in Floating-Point Variable.
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.

ERROR NO.

SIGNIFICANCE

245	Illegal Matrix Savevalue mnemonic.
246	Illegal format in MATRIX Initial Card.
247	Illegal mnemonic in SAVEVALUE Initial Card.
248	Illegal Halfword Savevalue.
249	Too many numeric digits in constant.
250	Illegal SNA mnemonic.
251	Missing operator in Variable.
252	Field E not blank when BV specified in UNLINK Block.
253	Illegal MNEMONIC in TRANSFER Block field A.
254	Fraction in TRANSFER Block field A not 3 digits.
255	Matrix Initial Card: Illegal index for rows.
256	Illegal index for columns.
257	Illegal Queue number.
258	Illegal JOBTAPE number.
259	Cyclic Definition of Variable.
260	Variable not defined.
261	Illegal Variable number.
262	Cyclic definition of Function.
263	Illegal Function number.
264	Undefined Function.
265	Illegal Function type.
266	Function must have more than one point.
270	No field C in EXAMINE Block.
271	Illegal Entity Number on RESET Card.
272	Illegal Entity Type requested on RESET Card.
273	Sequence Error on RESET Card.
274	Illegal request on Selective CLEAR Card.

ERROR NO.	SIGNIFICANCE
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 CLEAR Card.
279	READ/SAVE Identifier not found on specified READ Device.
280	Illegal Allocation of Entities on READ Device.
282	Error in block redefinition.
283	Illegal block number.
284	Illegal transaction referenced in chaining routine (not a user's error).
290	Illegal reference to GPSS/360 COMMON (not a user's error).
291, 293	Illegal SNA referenced (not a user's error).

EXECUTION ERRORS

ERROR NO.	SIGNIFICANCE
401	No new event in the system
402, 3, 4	Illegal Transaction in Future Events Chain (normally not a user's error)
405	Number of parameters exceeded
413	Illegal entry to GENERATE block
415, 16	Facility released by Transaction not SEIZing it
417	Interrupt count is minus (not a user's error)
421	Facility returned by a Transaction not PREEMPTing it
425	Transaction leaving by more than storage contents
428	Transaction which is leaving Queue by more than Queue contents
429	LOOP Block field A parameter zero before entering block
431	Value being stored in Halfword Savevalue is too large
432	Illegal Halfword Savevalue number

ERROR NO.	SIGNIFICANCE
433	Illegal Fullword Savevalue number
435	Illegal Table number
436	Table not defined by TABLE Card
437	Illegal transaction number referred to under blocked condition (not a user's error)
438	Attempting to place a Transaction on a delay chain when the Transaction is on the delay chain (not a user's error)
442	No preempt count in Transaction returned from preempt condition (normally not a user's error)
443	Attempting to create an active Transaction (not a user's error)
453, 63, 66, 67	Attempting to remove transactions from illegal chains (not a user's error)
468, 69	Number of Transactions exceeded
470, 71, 72, 75	Illegal Transaction number being acted on (not a user's error)
474	Preempt interrupt count exceeds 127
476, 77	Attempting to remove an interrupt on a Transaction which has not been interrupted (not a user's error)
492	Illegal Transaction Parameter number
497	Illegal Link Chain referenced
498	Illegal Facility number
499	Illegal Storage number
500	Illegal Queue number
501	Illegal Logic Switch number
505	Minus time-delay computation (ADVANCE or GENERATE)
506	Cyclic Function definition
507	Illegal Function number
508	Function not defined by FUNCTION Card
509	Illegal index evaluated for list-type function
512	Entering undefined block
514	Illegal Variable number

ERROR NO.	SIGNIFICANCE
515	Arithmetic Variable not defined by VARIABLE Card
516	Cyclic Definition 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 Card
562	Illegal Column in MSAVEVALUE Card
599	Limits of GPSS/360 COMMON core exceeded
601	Next sequential block number is illegal
602	Illegal Facility sequence in output
603	Illegal Block number
604	Illegal Table argument
607	Field A assembly or gather count is zero at ASSEMBLE or GATHER Blocks
609	Transaction of a one-member set is at MATCH, GATHER, GATE-M- or GATE-NM- Blocks
610	Upper limit less than lower limit in COUNT or SELECT Block
611	Illegal Transaction Parameter number
612	Illegal Block number referenced in TRANSFER BOTH or ALL
613	Illegal User's Chain
614	Priority exceeds maximum allowed (127)
615	Error in core assignment (not a user's error)
616	Transaction parameter zero referenced
617	Illegal MATRIX number
618	Cyclic definition of MATRIX
619	Matrix not defined by MATRIX Card
620	Illegal Matrix column number

ERROR NO.	SIGNIFICANCE
621	Illegal Matrix row number
622	Illegal Boolean Variable number
623	Boolean Variable not defined by BVARIABLE Card
624	Cyclic definition of Boolean Variable
626	Illegal Group number
627	Field C less than field B, TRANSFER PICK
669, 670	Improper Queue assignment (not user's error)
698	Illegal change in CHANGE Block
699	Illegal argument in EXECUTE Block
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 (other than END, SNAP, PRINT, TRACE, or ERROR IN EXECUTION)

Execution Warning Messages

A warning message will be written following the listing of the simulator input deck if a possible user error is detected during the execution phase of the simulation run. The format of the warning message will be as follows:

WARNING - EXECUTION ERROR NUMBER xxx, BLOCK NUMBER yyyy, CLOCK zzzzzzzz, SIMULATION CONTINUES.

where:

xxx = Number indicating source of error according to following list:

- xxx = 850: Attempt to store integer of magnitude greater than $2^{15}-1$ (32, 767) in a halfword parameter. Content of halfword parameter is maintained modulo $2^{15}-1$.
- xxx = 851: Attempt to store integer of magnitude greater than $2^{15}-1$ (32, 767) in a halfword Savevalue. Content of halfword Savevalue is maintained modulo $2^{15}-1$.
- xxx = 852: Attempt to store integer of magnitude greater than $2^{15}-1$ (32, 767) in a halfword Matrix Savevalue. Content of halfword Matrix Savevalue is maintained modulo $2^{15}-1$.
- xxx = 853: Transaction attempting to enter a QUEUE block is already a member of 5 Queues. The contents of the Queue specified by field A will be incremented by the amount specified in field B. Testing for maximum contents will be made and updating will be done accordingly. Cumulative time integral information will not be calculated. Therefore, statistics on average time per transaction in Queue, number of zero entries, percent zeros, and average time per transaction excluding zero entries, will be in error.
- xxx = 854: Transaction at a DEPART block is not a member of the Queue specified by field A of the DEPART block. The Queue contents are decremented by the amount specified by the B argument of the DEPART block and the transaction proceeds to the next sequential block.
- xxx = 861: This indicates that an End-of-File has been reached on JOBTAPE 1. Transactions on the current file have entered the simulation model.
- xxx = 862: Same as above for JOBTAPE 2.
- xxx = 863: Same as above for JOBTAPE 3.
- yyyy: Number of block at which the possible error is detected.
- zzzzzzzz: Absolute clock time when possible error first occurs at the indicated block.

Each possible error message is written only the first time that the error is detected at a given block. Because the condition may not actually be a serious error or significantly affect the simulation statistics, the simulation run is allowed to proceed.

GPSS/360 coding is done in a symbolic, free-form language. This is converted by the assembly phase into a numeric, fixed-field format acceptable to the actual simulator program. The reason for having a symbolic, free-form language is to reduce modeling effort and error and to provide more meaningful output.

All GPSS entities as well as block locations may be symbolically named. There are three fields in the GPSS/360 input card:

1. Location field (columns 2-6)
2. Operation field (columns 8-18)
3. Operand field (columns 19-72)

The GPSS program executes blocks in a sequential manner and therefore the assembly phase assigns sequential numbers to each block beginning with 1. There is no need to specify anything in the location field of a block unless this block will be directly referred to at another point in the model.

An entry in the location field should begin in column 2 and must end by column 6. The operation field entry must begin in column 8. Arguments which make up the operand field must begin in column 19 and be separated by commas. If a preceding field in the operand field is to be left blank this is indicated by showing only the separating comma. For example:

PRINT , , MOV has only a field C entry. In this case the first comma would appear in column 19.

The operand field is terminated by the first blank encountered. The assembly will assign the next block location to the field B of a TRANSFER block if the field B is left blank except for FN or P selection mode.

TRANSFER BOTH, , EXIT has no field B entry and therefore the next block number is automatically assigned as the field B value.

BLOCK AND ENTITY SYMBOLS

Each symbol must consist of three to five alphabetic characters, the first three of which must be letters from A-Z and the last two, if used, must be letters from A-Z or numbers from 0-9. No special characters are permitted.

The restriction on the first three characters being alphabetic is necessary to avoid confusing a standard numerical attribute with a block symbol.

LOCATION FIELD ARGUMENTS

Whenever the analyst wishes to define a block for reference elsewhere in a model he may do so by placing a symbol in the location field. Since most blocks are not referred to directly in a GPSS model, symbols are not required in the location field of each block and may be left blank.

As mentioned previously block numbers are sequentially assigned and whenever a block symbol is encountered the numeric block value that has been assigned by the assembly phase and the symbol are placed in a symbol table.

If the same symbol occurs in the location field of more than one block it is not assigned the next sequential block number but the originally assigned block number is assigned to the location field. A message indicating this has occurred appears below the card in the assembly listing printout: "MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD". Frequently, however, this is the analyst's intention and this condition will not cause termination of a run.

Entity definition cards such as STORAGE, FUNCTION, VARIABLE, and TABLE require entries in the location field. The user has the option of specifying numeric or symbolic values. If a symbol is placed in the location field the assembly phase will assign a free entity number to that symbol.

OPERAND FIELD ARGUMENTS

In a GPSS model symbols may also be encountered in the operand fields. A symbol which is encountered in field A of entity blocks such as SEIZE, QUEUE, LOGIC, etc. will be assigned a free number associated with the entity implied by the block type (symbolic entity reference will be discussed in more detail later). When a symbol is encountered in a field which does not imply an entity relationship such as does the A field of a SEIZE block, etc. it is assumed to be a block symbol and will be replaced by the corresponding block number. If this symbol has not been defined then an undefined block symbol error will be noted in the assembly listing printout.

When a symbolic standard numerical attribute is used in an operand field a dollar sign (\$) must be used to prefix the symbol. For example:

W\$ARM - the current block count at block ARM, the \$ is necessary to distinguish this from the

block symbol WARM. The analyst also has the option of placing a \$ before a block symbol in the operand field. If a dollar sign is used in the operand field a symbol must follow it. Dollar signs may never appear in the location field.

RELATIVE ADDRESSING OF BLOCK LOCATIONS

If an analyst wishes to refer to a specific block card without assigning a symbol to it he may do so if there is another block in the vicinity which contains a symbol in its location field. He could refer to the specific card using the following type of expression: ALPHA ± n, where ALPHA refers to a reference block and n refers to the number of blocks the desired block is located from the reference block. For example:

```
TRANSFER BOTH, ARM + 1, ARM + 6
```

The use of this technique will possibly lead to errors since the analyst may for instance add a block between ARM + 1, ARM + 6 and forget that he should also change the reference in the preceding TRANSFER block to ARM + 7.

