This part of the paper describes GPSS II, *a general purpose digital systems simulation program based on a block diagram language.*

The program is a result of incorporating improvements dictated by extensive experience in the application of an earlier version. However, this article is self-contained.

Development and application of the language are illustrated by means of an example.

A general purpose digital simulator and examples of its application

Part I — Description of the simulator

by R. Efron and G. Gordon

Recent years have seen the development of several general purpose digital computer programs aimed at simplifying the task of carrying out simulation studies.[1] Among these is a program called the General Purpose Systems Simulator (GPSS).[2,3] Substantial experience accumulated in the use of this program has led to a number of suggested improvements and the present article describes a second version of the program, called the General Purpose Systems Simulator II (GPSS II).[4,5]

GPSS II is consistent with the original GPSS program in the sense that it employs the same principles and that all functions performed by the earlier program can also be performed by GPSS II. Although this paper does not assume prior knowledge of GPSS, before describing the new program some insight may be gained by giving a summary of the major improvements incorporated. In brief, these improvements are:

- Greater ability to sense the current state of the system, and to implement decisions based upon that state. "Variable statements" permit FORTRAN-like algebraic computations upon system variables. These capabilities provide much improved control over the flow of transactions in response to the current state of the system.
- The ability to associate a greater amount of information with each transaction in the form of eight parameters.

- The introduction of an indirect specification feature which permits a transaction to specify from its parameters the characteristics of the block being entered, rather than requiring these characteristics to be fixed for an entire simulation. This adds flexibility and makes it possible to reduce the size of certain types of models.
- The generalization of GPSS functions to permit a wider variety of arguments and a greater number of data points for inserting data descriptive of the system.
- An optional assembly feature which simplifies the description of the block diagram by furnishing the block numbers from symbolic names, and which enables the program to set up and call in "block macros". These macros are user-defined segments of a block diagram which can be used repetitively within a model, with the block characteristics varied as desired.
- The ability to go into FAP subroutines prepared by the user.
- Expanded output statistics and error information.
- Generally faster execution.

block diagram language

In GPSS II, the structure of the system being simulated is described in the form of a block diagram drawn with a fixed set of predefined *block types*. Each block type represents a specific action that is characteristic of some basic operation that can occur in a system. Connections between the blocks of the diagram indicate the sequence of actions that occur in the system. Where there is a choice of actions, more than one connection is made from a block to indicate the choice.

Moving through the system being simulated are certain basic units that depend upon the nature of the system. For example, a communication system is concerned with the movement of messages, a traffic system with vehicles or a data processing system with data records, and so on. These units are identified with entities called transactions. The sequence of actions occurring in the system in real time is reflected in the movement of transactions from block to block in simulated clock time.

action times

Clock time is represented by an integral number, with the interval of real time corresponding to a unit change of clock time chosen by the program user. The program computes an *action time* for each transaction entering a block to represent the time taken by the system action simulated by the block. The transaction remains at the block for this interval of simulated time before attempting to proceed. The action time may be a fixed interval (including zero), a random variable, or it can be made to depend upon conditions in the system in various ways. An action time is defined by giving a *mean* and *modifier* for the block. If the modifier is zero, the action time is a constant equal to the mean. If the modifier is a number ($\leq$ mean), the action time is a random variable chosen from the range, mean $\pm$ modifier, with equal probabilities given to each number in the range.

It is possible to introduce into the simulation, a number of

functions which are tables of numbers relating an input variable x to an output variable y. Any number (> 1) of pairs of values (x, y) can be used in a table defining a function. The table can be interpreted in the continuous mode by assuming linear variation between the points, thereby approximating any desired function with straight line segments. In the discrete mode, the table defines a step function.

By specifying the modifier at a block to be a function, the output of the function controls the block time. There are several modes of operating the functions, depending upon the choice of the input variable for the function. If a random number mode is selected, for example, the input is a random variable with uniform distribution between 0 and 1 so that the output is a random variable with a distribution controlled by the function. Other modes of operating functions are described later.

Associated with the system being simulated, there are certain physical or control elements that operate on the units represented by transactions and direct their flow through the system. Three types of system entities are defined in GPSS to represent such elements. Two, called *facilities* and *storages*, are characteristic of the physical equipment of the system. A third, called *logic switches*, is characteristic of the control elements of the system.

system entities

A facility is defined as any piece of equipment that can be engaged by a single transaction at a time. A storage is defined as any piece of equipment that can be occupied by many transactions at a time up to some predetermined limit. A logic switch is a two-level indicator that can be used to record the state of some system condition that is instrumental in deciding when or how operations are executed. A number of systems entities of each type may be employed, and they are identified by number. Some examples of how the systems entities might be interpreted are shown in Table 1.

These definitions of the system entities are descriptive and are not meant to determine literally the manner in which system elements are translated into simulation entities. Certain control decisions that depend specifically upon the state of equipment represented by facilities or storages can be made directly without the need to employ a logic switch. On the other hand, facilities and storages may be introduced in the simulation not to represent identifiable items of equipment but to control the flow of transaction by the restrictions implied in their definitions. For example, if a segment of a system can only be entered by a limited number of units, entry to the part of the block diagram representing this segment can be made contingent upon transactions being able to enter a storage with a capacity set to this limit.

Table 1

GPSS II *entity*	*System entity*
Facility	Card punch
Storage	Memory
Logic switch	Sense switch
Facility	Toll booth
Storage	Road
Logic switch	Traffic light

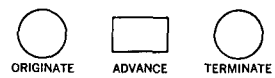
Similarly, the interpretation of transactions as representing physical units moving through the system should not be interpreted too literally. Transactions may be introduced to help control the system rather than to represent the basic units handled by the system. In the systems of Table 1, for example, the logic

switches might be controlled by transactions whose movements from block to block represent changes in system environment rather than the movement of some physical element.

choice of paths

Each block is given a number to identify it, and the connections between blocks are made by specifying at each block the number of the next block or blocks to which a transaction is to go. Provision is made for making a choice by specifying two next blocks, referred to as next blocks A and B. The method to be used for choosing between alternatives is indicated by a *selection factor* which can be set to indicate one of several modes. If there is no choice, the successor is next block A. A random choice can be made by setting the selection factor, S, to a decimal fraction. The probability of going to next block A is then 1 – S and to the next block B is S. A conditional mode, indicated by setting S = BOTH, sends a transaction to next block A if this move is possible, and to next block B if it is not. A random selection can also be made over any number of blocks by a selection mode called PICK. The choice can then be any block in the range of numbers A through B (B > A) with equal probability being given to each. Similarly, a mode called ALL makes a conditional choice over the range of blocks from numbers A through B by trying A first, A + 1 next, and so on.

Figure 1



creation, delay, and removal

Each block type is represented by a particular symbol and it is also given a name which is usually an imperative verb descriptive of the block action. Figure 1 shows the symbols and names of a number of the block types concerned with creating, delaying, and removing transactions. Transactions are created and entered into the simulation by a block type called ORIGINATE, with the action time controlling the interval between successive creations. The TERMINATE block removes the transactions from the simulation. An ADVANCE block is used to represent any action that takes time and therefore delays the transaction but does not involve any equipment. It is also used as a buffer in which to keep transactions while waiting for equipment to become available or for some condition in the system to change.

Figure 2

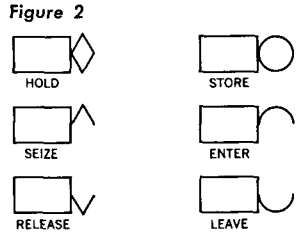


Figure 2 illustrates a number of blocks concerned with the use of equipment by transactions. The number of the item of equipment employed by the block is indicated in the flag attached to the symbol.