FUNCTION FOLLOWER CARDS

There are two formats that may be used for FUNCTION follower cards:

1. a fixed format which assumes a maximum of six points per card
2. a free format which allows n points per card.

These formats may not be intermixed in any one FUNCTION.

If the fixed format is used, the X_i and Y_i points must appear in successive six-column fields: 1-6, 7-12, 13-18, . . . 67-72 and not separated by commas.

If the free format is used the points must appear in the following format beginning in column 1 and ending before column 72:

$X_1, Y_1/X_2, Y_2/X_3, Y_3/ . . . X_n, Y_n$. In this format constants may occupy up to 10 columns including the decimal point for fractional constants.

With both formats the Y_i values may be coded symbolically or numerically depending on the analysts needs and the X values must, of course, be numeric.

SYMBOLIC ENTITY REFERENCE

GPSS/360 entities may be referred to either numerically or symbolically. It will be possible to refer, for instance, to a communication line by the symbol LINE.

```
SEIZE          LINE
ADVANCE        FN2
RELEASE        LINE
```

Notice in the above example that just the symbol LINE was placed in field A of the SEIZE block since field A of this block always contains a facility name or number. The assembly phase will assign a facility number to the symbol LINE in this case.

In some instances where an analyst would want to use a symbol, the field does not imply what entity this symbol should represent. For example:

```
ASSIGN          3, CPU
.
.
SEIZE          *3
```

The user wishes the symbol CPU to represent a facility but this is not apparent from its use in the ASSIGN block. This necessitates the introduction of a new control card.

```
2      8      19
NAME  EQU      k,I,J, . . .
```

The EQU card will assign the number k to the symbol (3 to 5 characters, first 3 alphabetic) which begins in column 2 and it will associate this symbol with the entities numbered k, given by I, J, etc.

```
CPU  EQU      5, F
```

The above card associates the symbol CPU with facility 5 so that

```
ASSIGN          3, CPU
```

would be assembled as

```
ASSIGN          3  5
```

Another use of the EQU card is to assure that a symbol representing more than one entity has the same number associated with it for each entity so that block sequences such as the following may be conveniently represented.

```

ASSIGN          6, LINE 1
.
.
.
QUEUE          *6
SEIZE          *6
.
.

```

The card

```

LINE 1      EQU      2, F, Q

```

would handle the above situation.

The following is a list of the mnemonics which may be used in the operand field of the EQU control card. The user may specify as many as he wishes as long as each entry is separated by a comma, and the entries end by column 71.

F - Facilities	T - Tables
S - Storages	V - Variables
Q - Queues	L - Logic Switches
X - Savevalues	C - User Chains
H - Halfword Savevalues	Z - Functions
M - Matrix Savevalues	B - Boolean Variable
Y - Halfword Matrix	G - Groups

Whenever a symbol is used in an EQU card it should not be used as a symbol to represent a block because when the assembly program encounters a symbol positioned as in the above ASSIGN block, it has no way of knowing that it isn't a block symbol. Therefore, whenever a symbol which has been used in an EQU card is used as a block symbol, an error will result.

The numbers the assembly program assigns various entity symbols will be sequential beginning with 1 unless directed otherwise by an EQU card or an entity function (to be discussed later).

```

LINE      EQU      5, F, Q
          SEIZE    1
          ADVANCE  10
          QUEUE   LINE
          SEIZE   LINE
          SEIZE   TERM

```

The above blocks would be assembled as:

```

          SEIZE    1
          ADVANCE  10
          QUEUE   5
          SEIZE   5
          SEIZE   2

```

Since Facility 1 is used directly it is not assigned to the symbol TERM when a facility number must be obtained for this symbol.

Symbolics may also be used with SNAs merely by separating the SNA mnemonic and the symbol with a \$.

```

TEST      VARIABLE  10*P5
          ASSIGN    3, V$TEST
          SEIZE     TRAY
          ADVANCE   FN$ONE

```

The cards above would be assembled as:

```

1      VARIABLE  10*P5
          ASSIGN    3          V1
          SEIZE     1
          ADVANCE   FN1

```

It is also possible to symbolically identify and reserve a sequential set of GPSS/360 entities. This is done by use of the EQU card.

```

TAPE     EQU      10(5), F

```

The above card will reserve Facility numbers 10-14 with the symbol TAPE associated with Facility number 10.

The card SEIZE TAPE + 2 would then be assembled as

```

SEIZE    12

```

The entity function will assign entity numbers to the symbolic points within the function. The format for the function follower cards is the same as it is for ordinary functions. The format for the entity function definition card is:

```

2          8          19
NAME or    FUNCTION  XXX, Sn, I, J, . . . .
NUMBER

```

The first operand field beginning in column 19 is the independent argument. The second argument specifies an entity function with n points and I, J, . . . specify the entity type(s) represented by the function points.

See the description of the EQU card for characters which represent the various entities.

```

FUNCTION  P5, S4, F, Q
LINE 1 3   LINE 2 8   LINE 3 9   LINE 4 10

```

The above function would cause the following facility numbers and queue numbers to be associated with the following symbols assuming Facility number 2 has already been associated with some other symbol by an EQU card.

```

LINE1          1
LINE2          3
LINE3          4
LINE4          5

```

The function would be assembled as;

```

FUNCTION  P5  D4
1      3  3  8  4  9  5  10

```

If a symbol has been used in an EQU card or as a block symbol or in a previous entity function it may not be used in an entity function.

ASSEMBLY PHASE OUTPUT

Output from the assembly phase will include an alphabetic list of block symbols which were defined within the model, the block number assigned to each symbol and a cross-reference of the card numbers in which the symbol was referenced.

The output will also include a table of entities which were defined symbolically within the model. The symbols will be listed alphabetically within entity type, along with the numerical value which was assigned to the symbol by the assembly phase.

The following listing illustrates this expanded output printout as well as the symbolic naming of entities.

BLOCK NUMBER	*LOG	OPERATION	A,B,C,D,E,F,G	COMMENTS	CARD NUMBER			
		SIMULATE			1			
	ILAN1	EQU	1,F		2			
	ILAN2	EQU	2,F		3			
	MAN1	EQU	3,S,L		4			
	MAN2	EQU	4,S,L		5			
	REG01	EQU	10,F		6			
	REG02	EQU	11,F		7			
	PREMI	EQU	12,F		8			
	GASTA	STORAGE	7		9			
	MAN1	STORAGE	1		10			
	MAN2	STORAGE	1		11			
	EXPON	FUNCTION	RN1,C24		12			
0	0	.1	.104 .2	.222 .3	.355 .4	.509 .5	.69	13
.6	.915	.7	1.2 .75	1.38 .8	1.6 .84	1.83 .88	2.12	14
.9	2.3	.92	2.52 .94	2.81 .95	2.99 .96	3.2 .97	3.5	15
.98	3.9	.99	4.6 .995	5.3 .998	6.2 .999	7.0 .9997	8.0	16
*	FUNCTION TO	DETERMINE GAS TYPE						17
	TYPE	FUNCTION	RN1,D2					18
0.60	REG	1.0	PREM					19
	RATE	VARIABLE	N\$DOG/8					20
	*							21
1		GENERATE	3,FN\$EXPON	GENERATE CARS PASSING - EXP DIST	22			
2		TRANSFER	.980,INPUT,PASS		23			
3	PASS	TERMINATE		DOES NOT NEED GAS	24			
4	INPUT	TRANSFER	BOTH,INPU1,PASS1	TEST IF CAR CAN ENTER STATION	25			
5		ADVANCE	3	TIME TO ENTER STATION	26			
6	INPU1	ENTER	GASTA,1	CAR ENTERS STATION	27			
7		QUEUE	ISLAN,1	ENTER QUEUE FOR ISLAND AREA	28			
8		ASSIGN	5,FN\$TYPE	ASSIGN GAS TYPE TO P5	29			
9		TRANSFER	BOTH,ISAL1,ISAL2	TRY TO ENTER ISLAND SPACE	30			
10	ISAL1	SEIZE	ILAN1	SEIZE ISLAND 1	31			
11		ASSIGN	12,ILAN1	ASSIGN ISLAND 1 TO P12	32			
12	POSIT	DEPART	ISLAN,1	LEAVE QUEUE FOR ISLAND AREA	33			
13		ADVANCE	4	POSITION CAR TO ISLAND	34			
	*				35			
	*	ATTENDANT ASSIGNMENT			36			
	*				37			
14		GATE LR	MAN1,ATTD2	TEST FOR ATTENDANT CHOICE	38			
15		ASSIGN	10,MAN1	ASSIGN ATTENDENT 1	39			
16		LOGIC S	*10	SET LOGIC SWITCH 3	40			
17	ATTEN	ENTER	*10,1	OBTAIN ATTENDANT	41			
18		ADVANCE	5	INSTRUCTIONS TO ATTENDANT	42			
19		TRANSFER	,*5	SELECT REGULAR OR PREMIUM GAS	43			
	*				44			
	*				45			
	*				46			

BLOCK NUMBER	*LOC	OPERATION SIMULATE	A,B,C,D,E,F,G	COMMENTS	CARD NUMBER
20	ISAL2	SEIZE	ILAN2	SEIZE ISLAND 2	47
21		ASSIGN	12,ILAN2	ASSIGN ISLAND 2 TO P12	48
22		TRANSFER	,POSIT		49
	*				50
	*				51
	*				52
23	ATT2	ASSIGN	10,MAN2	ASSIGN ATTENDENT TWO TO P10	53
24		TRANSFER	,ATTEN		54
	*				55
	*	CAR CANNOT ENTER STATION ADD TO TOTAL			56
	*				57
25	PASS1	SAVEVALUE	COUNT&,1	ADD TO NOT ABLE TO ENTER COUNTER	58
26		TERMINATE			59
	*				60
	*	REGULAR GAS PUMPS			61
	*				62
27	REG	TRANSFER	BOTH,REG1,REG2	SELECT REGULAR PUMP	63
28	REG1	SEIZE	REG01		64
29		ASSIGN	6,REG01	ASSIGN REGULAR PUMP TO P6	65
30	CONT	ADVANCE	10	ATTENDANT WITH NOZZLE	66
31		SPLIT	1,ATT1		67
32	CONT1	ADVANCE	180,60	PUMP GAS	68
33		RELEASE	*6	RELEASE PUMP	69
34		ENTER	*10,1	RECALL ATTENDANT	70
35		ADVANCE	1	COLLECT CASH OR CARDS	71
36		TRANSFER	.750,TWEN,SEVEN		72
37	TWEN	ADVANCE	15,5	CASH CUSTOMERS	73
38	CONT2	LEAVE	*10,1	RELEASE ATTENDANT	74
39		LOGIC R	*10	ALLOW FOR ATTENDANT RELEASE	75
40		ADVANCE	7	PUT CHANGE AWAY - START CAR	76
41		RELEASE	*12	RELEASE ISLAND	77
42		ADVANCE	10	LEAVE GAS STATION	78
43		LEAVE	GASTA,1	LEAVE GAS STATION	79
44	DOG	TERMINATE			80
	*				81
	*				82
	*				83
45	REG2	SEIZE	REG02		84
46		ASSIGN	6,REG02	ASSIGN REGULAR PUMP 2 TO P6	85
47		TRANSFER	,CONT		86
	*				87
	*	REGULAR PATH TO ACCOUNT FOR ATTENDANT UTILIZATION			88
	*				89
48	ATT1	ADVANCE	1	ATTENDANT DECIDES WHAT TO DO	90
49	ATT2	ADVANCE	60,30	WASH WINDOWS AND CHECK OIL	91
50		LEAVE	*10,1	RELEASE ATTENDANT	92