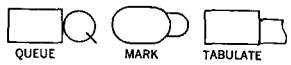
use of equipment

A HOLD block allows a transaction to engage a facility for as long as the transaction remains in the block. The SEIZE block similarly allows a transaction to engage a facility, but control of the facility is not given up by the transaction until some time later when it enters a RELEASE block. In a similar manner, the STORE block allows a transaction to occupy space in a storage while it is in the block, an ENTER block allows a transaction to occupy space only, and a LEAVE block allows space to be vacated.

gathering statistics

When the conditions for advancing a transaction are not satisfied, several transactions may be kept waiting at a block. Such transactions are kept in order by the program and allowed

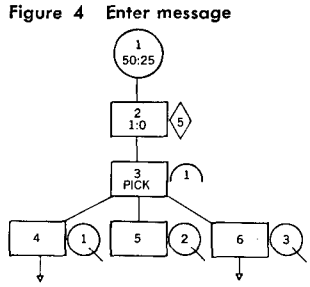
Figure 3



to move on by a first-in first-out rule. No information about the queue of transactions is gathered, however, unless the queue forms in a QUEUE block, shown in Figure 3, which is expressly designed to measure the average and maximum queue lengths and, if required, the distribution of time spent on the queue.

It is also desirable to measure the length of time taken by transactions to move through the system or parts of the system, and this can be done with the MARK and TABULATE blocks which are also shown in Figure 3. Each of these block types notes the time a transaction arrives at the block. Also, the TABULATE block enters in a table the difference between the times of arrival at the MARK and TABULATE blocks. In addition to working in conjunction with a MARK block to derive transit times of transactions, a TABULATE block may be used by itself to tabulate a wide variety of different quantities, as described later.

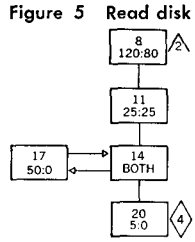
Figure 4 Enter message



data processing example

To illustrate the features of the program that have been described so far, consider the following example of simulating the flow of messages in a real-time data processing system. A computer receives messages from a terminal, locates corresponding records on a disk file, and uses these records to process the messages. It will be assumed that there are three disk files, each with an independently operated arm, but with all three files sharing a single channel for sending records into the computer. In this example, each message is a transaction, and the unit of time chosen is one millisecond. Action times, where used, are for simple rectangular distributions. In Figure 4, block 1 shows a mean of 50 and a modifier of 25 as indicated in the center of the block. The numbers appearing at the top of the blocks are the block numbers. At the bottom of some of the blocks, a selection factor is indicated.

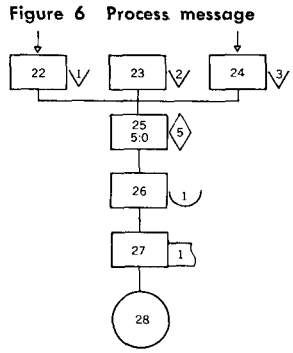
Figure 5 Read disk file



receiving messages

Figure 4 shows first the section of the block diagram concerned with entering messages into the system. Block number 1 is an ORIGINATE block that creates the transactions and sends them to a HOLD block using facility number 5 that represents the central processing unit of the computer. The time at this block, 1 millisecond, represents the time taken to read the message into the computer. A storage, number 1, is defined to represent the computer memory. The capacity of this storage is set to control the number of messages that can be held in the computer at one time. Messages occupy space by moving into an ENTER block. The ENTER block uses a random selection mode PICK to send transactions to one of three queues with equal probability. Here they wait for the availability of one of the disk files. Since the block diagram for each stream of transactions is identical from this point on, except that differently numbered disk files are used, only one stream is illustrated.