BLOCK NUMBER	*LOC	OPERATION	A,B,C,D,E,F,G	COMMENTS	CARD NUMBER
51		TERMINATE			93
	*				94
	*				95
52	SEVEN	ADVANCE	25,5	CREDIT CUSTOMERS	96
53		TRANSFER	,CONT2	TRANSFER TO MAIN PATH	97
	*				98
	*	PREMIUM GAS PUMP			99
	*				100
54	PREM	ASSIGN	6,PREM1	ASSIGN PREMIUM PUMP TO P6	101
55		GATE NU	*6,ALTER	TEST IF PREMIUM PUMP AVAILABLE	102
56		SEIZE	*6	SEIZE PREMIUM PUMP	103
57		TRANSFER	,CONT	TRANSFER TO MAIN PATH	104
	*				105
	*				106
	*				107
58	ALTER	SPLIT	1,PREM1	SPLIT TO ACCOUNT FOR ATTENDANT	108
59		QUEUE	PREM1,1	QUEUE FOR PREMIUM PUMP	109
60		SEIZE	*6	SEIZE PREMIUM PUMP	110
61		DEPART	PREM1,1	LEAVE QUEUE FOR PREMIUM PUMP	111
					112
62		TRANSFER	,CONT1		113
	*				114
	*				115
63	PREM1	ADVANCE	11	ACCOUNT FOR DECISION AND LOADING	116
64		TRANSFER	,ATT2		117
	*				118
	*	TIMING GENERATOR FOR AN 8 HOUR DAY			119
	*				120
65		GENERATE	28800		121
66		SAVEVALUE	RATE,V\$RATE		122
67		TERMINATE	1		123
		START	1		124
		END			125
					126

BLOCK NUMBER SYMBOL REFERENCES BY CARD NUMBER

58	ALTER	103
23	ATTO2	38
17	ATTEN	54
48	ATT1	67
49	ATT2	118
30	CONT	86 105
32	CONT1	113
38	CONT2	98
44	DOG	21
4	INPUT	23
6	INPU1	25
10	ISAL1	30
20	ISAL2	30
3	PASS	23
25	PASS1	25
12	POSIT	49
54	PREM	20
63	PREM1	109
27	REG	20
28	REG1	63
45	REG2	63
52	SEVEN	72
37	TWEN	72

FACILITY SYMBOLS AND CORRESPONDING NUMBERS

1	ILAN1
2	ILAN2
12	PREMI
10	REG01
11	REG02

STORAGE SYMBOLS AND CORRESPONDING NUMBERS

1	GASTA
3	MAN1
4	MAN2

QUEUE SYMBOLS AND CORRESPONDING NUMBERS

1	ISLAN
2	PREMI

SAVEVALUE SYMBOLS AND CORRESPONDING NUMBERS

1	COUNT
2	RATE

VARIABLE SYMBOLS AND CORRESPONDING NUMBERS

1	RATE
---	------

LOGIC SWITCH SYMBOLS AND CORRESPONDING NUMBERS

3	MAN1
4	MAN2

FUNCTION SYMBOLS AND CORRESPONDING NUMBERS

1	EXPUN
2	TYPE

MACROS

Macros are strings of frequently used blocks defined by the user, which he may later call with only one card. The only advantage obtained by using macros is the elimination of the need to code and keypunch repetitive strings of blocks.

A GPSS/360 user will be able to incorporate macros into his model deck. The number of macros in the standard GPSS/360 program will be 50 but may be reallocated by the user, as any GPSS entity may. A maximum of ten arguments per macro is allowed.

The definition of macros requires two new control cards STARTMACRO and ENDMACRO, and the calling of a macro requires one new control card, MACRO.

All macros to be used in a model must be defined at the beginning of the GPSS/360 input deck. The user informs GPSS that a macro definition is to begin by means of the STARTMACRO card.

```

2           8           19
NAME       STARTMACRO
  
```

The name of the macro to be defined starts in column 2 and must be 3 to 5 characters with the first 3 characters alphabetic. The macro definition is terminated by the ENDMACRO card.

```

2           8           19
NAME       ENDMACRO
  
```

The actual macro definition cards follow the normal GPSS format except that some fields may be replaced by macro arguments. Macro arguments are represented by following a special character # with a letter (A-J) which represent arguments 1-10 respectively.

Macros are called within a GPSS/360 program by means of the MACRO card.

```

2           8           19
NAME       MACRO           A, B, C, D, E, . . . J
  
```

The name of the macro being called starts in column 2 and the arguments to be substituted in the macro definition cards start in column 19.

As a simple example consider the following block sequence:

```

SEIZE      1
ADVANCE    14
RELEASE    1
  
```

The same sequence of SEIZE, ADVANCE, RELEASE will be used many times in a program but with different facilities and a different ADVANCE time. The block sequence could be defined as a macro as shown.

```

EASY       STARTMACRO
           SEIZE      #A
           ADVANCE    #B
           RELEASE    #A
           ENDMACRO
and then called later in the program.
           GENERATE   40,10
           .
           .
EASY       MACRO      4, 20
           .
           .
EASY       MACRO      2, FN10
           The above card sequence would produce the
           following block sequence.
           GENERATE   40,10
           .
           .
           .
           SEIZE      4
           ADVANCE    20
           RELEASE    4
           .
           .
           .
           SEIZE      2
           ADVANCE    FN10
           RELEASE    2
  
```

Macros may also be called within other macros.

```

EXAM       STARTMACRO
           ADVANCE    #A, #B
           TRANSFER   , #C
           ENDMACRO
TWO        STARTMACRO
           #A         #B, #C
           ADVANCE    #D
           TRANSFER   #E, #C
EXAM       MACRO      #D, #F, #C
           ENDMACRO
  
```

The above card sequence defines the macros EXAM and TWO. Whenever TWO is called within the model the macro EXAM will also be expanded.

The card:

```

2           8           19
TWO        MACRO      TRANSFER, . 704,
                       AGAIN, 45, PICK, FN5
  
```

will be expanded to

```

TRANSFER   . 704, , AGAIN
ADVANCE    45
TRANSFER   PICK, , AGAIN
ADVANCE    45, FN5
TRANSFER   , AGAIN
  
```


Notice that before the macro EXAM was expanded in the above example, the arguments from the macro TWO were substituted for the D, F, C given in the card which calls the macro EXAM so that when EXAM was expanded, the A argument was 45 which was the D argument of the macro TWO, the B argument was FN5 which was the F argument of the macro TWO, and the C argument was AGAIN, which was the C argument of the macro TWO.

Macros may be nested up to two levels, i. e., a macro which is called within a macro may also call still another macro so long as this third macro does not call another macro. Consider the following sequence.

```
TWO      STARTMACRO
          ADVANCE      #A,#B
          ENDMACRO

THREE   STARTMACRO
TWO     MACRO          #B,#C
          ADVANCE      #A
          ENDMACRO

FOUR    STARTMACRO
THREE   MACRO          #A,#B,#C
          ADVANCE      #C
          ENDMACRO
```

If the following MACRO calling card were encountered

```
FOUR    MACRO          20,30,40
the above MACRO definition would cause the following blocks to be assembled.
```

```
ADVANCE      30,40
ADVANCE      20
ADVANCE      40
```

BLOCK NUMBER	*LOC	OPERATION	A, B, C, D, E, F, G	COMMENTS
		SIMULATE		
	TER	STARTMACRO		
	#D	#A	XI, #B, #C	
		ADVANCE	2	
		#E	1	
		SAVEVALUE	#F, #G	
		TERMINATE	#H	
		ENDMACRO		
1		GENERATE	1	
2		ADVANCE	1	
3		SAVEVALUE	1+, 1	
	TER	MACRO	TEST LE, 10, BIG, DMJ, GATE LR, 2+, 5, 0	
4	DMJ	TEST LE	XI, 10, BIG	
5		ADVANCE	2	
6		GATE LR	1	
7		SAVEVALUE	2+, 5	

The user should not confuse calling many macros within a macro, on which there is no limitation if these called macros do not refer to other macros, with nesting of macros. The following trivial example should demonstrate this.

```
FIVE     STARTMACRO
          ADVANCE      #A
          ENDMACRO

SIX      STARTMACRO
          ADVANCE      #A
          ENDMACRO

SEVEN    STARTMACRO
          ADVANCE      #A
          ENDMACRO

EIGHT    STARTMACRO
          ADVANCE      #A
          ENDMACRO

NINE     STARTMACRO
          FIVE         MACRO          #A
          SIX          MACRO          #B
          SEVEN        MACRO          #C
          EIGHT        MACRO          #D
          ENDMACRO
```

The card

```
NINE     MACRO          1,2,3,4
would cause the following sequence to be assembled
```

```
ADVANCE      1
ADVANCE      2
ADVANCE      3
ADVANCE      4
```

The card sequence below and on the next page illustrates definition and calling of macros and their expansion as it will be produced in GPSS/360.

BLOCK NUMBER	*LOC	OPERATION	A, B, C, D, E, F, G	COMMENTS
8		TERMINATE	0	
	TER	MACRO		TEST GE, 5, TERM, BIG, LOGICS, 3+, 4, 1
9	BIG	TEST GE	XI, 5, TERM	
10		ADVANCE	2	
11		LOGICS	1	
12		SAVEVALUE	3+, 4	
13		TERMINATE	1	
14	TERM	TERMINATE	1	
		START	5	
		END		

UPDATE FEATURE

This eliminates the need to transport, store, maintain, and submit large symbolic decks. The update feature will provide the user with the following services:

1. the ability to create a master tape.
2. the ability to update the master tape (delete or insert cards) to create a new master and then assemble and execute the model. This would not modify the old master.
3. the ability to punch the master tape.

A master file is originally created by means of the CREATE control card:

```
2      8      19
      CREATE
```

This card is placed at the beginning of the symbolic deck and will cause the deck to be written on a master tape.

In order to perform an update, three or four card types are required.

```
2      8      19
      UPDATE PUNCH, NOASSEMBLE
      DELETE A-B
      ADD     A
      REPLACE A
      ENDUPDATE
```

The UPDATE card informs the program an update is to take place. This card has two options: PUNCH will cause the new master tape to be punched, NOASSEMBLE will delete the normal transfer of control to the assembly program and will cause the new master tape to be printed. If PUNCH is omitted, no deck will be punched from the new master tape. If NOASSEMBLE is omitted, the update program will automatically transfer control to the GPSS/360 assembly program.

The DELETE card will cause the cards given by A to B inclusive to be eliminated from the new master tape.

```
DELETE 10-25
```

The above card will cause the tenth to twenty-fifth card inclusive on the old master not be placed on the new master file.

```
DELETE 15
```

The above card will cause the fifteenth card on the old master not to be placed on the new master file.

The ADD card will cause the cards following to be placed on the new master file after the card specified in the operand field.

```
ADD 10
```

The above card will cause the cards following the ADD card to be placed on the new master after the tenth card on the old master. This operation will continue until a DELETE, ADD, REPLACE or ENDUPDATE card is encountered.

The REPLACE card will cause the cards following to replace the card specified in the operand field on the new master file.

```
REPLACE 20
```

The above card will cause the cards following it to replace the twentieth card on the new master file. This operation will continue until a DELETE, ADD, REPLACE, or ENDUPDATE card is encountered.

The ENDUPDATE card signals the end of the update. The new master file will be punched if the PUNCH option has been specified and then either control will be transferred to the assembly program or the new master file will be printed depending on whether or not the NOASSEMBLE option was included.

If a master file contained the following records

```
*      SIMULATE
      GENERATE 10, 5
      SPLIT   1, AAA
      SEIZE   1
      ADVANCE 5
```

	RELEASE	1
	TERMINATE	
AAA	SEIZE	2
	ADVANCE	5
	RELEASE	2
	SEIZE	3
	ADVANCE	1
	RELEASE	3
	TERMINATE	1
	START	20
	END	

and an update was performed with the following cards

	UPDATE	NOASSEMBLE
	REPLACE	2
	GENERATE	15,5
	ADD	6
	SEIZE	3
	ADVANCE	1
	RELEASE	3
	DELETE	11-13
	ENDUPDATE	

the new master file would contain the following records

*	SIMULATE	
	GENERATE	15,5
	SPLIT	1,AAA
	SEIZE	1
	ADVANCE	5
	RELEASE	1
	SEIZE	3
	ADVANCE	1
	RELEASE	3
	TERMINATE	
AAA	SEIZE	2
	ADVANCE	5
	RELEASE	2
	TERMINATE	1
	START	20
	END	

More than one operation cannot be performed on any card. For example, consider the following master file:

	ADVANCE	5	35
	SEIZE	1	36
	ADVANCE	10	37
	RELEASE	1	38
	.		
	.		
	.		

cards 36-38 are to be replaced by:

SEIZE	5
ADVANCE	2
RELEASE	5
DELETE	36-38
ADD	38
SEIZE	5
ADVANCE	2
RELEASE	5

The operation could not be performed by

because card 38 is being referenced in both the DELETE and ADD card. The following procedure should be used to perform such an operation:

.	
.	
.	
.	
DELETE	36-37
REPLACE	38
SEIZE	5
ADVANCE	2
RELEASE	5
.	
.	
.	

GPSS/360 ASSEMBLY CONTROL CARDS

SIMULATE Card

If a GPSS simulation run is desired, a SIMULATE card must be present. Two forms of the SIMULATE card are acceptable: a* in column 1 and the word SIMULATE beginning in column 8; if this form is used, this card must be the first or second card in the model. The other form is a card with the word SIMULATE beginning in column 8; if this form is used, the card may appear anywhere in the data deck.

If no SIMULATE card is present, the job will terminate after the assembly phase.

JOB Card

Frequently an analyst wishes to assemble (and often simulate) more than one model. This may be accomplished by inserting a JOB card between each model. When a JOB card is recognized by the assembly phase it completes processing the preceding cards and then transfers control to the simulator

phase which will transfer control back to the assembly phase after execution of that model. This process will be repeated until all jobs have been processed.

The format for this card is the word JOB beginning in column 8.

END Card

An END card is the last card of an input deck. It performs the function of informing GPSS that all cards have been presented.

The format for the END card is the word END beginning in column 8.

PSEUDO-OPERATIONS

The assembly phase also recognizes five pseudo-operations: ORG, ICT, SYN, ABS, and ENDABS. The pseudo-operations are included to give the user some control over the allocation of block addresses.

ORG (Origin)

This card is used to set the next block number to be assigned by the assembler to a desired value. The work ORG appears beginning in column 8. A number or symbol appears beginning in column 19.

The value of the entry beginning in column 19 is the next block number to be assigned. If this entry is a symbol, the symbol must have been previously defined. In this case, the block number assigned to the symbol will be the next block number assigned.

ICT (Increment)

The ICT pseudo-operation is used to increment the next block number to be assigned by the assembler by a desired value. The word ICT appears beginning in column 8 and a numeric value appears beginning in column 19. The current block count is incremented by the value specified beginning in column 19 and the sum is used by the assembler as the next block number to be assigned.

The ORG and ICT pseudo-operation should be used very carefully.

SYN (Synonymous)

This card is used to equate one block symbol with another. The word SYN appears beginning in column 8, a symbol appears beginning in column 2 and a symbol appears beginning in column 19. The symbol in the location field is assigned the same block number that was assigned to the symbol in the operand field. This assumes the symbol in the

operand field is already defined. The symbol in the location field must be a legal and undefined symbol.

ABS and ENDABS

The ABS pseudo-operation (the word ABS beginning in column 8) is used to specify where absolute GPSS coding is to be used. The ENDABS pseudo-operation (the word ENDABS beginning in column 8) is used to discontinue the absolute GPSS mode.

When the assembly phase is operating in absolute mode it simply reads each card to determine if it is an ENDABS card and if not, writes this in the simulator input device without further examination. The assembler assumes that every card between ABS and ENDABS card has no symbolic coding in any of its fields and is in a fixed-field format. This precludes the use of a card containing a matrix savevalue or free-form function follower cards between ABS and ENDABS words because these cards must be processed by the assembly phase.

Care must be taken when absolute and symbolic coding are used in the same program. A JOB card used during absolute mode, will not be recognized, and the job that follows will be considered an extension of the preceding one. In order to have the JOB card recognized, the pseudo-operation ENDABS must appear before the JOB card. An ABS card could immediately follow the JOB card and thus indicate that the next job is to begin in the absolute mode.

The assembler block counter is not changed by the absolute coding mode; when this mode is ended, the block counter will start assigning block numbers at the point where it stopped previously. Because of this, the analyst must ensure that the absolute block numbers will not be the same as the assembler-assigned block numbers. To eliminate this problem, ORG or ICT pseudo-operations may be used to change the block counter.

ERROR STATEMENTS

Errors are indicated by the statement:

ERROR NUMBER(S) W, X, Y, Z

below the card which contains the error or errors, where W, X, Y, and Z are error numbers. If a card contains more than four errors, the fourth, ... N-1 errors found will not be printed out, only the first, second, third, and Nth error will appear.

MULTIPLE DEFINITION OF BLOCK SYMBOLS

The assembly phase does not consider multiple definition of a block symbol as an error. This is because the analyst may wish to redefine a block which has a symbolic location one or more times in a simulation job. However, it is possible that

the analyst might mistakenly assign the same symbolic location to two separate blocks in a model. Therefore, when such a situation is encountered, the message "MULTIPLE DEFINITION OF SYMBOL IN ABOVE CARD" is printed below the card containing the multiple symbol definition.

APPENDIX C: OUTPUT EDITOR

The GPSS/360 Output Editor allows the user to select statistics from and modify the standard output provided by GPSS/360 into a format more appropriate to a given application. The various request cards associated with the output editor allow the user to:

1. Select the output statistics of interest.
2. Title and/or comment at appropriate sections of the output.
3. Control spacing, page skipping, and order of output.
4. Display GPSS/360 SNA values in graphic format.

GPSS/360 Output Editor request cards can be categorized into two areas:

1. Selection of statistics, titling, comments, and spacing
 2. Graphical representation of SNA values
- The request cards associated with selection of statistics, titling, spacing, etc., are:

REPORT
TITLE
INCLUDE
FORMAT
TEXT
COMMENT
EJECT
SPACE
OUTPUT

The request cards associated with graphic output are:

GRAPH
ORIGIN
X
Y
STATEMENT
ENDGRAPH

The format and a complete description of the GPSS/360 Output Editor request cards follows.

SELECTION OF STATISTICS, TITLING,
COMMENTS, AND SPACING

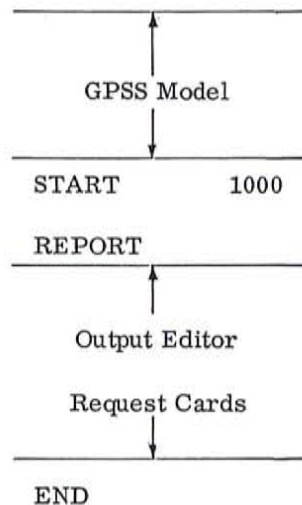
REPORT Card

To obtain the services of GPSS/360 Output Editor, the user must submit the appropriate GPSS/360

Output Editor request cards immediately after the last GPSS/360 START card and before the GPSS/360 END or JOB card associated with a given simulation model. The first of these cards must be the REPORT card, which has the following format:

8
REPORT

The REPORT card is then followed by the Output Editor request cards, which may appear in any order. The general deck format when using the GPSS/360 Output Editor is:



The output format specified by the Output Editor request cards will be applicable for all GPSS/360 START cards associated with a given model. This also includes SNAPS if specified in the START cards. It is not possible to redefine Output Editor request cards when using multiple START cards etc.

GPSS/360 execution errors will unconditionally result in the standard GPSS/360 error output regardless of any request for Output Editor services.

Each Output Editor request card is processed independently and is not completely checked for errors until output time. If an error in an Output Editor request card is encountered, an error message is listed and the card is not processed. Remaining Output Editor request cards are then processed independently. If any errors are encountered while processing GPSS/360 Output Editor request cards, the

standard GPSS/360 output is unconditionally listed after the last Output Editor request card is processed.

TITLE Card

The TITLE card enables the user to title the various statistic sections, such as facilities, queues, specific tables, etc. The format for this card is:

```

2           8           19           A, B           72
entity type  TITLE     X,aaa----- c

```

The group of statistics to be titled is named beginning in column 2. The title to be printed begins in field B, and continues up to column 71, and, if necessary, may be continued on the next card beginning in column 1 and ending by column 71. If a second card is needed, a nonblank character must be punched in column 72 of the TITLE card. The title may not exceed 124 characters. Listed below are the mnemonics used, beginning in column 2, to indicate the entity type to be titled:

- *BLO-- block counts
- SAV-- fullword savevalue contents
- HSAV--halfword savevalue contents
- MSAV--matrix savevalue contents
- MHSA--halfword matrix savevalue contents
- CHA-- user chains statistics
- FAC-- facility statistics
- STO--storage statistics
- QUE-- queue statistics
- TAB-- table statistics
- GRO-- group members
- *CLO-- clock statistics

The statistics for just one particular member of any member of an entity class (for example, FACILITY 13, STORAGE 2, or MATRIX SAVEVALUE 4), may be obtained by placing the number (13, 2, or 4) in field A of the TITLE card. For example:

```

2           8           19
MSAV      TITLE     4, THE PROFIT MATRIX

```

The above TITLE request card causes the following output:

THE PROFIT MATRIX

```

MATRIX FULLWORD  SAVEVALUE  4
  COLUMN         1   2   3   4
ROW   1         0   0   73  75
      2         71  0   0   0

```

If no entry is placed in field A, all statistics associated with the entity type are printed out following the

title. The field A option of the TITLE card applies to all above entities except those marked by asterisks (CLO and BLO). Whenever these are requested in TITLE cards, the entire output associated with them is printed out regardless of any entry in field A. Symbols may also be used in field A to specify any entity, for example, F\$CPU, MX\$CHAN etc. If no field A entry is required, a comma must appear in column 19 before the TITLE information to indicate the omission of field A.