Figure 6 Process message



reading the disk file

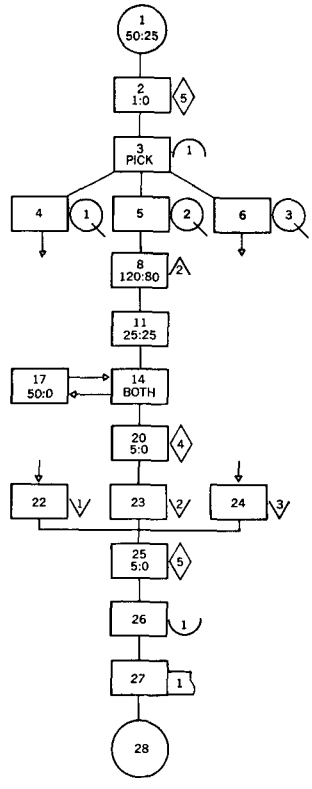
The disk file can only search for one record at a time. It is, therefore, defined as a facility. The transactions in the QUEUE block are waiting for the disk file to become available. When this happens, the transaction at the head of the queue moves into the

SEIZE block and so takes over control of the disk file (Figure 5). The time at the SEIZE block represents the time taken to position the arm over the correct track of the disk. When the transaction emerges from the SEIZE block, the arm is correctly positioned. It must now wait for the record to come under the head of the arm. The disk revolution time is assumed to be 50 milliseconds, so the waiting time can be anything from 0 to 50 units. The transaction is, therefore, sent to an ADVANCE block with a mean and modifier both equal to 25.

One more condition must be satisfied before the record can be read. The channel, which is being shared by all three disk files, must be available at the time the record comes under the head. To test for this condition, the transaction passes into an ADVANCE block with zero time and a selection factor of BOTH. The channel is represented by facility number 4, and if it is available, the transaction passes into a HOLD block to use the channel. If the channel is not free, the conditional selection mode at the ADVANCE block sends the transaction to another ADVANCE block where it waits for a period of 50 milliseconds and returns to try for the channel again when the record next comes under the head. If necessary, it keeps retrying in this manner until it does get onto the channel.

When the transaction has passed through the HOLD block representing the channel, the record has been read and the disk file is released by moving the transaction to a RELEASE block (Figure 6). The transaction moves into a HOLD block, using facility number 5 to represent the processing being carried out by the computer. When processing of the record at the HOLD block is complete, the transaction goes to a LEAVE block to give up the computer memory space and then goes to a TERMINATE block to be removed from the simulation. The complete block diagram is illustrated in Figure 7.

Figure 7 Simple real-time data processing system



As described so far, transactions have no particular identity. Each is treated by a block in the same manner as any other. In fact, transactions have two attributes—*priority* and *parameters*, which influence the way they are processed by blocks.

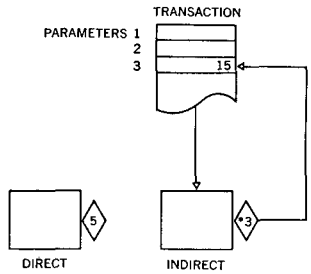
transaction properties

Each transaction has a priority assigned to it. There are eight priority levels, 0 being the lowest level which is the level set at the time of creating the transaction. At any point in the block diagram, the priority can be reset up or down by a PRIORITY block illustrated in Figure 8. Where there is competition between transactions to occupy a block or take over equipment, the service rule established by the program is to advance transactions in order of priority and first-in, first-out within a priority class.

Parameters are integral positive numbers that can be attached to a transaction. Up to eight parameters can be placed on any one transaction. They are placed there by an ASSIGN block (Figure 8) which can use as a source for the parameter any function or any of the system variables that are described later. An ASSIGN block can either add to, subtract from, or replace a parameter.

Figure 8

Figure 9 Indirect addressing



The meaning of parameters is determined by the program user and depends upon the use to be made of the parameter. The most important use of parameters is through a program feature, called *indirect addressing*, that is associated with blocks. When a block is defined normally (directly), a set of numbers must be given to determine such factors as the mean, modifier, equipment number, and the next blocks. With indirect addressing, one or more of these numbers can be left unspecified at the time of defining the block. Instead, the program can be instructed to take for this number a parameter that has been assigned to the transaction entering the block. In this way, the manner in which the block processes the transaction depends upon the transaction itself.