INCLUDE Card

The user may select only the entity statistics of interest by means of the INCLUDE card, which has the following format:

```

2           8           19
entity      INCLUDE  range/spec. col.of output

```

The location field contains the mnemonic for the entity that is to be printed--for example, SAV, FAC, STO, etc. The entries to the left of the slash in the operand field indicate the range of entity that is printed. The entries to the right of the slash indicate the columns of normal output statistics to be printed. For example, the first column of normal facility statistics is the facility name or number, the second column is average utilization etc. For example:

```
FAC INCLUDE F1-F10/1,2
```

As a result of the above card, the only facility statistics printed will be the facility name or number and the average utilization for facilities 1 through 10. If the user specifies zero in column 19 of an INCLUDE card, no statistical output will occur.

The INCLUDE card can be used in conjunction with the TITLE card to specify the range of entity to be included with the TITLE. If an INCLUDE card immediately follows a TITLE card of the same entity type (column 2), the restrictions of the INCLUDE card request are placed on the statistics associated with the TITLE card. For example:

```

2           8           19
FAC  TITLE     , UTILIZATION FOR TERMINALS 1-5

```

```
FAC  INCLUDE F$TERM1-F$TERM5/1,2
```

The above cards would cause the following statistics to be listed:

```

UTILIZATION FOR TERMINALS 1-5
FACILITY          AVERAGE
                  UTILIZATION
TERM1             .257
TERM2             .270
TERM3             .258
TERM4             .258
TERM5             .257

```

If the field A option is specified in the TITLE card, the INCLUDE card is treated independently of the TITLE card. For example:

```

2      8      14      19
FAC   TITLE   3      TITLE FACILITY 3
FAC   INCLUDE F1-F8/1,2,3

```

The above cards would cause the following printout:

TITLE FACILITY 3				
FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRANS.	SEIZING TRANS. NO. PREEMPTING TRANS. NO.
3	.959	28	4.214	
FACILITY	AVERAGE UTILIZATION	ENTRIES		
1	.941	26		
2	.948	27		
3	.959	28		
4	.065	2		
5	.089	2		
6	.032	1		
7	.186	5		
8	.073	2		

The following five entity types may be specified in INCLUDE card, with the previously defined format (F1-F10/1,2). The letter or letters within parentheses are to be used for the entity type in the entries to the left of the slash. The numbers designate the columns of output statistics which may be requested (to the right of the slash) for each entity type. Note that if the first number to the right of the slash is not a 1, the specific number of the entity is not printed out. The column request numbers must be in ascending order. The range request (F1-F10) must also be from low to high, and both numbers must be legal for the particular entity type. The entity mnemonics (FAC, etc.) are the same as those given for TITLE cards.

Facilities (F) 1 Facility name or number
 2 Average utilization
 3 Number of entries
 4 Average time per transaction
 5 Seizing transaction number
 6 Preempting transaction number

Storages (S) 1 Storage name or number
 2 Capacity
 3 Average contents
 4 Average utilization
 5 Number of entries
 6 Average time per transaction
 7 Current contents
 8 Maximum contents

Queues (Q) 1 Queue name or number
 2 Maximum contents
 3 Average contents
 4 Total entries
 5 Zero entries
 6 Percent zero entries
 7 Average time per transaction
 8 Average time per transaction, excluding zero entries
 9 Table number associated with queue
 10 Current contents of queue

User Chains (CH) 1 User chain name or number
 2 Total entries
 3 Average time per transaction
 4 Current contents
 5 Average contents
 6 Maximum contents

Tables (T) 1 Table name or number
 2 Entries in table--nonweighted
 3 Mean argument--nonweighted
 4 Standard deviation--nonweighted
 5 Sum of arguments--nonweighted
 6 Entries in table--weighted
 7 Mean argument--weighted
 8 Standard deviation--weighted
 9 Sum of arguments--weighted
 10 Upper limit
 11 Observed frequency
 12 Percent of total
 13 Cumulative percentage
 14 Cumulative remainder
 15 Multiple of mean
 16 Deviation from mean

The following five entity types may be specified in INCLUDE cards with just the range of members of the particular entity type that are requested in field B:

2	8	19
SAV	INCLUDE	, X2-X4
HSAV	INCLUDE	, XH2-XH4
MSAV	INCLUDE	, MX2-MX4
MHSAV	INCLUDE	, MH2-MH4
GRO	INCLUDE	, G2-G4

In each case the standard output for the entity type would be printed out only for the requested members of the type. For example:

2	8	19
GRO	INCLUDE	, G3-G4

The above card would cause the following printout:

```
CURRENT MEMBERS OF GROUP 3
TRANSACTION MODE
```

1

```
CURRENT MEMBERS OF GROUP 4
TRANSACTION MODE
```

1 3

For all INCLUDE cards, the range may be stated either numerically (F1-F5) or symbolically (F\$TERM1-F\$TERM5).

Each INCLUDE card may refer to only one entity type. With each column of statistics requested, the heading for that column of statistics is also listed. For entity statistics not requested by columns (SAV, HSAV, GRO, MSAV, MHSAV) only the entity type headings are printed, as shown in the example of the GROUP output.

FORMAT Card

A card type similar to the INCLUDE type card is the FORMAT card:

2	8	19
n	FORMAT	range/output statistics

The entries to the left of the slash indicate the range of the entities to be used. The entries to the

right of the slash indicate the output statistics to be printed.

This card type can best be explained by use of an example: Assume that 20 transmission lines are represented by facilities 1-20, and that the amount of time a message spends waiting to be serviced by the line at a terminal queue is contained in queues 1-20. Presentation of this line utilization and waiting time in tabular form could be accomplished with the following card:

8	19
FORMAT	1-20/F1, F2, Q2

This card would produce the following printout:

1	.382	.61
2	.471	1.02
.	.	.
.	.	.
.	.	.
.	.	.
20	.144	.15

In the above example the first entry to the right of the slash is F1, which indicates the first column of normal facility statistics (this is the facility name or number); the second entry indicates the second column of normal facility statistics, etc.

The entries to the left of the slash indicate the entity attributes to be associated with the output statistics, i. e., a kind of looping constant. The first output line produced by the above FORMAT card is the facility name or number (in this case the number 1), the utilization for facility 1, the maximum queue length for queue 1; the second line is the same information for facility 2 and queue 2, etc.

The following table lists the entries that may be specified to the right of the slash on a FORMAT card:

F1 Facility name or number
 F2 Average utilization of facility
 F3 Number of entries for facility
 F4 Average time per transaction for facility

S1 Storage name or number
 S3 Average contents of storage
 S4 Average utilization of storage
 S5 Number of entries in storage
 S6 Average time per transaction in storage
 S7 Current contents of storage
 S8 Maximum contents of storage

- Q1 Queue name or number
- Q2 Maximum contents of queue
- Q3 Average contents of queue
- Q4 Total entries in queue
- Q5 Zero entries in queue
- Q6 Percent zero entries in queue
- Q7 Average time per transaction in queue
- Q8 Average time per transaction in queue, excluding zero entries
- Q10 Current contents of queue

- T1 Table name or number
- T2 Entries in table--nonweighted
- T3 Mean argument for table--nonweighted
- T4 Standard deviation for table--nonweighted
- T6 Entries in table--weighted
- T7 Mean argument for table--weighted
- T8 Standard deviation for table--weighted

- X1 Savevalue name or number
- X2 Contents of savevalue

- XH1 Halfword savevalue name or number
- XH2 Contents of halfword savevalue

- CH1 User chain name or number
- CH2 Total entries in user chain
- CH3 Average time per transaction in user chain
- CH4 Current contents of user chain
- CH5 Average contents of user chain
- CH6 Maximum contents of user chain

One advantage of the FORMAT card over the INCLUDE card is that the user can mix entity types with the FORMAT card. As seen in the previous example, the first two columns of statistics were associated with facilities and third column with queues.

Also with the FORMAT card, the user can specify the print column in which to start the first column of statistics. The user cannot control spacing between subsequent columns of statistics. The starting print column is specified in column 2 of the FORMAT card. If this column is blank, the statistics will start in print column 1. No statistics can be printed past the 132nd print position. Eighteen columns are used for each statistic type specified by the FORMAT card. The user should consider this figure as a guide when setting up titles, etc. for statistics that are listed by means of the FORMAT card.

When using the FORMAT card, no automatic titling of column statistics is given. Appropriate titles can be inserted by use of either a TEXT card

or a COMMENT card immediately before the FORMAT card.

Any FORMAT card in error results in a printout of the card followed by an error message.

TEXT Card

This card allows the user to intermix numerical output data with alphanumeric data in sentence form.

2	8	19	72
n	TEXT	alphanumeric & data c	

The number n beginning in column 2 indicates the starting print position. If this is blank, the starting print position is assumed to be 1. The text to be printed begins in column 19 and continues up to column 71 and, if necessary, may be continued on the next card beginning in column 1 and ending by column 71. If a second card is needed, a nonblank character must be punched in column 72 of the TEXT card.

Data obtained and the way it is printed is distinguished from ordinary alphanumeric by preceding and following the information by the character #. This data and the desired output format for the data are presented as follows:

#data/format#

The operations associated with this card type can best be explained by use of an example:

2	8	A
5	TEXT	CPU UTILIZATION IS #F1,2/2RXX#%

The entry after the first # (F1,2) indicates that the data obtained is the average utilization (the second column of facility output) for Facility 1. The entry to the right of the / (2RXX) indicates the manipulation performed on the data obtained. This entry indicates that the decimal point of the data (average utilization of Facility 1) is to be moved two places to the right (2R). The final part of the entry (XX) indicates that only the two characters to the left of the decimal point, in its new position, are to be printed.

The above card would cause the statement

CPU UTILIZATION IS 38%

to be printed out beginning in column 5, assuming that the utilization of Facility 1 is .3826.

In general, the format presentation consists of

X's (representing digits), a decimal point, and possibly an instruction to move the decimal point to the left or right.

The first operation is to move the decimal point, if this is indicated, by a number followed by either R or L: 1L -- move the decimal point 1 place to the left; 4R -- move the decimal point 4 places to right. This is performed on the actual data.