Figure 10 Data processing system—using indirect addressing

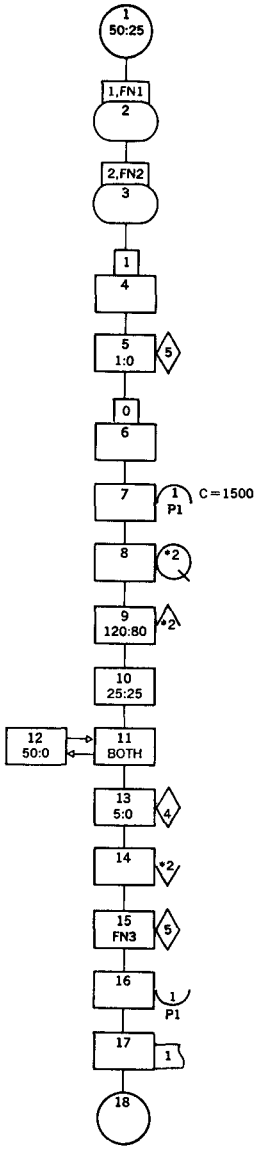


Figure 11 Listing of problem input

```
 LOC    NAME        X    Y    Z   SEL    NBA  NBB   MEAN   MOD  REMARKS
*
*                   MESSAGES FROM A TERMINAL ARE READ INTO A COMPUTER.
*                   A SEARCH IS MADE ON ONE OF THREE DISK FILES FOR A
*                   RECORD WHICH IS THEN PROCESSED. THE SYSTEM HAS
*                   THREE INDEPENDENT DISK FILES WHICH SHARE A COMMON
*                   CHANNEL. TIME UNIT IS 1 MILLISEC.
*
        ORIGINATE                                          50      25
        ASSIGN      1    FN1
        ASSIGN      2    FN2
        PRIORITY    1
        HOLD        5                                      1
        PRIORITY    0
        ENTER       1    P1
        QUEUE       *2
        SEIZE       *2                                     120     80
        ADVANCE                                            25      25
 TRY    ADVANCE                      BOTH  GO    WAIT
 WAIT   ADVANCE                            TRY             50
 GO     HOLD        4                                      5
        RELEASE     *2
        HOLD        5                                              FN3
        LEAVE       1    P1
        TABULATE    1
        TERMINATE   R                COUNT TO END OF SIMULATION
*
*       FUNCTION FOR ASSIGNING MESSAGE LENGTH. RANDOM MODE
*
 1      FUNCTION    RN1  C5
 0      5      .2   6    .4   9   .6     19    .8     24      1.0    25
*
*       FUNCTION FOR ASSIGNING DISK NUMBER. RANDOM MODE
*
 2      FUNCTION    RN1  D3
.333    1      .667 2    1.0  3
*
*       FUNCTION TO DETERMINE PROCESSING TIME. PARAMETER MODE
*
 3      FUNCTION    P1   C2
0       0      25   10
*
*       CARD TO DEFINE CAPACITY OF STORE
*
 1      CAPACITY    1500
*
*       CARD TO ESTABLISH TABULATION INTERVALS
*
 1      TABLE       M1   0    250  25     TABULATES TIME IN SYSTEM
*
*       CARD TO CONTROL LENGTH OF RUN
*
        START       1000              RUN UNTIL 1000 TRANS. TERMINATE
*
        END         PLACED AT END OF ALL PROBLEMS BEING RUN
```

indirect addressing

Indirect addressing is indicated by placing an asterisk followed by a parameter number in a field defining the block. For example, *3 in the equipment number field of a HOLD block makes that block use the value of parameter number 3 as the facility number. Such an indirectly defined HOLD block entered by a transaction with the number 15 in parameter 3 would act exactly as a HOLD block in which the facility was specified directly as being 15. This is illustrated in Figure 9. Indirect addressing increases the ability of the program to represent systems. It can also substantially decrease the size of the block diagram representing the system by making it unnecessary to duplicate segments of the diagram which are functionally equivalent but differ in specific values.