The next step consists of printing the number of digits to the left and right of the decimal point as indicated in the format presentation. For example:

Source Data	Format in TEXT Card	Printed Result
14238	3LXX.X	14.2
.12856	2RXXX.X	12.8
1581	XXX.	ERROR SIGNIFICANT DIGIT HAS BEEN IGNORED, SOURCE DATA IS 1581
32.456	1LXX.XX	3.24

The following table illustrates the entries which may be specified to the left of the slash on a TEXT card:

Fn,2 Average utilization of facility n
Fn,3 Number of entries for facility n
Fn,4 Average time per transaction for facility n

Sn,3 Average contents of storage n
Sn,4 Average utilization of storage n
Sn,5 Number of entries for storage n
Sn,6 Average time per transaction for storage n
Sn,7 Current contents of storage n
Sn,8 Maximum contents of storage n

Qn,2 Maximum contents of queue n
Qn,3 Average contents of queue n
Qn,4 total entries of queue n
Qn,5 Zero entries for queue n
Qn,6 Percent zeros for queue n
Qn,7 Average time per transaction in queue n
Qn,8 Average time per transaction in queue n excluding zero entries
Qn,10 Current contents of queue n

Tn,2 Entries in table n--nonweighted
Tn,3 Mean argument for table n--nonweighted
Tn,4 Standard deviation for table n--nonweighted
Tn,6 Entries in table n--weighted
Tn,7 Mean argument for table n--weighted
Tn,8 Standard deviation for table n--weighted

Xn,2 Contents of savevalue n
XHn,2 Contents of halfword savevalue n

CHn,2 Total entries for user chain n
CHn,3 Average time per transaction for user chain n
CHn,4 Current contents of user chain n
CHn,5 Average contents of user chain n
CHn,6 Maximum contents of user chain n

The following Output Editor cards:

```
TEXT AV TIME/XACT #CH3, 3/XXX# UNITS
SPACE 2
TEXT SAVEX 3=#X3, 2/XXX# IN BIG
TEXT SAVEX 3=#X3, 2/2LXX.XX# IN DECIMAL
TEXT LITTLE SAVE 2 #XH3, 2/XXX# ANSWER
TEXT MAXIMUM Q CONTENTS=#Q1, 4/XXX#
PROBABLY
```

would result in the following printout:
AV TIME/XACT 55 UNITS

```
SAVEX 3=11 IN BIG
SAVEX 3=.11 IN DECIMAL
LITTLE SAVE 2= 7 ANSWER
MAXIMUM Q CONTENTS=360 PROBABLY
```

The user should have a good idea of what his output will look like before using TEXT cards. Since an exact count on digits is required, an incorrect count will result in no processing of the TEXT card. Instead the card will be printed out with an error message and the source data that should have been inserted.

For example:

```
8 19
TEXT #F1, 2/2RX.XX#
```

will result in the printout: TEXT #F1, 2/2RX.XX#
SIGNIFICANT DIGIT WILL BE IGNORED IF
SPECIFICATION IN ABOVE STATEMENT IS
FOLLOWED. SOURCE DATA IS .625.

By shifting the decimal point two places to the right, the result is 62.5. Since the request was for X.XX (only one place to the left of the decimal point) a significant digit would be ignored, therefore the error.

The decimal point may be shifted to the right only n + 1 places where n is the number of decimal digits normally considered for this statistic. If the requested right shift is for more than n + 1 places, blanks will be filled in.

If facility utilization (column 2) is .315 and number of entries (column 3) is 36 then the following cards:

- (1) #F1,2/4RXXXXX#
- (2) #F1,2/1RXXXXX.X#
- (3) #F1.2/5RXXXXXX#
- (4) #F1.3/XX#
- (5) #F1,3/1RXXXXX#
- (6) #F1,3/2RXXXXX#

would result in

- (1) 3150
- (2) 3.1
- (3) 3150
- (4) 36
- (5) 360
- (6) 360

COMMENT Card

The COMMENT card is similar in format to the standard GPSS/360 COMMENT card in the sense that an asterisk appears in column 1. The contents of the COMMENT card will be printed out with column 2 of the card being listed in the first print position.

For example:

```
1
* TO COMMENT THE MODEL would result in the
printout:
```

```
    TO COMMENT THE MODEL
```

The user can continue his comment on a second card (not to exceed 132 print positions) by placing a nonblank character in column 72 and starting the continuation card in column 1.

EJECT Card

The EJECT card enables the user to skip to a new page before further output requests are serviced. The format of the EJECT card is:

```
8
EJECT
```

SPACE Card

The SPACE card enables the user to skip a specified number of lines. The format of the SPACE card is:

```
8      19
SPACE  # of lines to skip
```

The SPACE card will skip the number of lines specified in column 19 which may be 1, 2, or 3.

When using the services of the GPSS/360 Output Editor, no page skipping or line spacing is done unless it is specifically requested by use of the EJECT and SPACE cards.

OUTPUT Card

The OUTPUT card enables the user to obtain the standard GPSS/360 output in addition to the Output Editor printout. The format of the OUTPUT card is:

```
8
OUTPUT
```

The standard output will start on a new page. After it is complete, the Output Editor will skip a page and continue to process any additional Output Editor request cards. As a general rule, the OUTPUT card should always be used the first time a simulation run is made using Output Editor request cards so that if the Output Editor cards are not what the user intended, the complete output statistics are available. If any errors in Output Editor cards are encountered, standard output will follow the complete processing of all Output Editor request cards unless standard output had been specifically requested previously by an OUTPUT card.

GRAPHIC OUTPUT

The user may obtain graphic representation of GPSS/360 SNA values by use of the graphic request cards associated with the GPSS/360 Output Editor. Each graph may be considered as consisting of a 60 row by 132 column matrix. Within this matrix a histogram can be structured and displayed. The histogram can be visualized as a series of rectangles where each rectangle represents a specific member of an entity type (Facility 1, Storage 5, User Chain 20, etc.) and the height of the rectangle would represent the value of the SNA associated with a specific member of an entity type.

By use of the graphic request cards, the user has the ability to:

1. Specify the entity SNA to be plotted.
2. Specify the character to be used in plotting the graph.
3. Specify the origin for the X and Y axes.
4. Specify the range of the entity type (Facility 1 - Facility 15).
5. Specify the values to be assigned to the X and Y ordinate points and to their corresponding axes.
6. List alphameric information and/or comments on the graph.

Requests for graphic output may appear anywhere in the sequence of Output Editor request cards. The only restriction is that the request cards used to specify a graph must be ordered as follows:

```
GRAPH
ORIGIN
X
```

Y
STATEMENT }
STATEMENT } if any
ENDGRAPH }

If the request cards are not in the correct order or in error, the graph will not be processed and the Output Editor will proceed to the ENDGRAPH card or to the next legal Output Editor request card.

GRAPH Card

The GRAPH card is used to specify the entity SNA to be plotted, the range of the entity to be plotted, and the character to be used in plotting the graph. The general format of the GRAPH card is:

8	19			
GRAPH	A SNA to be plotted	B Lower limit of entity range to be plotted. Table no. if field A specifies an SNA marked with an asterisk.	C Upper limit of entity range to be plotted.	D Character to be used in plotting graph if blank, an asterisk is assumed.

The A field specifies the entity SNA to be plotted which must be one from the following lists.

Facilities

FR Facility utilization
FC Facility entry count
FT Facility average time per transaction

Storages

SR Storage utilization
SA Storage average contents
S Storage current contents
SM Storage maximum contents
SC Storage entry count
ST Storage average time per transaction

Groups

G Current contents of group

User Chains

CA User chain average contents
CH User chain current contents

CM User chain maximum contents
CC User chain entry count
CT User chain average time per transaction

Queues

QA Queue average contents
Q Queue current contents
QM Queue maximum contents
QC Queue entry count
QZ Queue zero entries
QT Queue average time per transaction
QX Queue average time per transaction (excluding zero entries)

Fullword Savevalues

X Fullword savevalue contents

Halfword Savevalues

XH Halfword savevalue contents

Tables

TC Table entry count
TB Table mean
TS Table standard deviation
*TF Observed frequencies
*TP Per cent of total
*TD Cumulative percentage
*TR Cumulative remainder

Blocks

N Block counts

When using any of the above SNA's, except those marked with an asterisk, the user must specify the lower and upper limit of the entity class to be plotted, for example, FR for Facility 5 through Facility 15, etc. The field B of the GRAPH card is used to specify the lower limit and the C field is used to specify the upper limit of the entity class to be plotted. These entries may be specified symbolically or numerically with the restriction that the field B represents a smaller integer value than the field C. For example:

8	19
GRAPH	FR, LINE, CPU
GRAPH	X, 10, 15
GRAPH	QM, 3, 10

The first GRAPH card in the above example requests that the utilization for FACILITIES LINE through CPU inclusive be plotted. The second GRAPH card requests that the contents of fullword SAVEVALUES 10 through 15 inclusive be plotted. The third GRAPH card requests that the maximum contents observed for QUEUE's 3 through 10 inclusive be plotted.

The SNA's in the previous list marked with an asterisk represent requests for a graph which will plot the values of the frequency classes associated with a given TABLE. When using the SNA's marked with an asterisk, the TABLE number is specified in the field B of the GRAPH card and the field C is left blank.

For example:

```

8          19
GRAPH     TR,TAB1
GRAPH     TR,TAB2
GRAPH     TP,TAB3
GRAPH     TD,TAB4
  
```

The first GRAPH card in the above example plots the observed frequency value of the frequency classes associated with TABLE TAB1. The second GRAPH card plots the cumulative remainder of the frequency classes associated with TABLE TAB2. The third GRAPH card plots the percentage of total of the frequency classes associated with TABLE TAB3. The fourth GRAPH card plots the cumulative percentage of the frequency classes associated with TABLE TAB4. For these examples, the upper limit of the various frequency classes are labeled on the X axis of the graph, and range of the specific SNA value, on the Y axis of the graph.

The character normally used in the actual plotting of the graph axes and the rectangles is an asterisk. If a character other than an asterisk is used in plotting the graph, the character must be specified in the field D of the GRAPH card. For example:

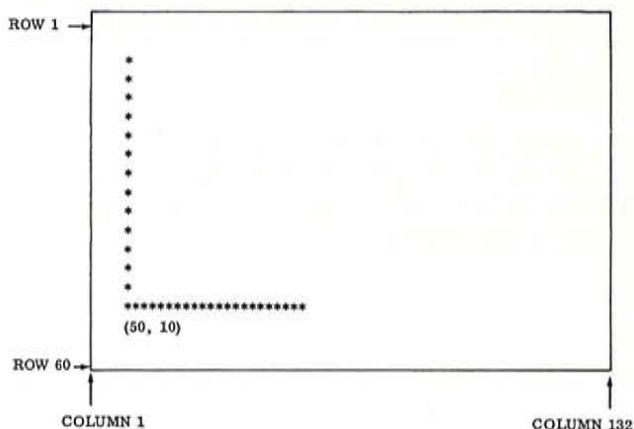
```

8          19
GRAPH     FR,2,5,+
GRAPH     TR,3,,.
  
```

The first GRAPH card causes the axes and rectangles of the resultant graph to be plotted with the + character. The second GRAPH card causes the axes and rectangles of the resultant graph to be plotted with the . character.

ORIGIN Card

As previously mentioned, the graph is plotted in what can be considered a 60 row by 132 column matrix. This is illustrated in the following diagram.



Row 1 is at the top of the page, and row 60 is at the bottom of the page. Column 1 is the leftmost column, and column 132 is the rightmost column on the page. The ORIGIN card enables the user to specify where within the 60 row by 132 column matrix the X and Y axes are to intersect. The general format of the ORIGIN card is:

8	19	
	A	B
ORIGIN	Row specifying where X axis should be plotted.	Column specifying where Y axis should be plotted.

The A field of the ORIGIN card specifies the row where the X axis is plotted, and the field B specifies the column where the Y axis is plotted. Entries in both fields A and B must be numeric quantities and the ORIGIN card must follow the GRAPH card and precede the X card. For example:

```

8          19
ORIGIN    50,10
  
```

The ORIGIN card specifies the x axis to be plotted on the 50th row and the Y axis to be plotted on the 10th column. The axes plotted would be as shown by the asterisks in the previous diagram. Note that the origin indication (50,10) shown in the diagram would not appear in the actual graph. It is shown only to illustrate the relative positions set up by the ORIGIN card.

X Card

The X card enables the user to specify the following:

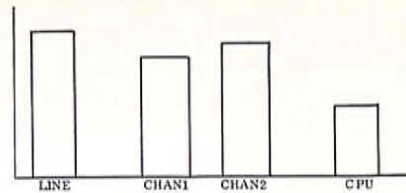
1. If labeling for the X axis should be listed.
2. If symbols or numerics should appear in the labeling of the X axis for the members of the entity class being plotted.
3. The width of the rectangles associated with the SNA of the member of the entity class being plotted.
4. The spacing between the rectangles.
5. Combination of frequency classes when plotting TABLE SNA's

There are two general formats for the X card. The first (format 1) is associated with all SNA's previously mentioned except those marked with asterisks. The second format (format 2) is associated with the SNA's marked with asterisks, namely: TF, TP, TD, and TR. The X card must follow the ORIGIN card and precede the Y card. The general format 1 of the X card is:

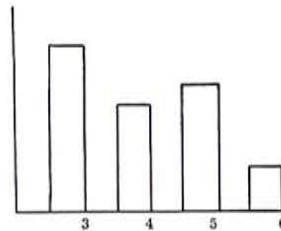
8	19						
	A	B	C	D	E	F	G
X	SYM if symbolics should be listed on X axis. If blank, numeric values will appear on X axis labeling.	Width (columns) including end points for the rectangles. If blank, 1 is assumed.	Spacing between rectangles. If blank, 1 is assumed.	-	-	-	X label indicator. No, if no symbolic or numeric values are to be listed in labeling the axis.

The field A is used to indicate whether symbolics or numerics appear in the X axis label. If symbolics are desired, SYM should appear as the field A entry. If numerics are desired, the field A should be left blank. If SYM appears in the field A, the symbols

for the entity class members appear under the rectangle associated with each member as shown in the following diagram.



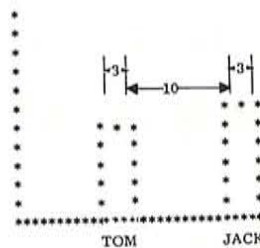
If symbols are not used in the model, and if they are not to be printed as the X axis labels, the field A should be left blank. If the field A is blank, the X axis label for the graph appears as:



Fields B and C specify the width (including end points) of the rectangles and spacing between rectangles, respectively. Both field B and C entries must be specified numerically. If field B is left blank, it is assumed to be a 1. The sum of field B and C entries must not be less than four or an error message will be given and the graph will not be processed. For example

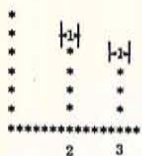
```
8          19
X          SYM, 3, 10
```

would result in:



8 19
X , , 6

would result in



If the user wants no labels listed for the X axis, NO should be specified as the G field entry of the X card.

The general format 2 of the X card is:

For TABLE SNA's TP, TF, TD and TR

Since no symbolics are associated with the frequency classes of TABLES, the field A of the X card must be blank when plotting the TABLE SNA's TP, TF, TD, and TR.

The field B specifies the width of the rectangle for the frequency classes when plotting the TABLE SNA's TF, or TP. The field B entry must be a numeric quantity. Since the TABLE SNA's TD, and TR can be visualized as cumulative distributions, only a single point (rather than a rectangle) is plotted per frequency class. Therefore when plotting the TABLE SNA's TR, or TD, the field B of the X card should be blank.

The field C specifies the spacing between the rectangles or points depending on the SNA requested. The field C entry must be a numeric quantity.

Fields D, E, and F of the X card using format 2 allow the user to condense and/or select the TABLE frequency classes of interest.

8	19	A	B	C	D	E	F	G
X		Must be blank for Format 2	For TF and TP, Width (including End Points) for the Rectangle. For TD and TR, blank.	Spacing Between Rectangles (TF and TP). Spacing Between Points (TD and TR).	Upper Limit of Lowest Frequency Class to be Plotted.	No. of Frequency Classes to be Included per X Axis Increment. If Blank, 1 is Assumed.	No. of Increments to be Plotted.	X label Indicator NO. If no Numeric Values are to be Listed on the X Axis Label.

The field D specifies the upper limit of the lowest TABLE frequency class to be plotted. This must be a numeric quantity and should not specify a limit less than that associated with a particular TABLE specification.

The field E is the number of frequency classes per X axis increment. If this field is left blank, it will be interpreted as 1. The field F has the number of increments to be plotted on the X axis. Both fields E and F must be specified numerically. The product of fields E and F must not exceed the number of frequency classes associated with the referenced TABLE.

The field G is the X axis label indicator. If NO is the field G entry, the X axis label will not be automatically computed and listed. For example:

A TABLE has 20 frequency classes. The upper limit of the lowest frequency class is 20. The interval between frequency classes is 10.

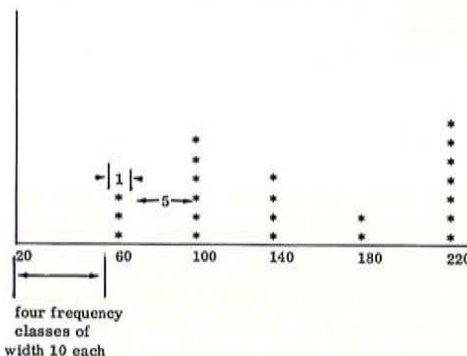
Assuming the TABLE SNA TP is specified, and all frequency classes are plotted individually on the graph, the X card entries would be:

8 19
X , 2, 5, 20, 1, 20

If the frequency classes were condensed into five different categories (each including four frequency classes) for plotting purposes, the X card entries would be:

8 19
X , 1, 5, 20, 4, 5

The above X card would result in

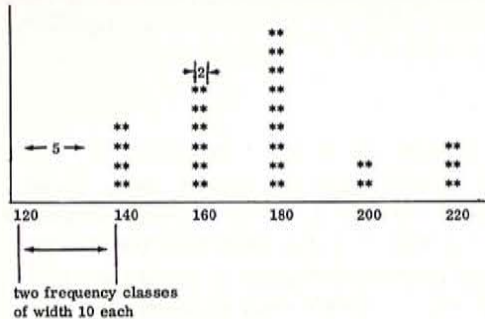


If only the upper frequency classes are of interest, they are plotted in more detail by specifying an X card as follows:


```

      8      19
      X      , 2, 5, 120, 2, 5
  
```

The above X card would result in



Y Card

The Y card enables the user to:

1. Specify the lower limit of values for the Y axis.
2. Specify the number of increments, and number of rows/increment to be allocated for the Y axis.

This information is used for automatically labeling the Y axis.

The Y card must follow the X card and precede STATEMENT cards. The general format for the Y card is:

	8	19		
	A	B	C	D
Y	Lower limit for Y axis label.	Increment size for Y label.	Number of increments to be included on the Y axis.	Number of rows to be allocated for each increment.

All fields of the Y card must have numeric quantities as entries. The field A specifies the lower limit of the Y axis, and the field B specifies the size of each increment to be made on the axis. The field C specifies the number of increments to be included on the Y axis, and the field D specifies the number

of rows to be allocated for each increment. For example:

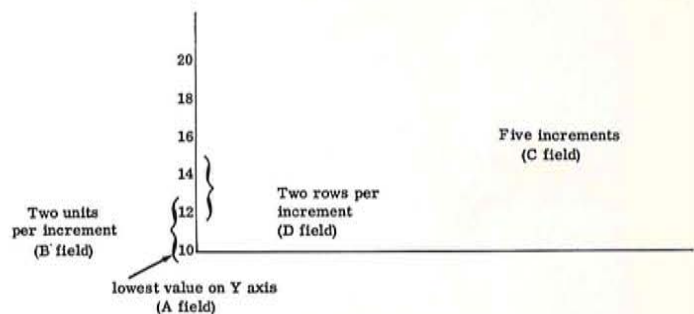
Plot SAVEVALUE contents whose values range between 10 and 20.

Five increments should be included with an increment size of 2.

```

      8      19
      Y      10, 2, 5, 2
  
```

The above Y card would result in the following Y axis labels and organization.



The product of the field C (number of increments) and field D (rows per increment) must be less than the number of rows available (field A of ORIGIN card).

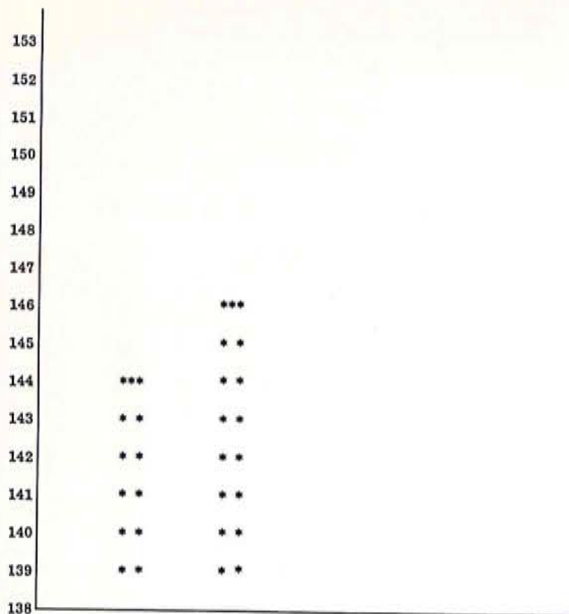
Fields A and B of the Y card control not only the actual labels to be listed, but also the height of rectangles plotted. Therefore when these fields are specified, the user should know the range to be plotted and the general range which the SNA values will cover. If the above card were used and the SAVEVALUE contents were in the range of 0 to 10, no rectangles would be plotted since all values are below the lowest limit specified for the Y axis card. Also, if all values being considered were between 140 and 150, the user would want a graph which emphasized the distinction between these values. For example:

```

      8      19
      Y      138, 1, 15, 1
  
```

The above Y card would result in

The above Y could result in



With such a graph the user could differentiate between, for example, 145 and 147; but if a more general scheme were used, these two values would appear the same.

When setting up the Y axis card for a particular SNA, the analyst should note the format of the output for that SNA. SNA's regarding percentage are taken from 0 to 100. These include the TABLE SNA's TP, TD, and TR. Decimal values may only be used for FR (facility utilization) and SR (storage utilization). The user can specify the Y card as:

```

      8      19
      Y      .25,.05,10,2
  
```

The above card would plot the utilizations ranging from .25 through .75 (ten increments of .05 each). These decimal values may be from one (.1) to three digits (.103) each.

The first character of both fields A and B must be a decimal point for these two SNA's. For all other SNA's, only integer values are considered and no decimal points should be used in field A and B entries of the Y card.

STATEMENT Card

The STATEMENT card enables the user to place alphameric information anywhere within the 60 row by 132 column matrix in which the graph appears.

The general format of the STATEMENT card is:

2	8	19	72		
Starting column for the statement.	STATEMENT	A Row on which the statement should be listed.	B Number of characters in the statement.	C Statement information to be listed.	1 1 if statement continuing on next card.