example of indirect addressing

As an example, Figure 10 shows the same system described before, but in this case making use of priority, parameters, and indirect addresses. The transactions leaving the ORIGINATE block are sent through two ASSIGN blocks. The first block places in parameter number 1 a number representing the message length which is derived from a continuous function, number 1, operating in a random number mode. The second ASSIGN block sets parameter number 2 to be a number 1, 2, or 3 derived from a discrete function, number 2, operating in a random number mode, which represents the disk file to be searched. The proportion of transactions sent to each of the disk files can be controlled with this function by choosing the values of x at which it is defined; in this case, the probabilities of going to the three disks are equal.

To illustrate the use of priority, the system is arranged to give priority for the use of the central processing unit to new messages arriving in the system. This is done by sending the transactions to a PRIORITY block that sets the level of priority to 1 just prior to the HOLD block representing the process of reading messages into the computer, and then resetting the priority to zero when the message has been read in. Now, with indirect addressing, it is no longer necessary to draw a separate segment of the block diagram for each of the transaction streams. Instead, a single QUEUE and a single SEIZE block are used with the queue number and the facility number specified indirectly by parameter number 2. The remainder of the block diagram is the same as before except that the RELEASE block that gives up the disk file is also indirectly addressed on parameter number 2.

further uses of parameters

Two other uses of parameters are illustrated in this example. Functions can operate in a *parameter mode* in which case they take as the input variable a parameter of the transaction calling for the function. At block number 15 of Figure 10, this feature is used to make the processing time depend upon the message length as indicated by parameter number 1. It is also possible to allow a parameter to control the amount of space the transaction occupies in a storage. This feature is used at block numbers 7 and 16 of Figure 10, to make the space occupied by the transaction equal to the number of characters in the message.

program input

Having drawn the complete block diagram, as shown in Figure 10, one card is punched for each block, and this set of cards, together with some control and definition cards, forms the input to the program. The cards are read by the program and used to set up the simulation model, execute the simulation for a specified length of time, and print out results in a single program run. Figure 11 shows a listing of the cards that need to be punched for the block diagram of Figure 10. There is an assembly option in the program that allows block numbers to be given symbolically and also assigns sequential block numbers automatically. This option has been employed in Figure 11. After assembly of the cards shown in Figure 11, the block numbers are assigned as shown in Figure 10.

Figure 12 Results of simulation run

CLOCK TIME REL 52758 ABS 52758

BLK	TRANS,TOTAL	BLK	TRANS,TOTAL	BLK	TRANS,TOTAL	BLK	TRANS,TOTAL	BLK	TRANS,TOTAL
1	0, 1074	2	0, 1074	3	0, 1074	4	0, 1074	5	0, 1074
6	0, 1074	7	0, 1074	8	71, 1074	9	2, 1003	10	1, 1001
11	0, 1071	12	0, 71	13	0, 1000	14	0, 1000	15	0, 1000
16	0, 1000	17	0, 1000	18	0, 1000				

FACILITY NR	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRANS	TRANS	$TRANS
1	.9769	336	153.39	65,S	0
2	.9616	333	152.35	32,S	0
3	.9987	334	157.75	36,S	0
4	.0948	1000	5.00	0	0
5	.1213	2074	3.08	0	0

STORAGE NR	CAPACITY	AVERAGE CONTENTS	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRANS	CURRENT CONTENTS
1	1500	606.06	.4040	15227	2099.87	984

QUEUE NR	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PER CENT ZEROS	AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
1	32	14.52	364	7	1.9	2105.22	0	28
2	14	3.76	343	17	5.0	577.94	0	10
3	37	21.19	367	1	.3	3045.78	0	33