The entry beginning in column 2 specifies the column in which the statement should start. If no entry is made in column 2, the statement begins in column 1. The field A entry specifies the row in which the statement information is to be listed. This must be one of the 60 rows allowed for the graph. Again, the top row is row 1 and the bottom row is row 60.

The field B is the number of characters which make up the statement. Blanks are counted as characters. The actual statement follows the field B entry. If the entire statement cannot be specified on a single card, a 1 should be punched in column 7 and the statement continued beginning in column 1 of the next card. Any number of STATEMENT cards may be used for a given graph. The only restriction is that the STATEMENT cards must be ordered ascendingly by the row in which the statements are listed. For example:

```

      2      8      19
    100    STATEMENT    3,12,FIGURE NO. 1
    100    STATEMENT    4,14,FACILITY
                          USAGE
  
```

Both above statements are listed starting in column 100: the first statement, in row 3 beginning in column 100; the second statement, in row 4 beginning in column 100.

ENDGRAPH Card

The ENDGRAPH card is the last card associated with graph specification. The ENDGRAPH follows the last STATEMENT card associated with a given graph. The format for the ENDGRAPH card is:

```

      |8
      |ENDGRAPH
  
```

The user may request any number of graphs to be listed. The graph specification cards (for a given graph) may be intermixed with other Output Editor request cards if desired. The only restriction is that the group of GRAPH cards for a given graph must appear in a fixed uninterrupted sequence. For example:

For example:

1 2
 FAC
 FAC

20

*

TAB

TABLE

8
TITLE
INCLUDE
GRAPH
ORIGIN
X
Y
STATEMENT
ENDGRAPH
EJECT
1 WILL FOLLOW
TITLE 1
GRAPH
ORIGIN
X
Y
ENDGRAPH
END

19
UTILIZATION OF CHANNELS
F1-F4/1, 2
FR, CHAN1, CHAN4
50, 10
SYM, 3, 5
.100, .100, 6, 4
58, 20, FACILITY UTILIZATION

TABLE OF TRANSIT TIMES
TF, 1
50, 10
, 4, 2, 30, 2, 6
10, 5, 20, 2

INDEX

- ABS card: 226
- Absolute clock time: 4
- ADD card: 224
- ADVANCE block: 5, 60-62
- Allocation: (see "core allocation")
- ALTER block: 119
- Arithmetic variable: 1, 9, 12, 20-25
 - Examples: 21, 22
- ASSEMBLE block: 88-91
 - Example: 91
- Assembly program: 196, 215-227
 - Sample output: 218-222
- Assembly sets: 16, 86, 87
- ASSIGN block: 78
 - Examples: 79, 80
- ASTERISK card: 18
- Attributes: (see "logical attributes", "standard numerical attributes", "system numerical attributes")

- Block diagrams: 4
- Block definition cards: 38, 215
- Blocks: 1, 6, 38-52
- Boolean variables: 9, 23-25
 - Examples: 24, 25
- BUFFER block: 96, 97
 - Examples: 104, 106, 107
- BUFFER option: 96, 97
 - Examples: 104, 106

- Card format: 18
 - Block: 38, 215
 - Function definition: 27
 - Initial: 123, 124
 - Matrix definition: 121
 - Remarks card: 18
 - Storage definition: 156
 - Table definition: 179
- Variable definition: 20-22
- Chains: (see "current events chain", "delay chains", "future events chain", "interrupt chain", "matching chain", "user chains")
- CHANGE block: 42
- Changes: (see "event changes", "status changes")
- CLEAR card: 191, 192
 - Effect on blocks: 40
 - Facilities: 152, 155
 - GENERATE blocks: 68
 - Queues: 174, 175
 - Storages: 160, 161, 167
 - Tables: 184, 186
- Clock time
 - Relative: 4, 14
 - Absolute: 4

- Constant: 11
- Continuous numerical valued function: 26-29
- Control cards: 3, 189-194
- Control programming example: 126, 127
- Core allocation: 1, 18
 - Blocks: 40, 41
 - Boolean variables: 24
 - Facilities: 138, 139
 - Functions: 36, 37
 - Logic Switches: 130
 - Matrix Savevalues: 122, 123
 - Queues: 169
 - Storage: 164, 165
 - Tables: 178, 179
 - Transactions: 53, 54
 - User chains: 70
 - Variables: 22
- COUNT block: 39, 81
- CREATE card: 224
- Cumulative distributions: 32
- Cumulative time integral
 - Facilities: 149, 150
 - Queues: 170, 174
 - Storages: 158, 165, 166
- Current events chain: 16, 55, 58

- Definition cards: (see "card format")
- Delay chains: 100, 134, 136, 137, 140, 141, 144, 158, 159
- Delay indicator: 57, 65
- DEPART block: 170, 171
 - Examples: 174-176
- Diagrams, block: 4, 44
- Differences between GPSS III and GPSS/360: 195-199
- Discrete attribute valued function: 30
- Discrete numerical valued function: 29
- Distributions: 31
- Distribution tables: 2, 8, 177-186
 - Examples: 180, 181

- EJECT card: 234
- ENDABS card: 226
- END card: 187, 192, 226
- ENDGRAPH card: 235, 240
- ENDMACRO card: 222
- ENTER block: 157, 158
 - Examples: 162-164
- Entities: 5, 18
 - Symbolic EQU 215 reference: 215-217
 - (See also "blocks", "facilities", "functions", "groups", "logic switches", "matrix savevalues", "queues", "savevalues", "storages", "tables", "transactions", and "variables".)
- Errors
 - Assembly: 204-207

Input: 207-210
Execution: 210-213
Event changes: 16
EXAMINE block: 117
EXECUTE block: 42
Exponential distribution: 33

Facilities: 2, 7, 138-154
Floating-point variable: 23
FORMAT card: 231
Free format function follower cards: 28, 197
Functions: 1, 9, 26-37
Future events chain: 16, 55

GATE block: 39, 85
LR, LS: 86, 131
M, NM: 86, 94, 95, 96
SE, SF, SNE, SNF: 85, 86, 159, 160
I, NI, NU, U: 85, 150, 151
GATHER block: 91-93
GENERATE block: 66-69
GRAPH card: 234, 235
Graphic output: 234-241
Groups: 114, 115

HELP block: 110-114

ICT (increment) card: 226
INCLUDE card: 229
INDEX block: 80
Indirect addressing: 15, 16
INITIAL card: 123-125
Interrupt chain: 16, 60
Introduction: 1

JOB card: 192, 225
JOBTAPE card: 192, 193
JOIN block: 115

LEAVE block: 158, 159
Examples: 162-164
LINK block: 70
Examples: 74-78
LIST card: 193
List attribute valued function: 30
List numerical valued function: 29
LOGIC block: 131
Logical attributes: 8
Facilities: 138
Logic switches: 131
Storages: 156, 157
Transactions: 55
Logic switches: 1, 7, 130-137
LOOP block: 83

MACRO card: 220
Examples: 220, 221
Mapping functions: 34
MARK block: 81
Mark time: 53, 81
MATCH block: 93, 94
Examples: 94, 95, 96
Matching chain: 16, 60
MATRIX definition card: 121
Matrix savevalues: 2, 121
MSAVEVALUE block: 122
Multiple queues: 171, 172

Numerical attributes
Blocks: 40
Facilities: 138, 198
Groups: 115, 198
Matrix savevalues: 120
Queues: 168
Savevalues: 120
Storages: 156
Tables: 177
Transactions: 53-55, 198
User chains: 70, 198

ORG (origin) card: 226
ORIGIN card: 234, 235
OUTPUT card: 234
Output editor: 228
Output: (see "statistical output")

Packing, function: 33
Parameters: 15, 53 (see also "assign", "index",
loop", "mark", and "split")
Parameter transit time: 53
Polling example: 133
Practical usage of GPSS/360: 199-203
PREEMPT block: 142-148
Examples: 153
PRINT block: 187, 196
PRIORITY block: 96, 97
Examples: 104-108
Priority class table: 59, 60
Probability distributions: 31, 33
Pseudo operations: 226

QTABLE card: 171
Queues: 2, 8, 168-176
QUEUE block: 168, 170
Examples: 174-176

READ/SAVE feature: 193
Reallocation of entities: 18, 19
Redefinition of blocks: 41, 69

Functions: 36
 Variables: 22
 Matrix savevalues: 123
 Storage capacity: 162
 Tables: 184
 Relative addressing: 216
 Relative clock time: 4, 9
 RELEASE block: 140-142
 Remarks cards: 18
 REMOVE block: 116
 REPLACE card: 224
 REPORT card: 228
 RESET card: 190, 191
 Effect on blocks: 40
 Facilities: 151, 152, 155
 Storages: 161, 167
 Queues: 174, 175
 Tables: 186
 User chains: 71
 RETURN block: 148-150
 REWIND card: 193
 Run control: 69, 190

 SAVE card: (see "READ/SAVE feature")
 SAVEVALUE block: 120
 Examples: 125-129
 Savevalues: 1, 120-129
 SCAN block: 118
 Scan, overall GPSS/360: 97-104
 Current events chain: 98-103
 SEIZE block: 140
 Examples: 152, 153
 SELECT block: 39, 83
 Selection modes: 62-66
 SIMULATE card: 225
 Simulation run length: 69, 190
 SPACE card: 234
 SPLIT block: 87, 88
 Examples: 90, 91
 Standard numerical attributes: 8-14 (see also
 "numerical attributes")
 START card: 189, 190
 STARTMACRO card: 222
 Statistical output
 Blocks: 41
 Facilities: 151, 154
 Groups: 115
 Logic switches: 131
 Matrix savevalues: 125
 Queues: 172, 173, 174
 Savevalues: 125
 Storages: 160, 166
 Tables: 177, 183-185
 Transactions (chains): 56, 57
 User chains: 78

 Statistical problems in simulation: 203
 Status changes: 16, 102, 104, 105 (see also
 "delay chains")
 Status flag: (see "status changes")
 STORAGE definition card: 156
 Storages: 2, 7, 156-167
 Symbols
 Block: 43-52
 Entity: 216, 217
 SYN (synonymous) card: 226
 System numerical attributes: 10-13

 TABLE definition card: 179, 180
 Tables: (see "distribution tables")
 TABULATE block: 177, 179
 TERMINATE block: 69
 TEST block: 39, 84
 Examples: 85
 TEXT: 232
 TITLE card: 229
 TRACE block: 187, 188
 Transactions: 1, 6, 53-60 (see also "assembly sets",
 "chains", "parameters", "priority", "mark-
 time", "transit-time")
 Transaction printout: 55-58
 TRANSFER block: 62-66
 Examples: 63-66
 Transit time: (see also "MARK")
 Parameters: 53
 Transaction: 53

 Uniform distribution function: 34
 UNLINK block: 72
 Examples: 74-78
 UNLIST card: 189, 193
 UNTRACE block: 187, 188
 UPDATE card: 224
 Update feature: 224, 225
 User chains: 16, 60
 Utilization
 Facility: 151
 Storage: 160

 Variables: (see "arithmetic variable", "boolean
 variables", "floating-point variable")

 Warning messages: 213, 214
 WRITE block: 109, 110

 X card: 237

 Y card: 239

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)