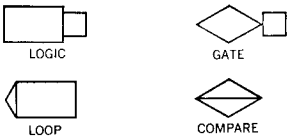
TABLE NUMBER 1

ENTRIES IN TABLE	MEAN ARGUMENT	STANDARD DEVIATION	
1000	2089.295	1555.665	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	−1.343
250	57	5.70	5.7	94.3	.120	−1.182
500	134	13.40	19.1	80.9	.239	−1.022
750	105	10.50	29.6	70.4	.359	−.861
1000	47	4.70	34.3	65.7	.479	−.700
1250	68	6.80	41.1	58.9	.598	−.540
1500	60	6.00	47.1	52.9	.718	−.379
1750	21	2.10	49.2	50.8	.838	−.218
2000	37	3.70	52.9	47.1	.957	−.057
2250	26	2.60	55.5	44.5	1.077	.103
2500	46	4.60	60.1	39.9	1.197	.264
2750	41	4.10	64.2	35.8	1.316	.425
3000	59	5.90	70.1	29.9	1.436	.585
3250	54	5.40	75.5	24.5	1.556	.746
3500	27	2.70	78.2	21.8	1.675	.907
3750	26	2.60	80.8	19.2	1.795	1.068
4000	41	4.10	84.9	15.1	1.915	1.228
4250	31	3.10	88.0	12.0	2.034	1.389
4500	50	5.00	93.0	7.0	2.154	1.550
4750	16	1.60	94.6	5.4	2.273	1.710
5000	18	1.80	96.4	3.6	2.393	1.871
5250	13	1.30	97.7	2.3	2.513	2.032
5500	5	.50	98.2	1.8	2.632	2.192
5750	7	.70	98.9	1.1	2.752	2.353
OVERFLOW	11	1.10	100.0	.0		

The assembly program also allows the user to define macros consisting of sets of blocks describing some segment of a system which is to be used repeatedly. Once defined, the macro is given a name and thereafter can be called by that name. Any number of the fields defining the blocks in the macro can be left unspecified at the time of definition—instead they are supplied at the time of calling the macro.

Figure 13



program output

The results of a run in which 1000 messages were processed are shown in Figure 12. Information is given about the number of times each block is entered; the utilization made of the facilities and storages; the size of the queues and the tabulation of the transit times through the system. In this case, because the transit time is measured from the time of creation, it is not necessary to use a MARK block.

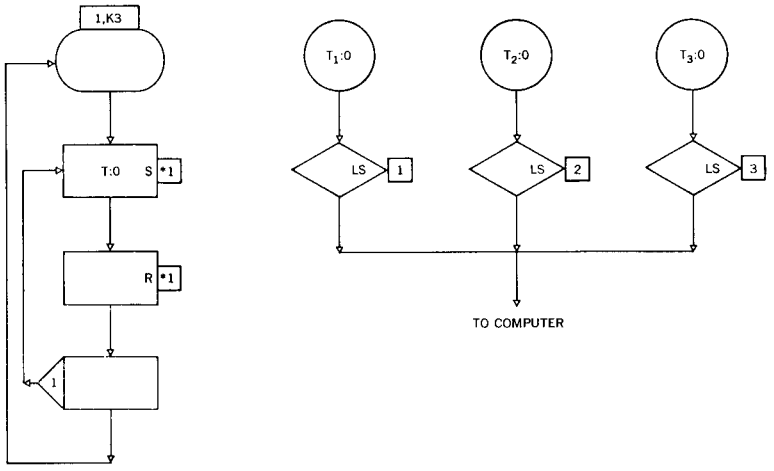
controlling transaction flow

Some of the block types, illustrated in Figure 13, are concerned with the use of logic switches and the control of transaction flow. One block type, called LOGIC, allows a transaction to either set, reset, or invert a logic switch. Another block type, called GATE, is able to test the status of a logic switch, facility, or storage. The program allows a transaction into this block only if the condition being tested is satisfied. Also shown in Figure 13 is a LOOP block which decrements a specified parameter and sends the transaction one way or the other, according to whether the result is zero or not (the COMPARE block is discussed later).

simple polling system

To show how these blocks are used in controlling the flow of transactions, suppose that in the previous example there are several terminals sending messages to the computer, and a simple polling system is set up whereby each terminal in turn gets access to the computer for a fixed length of time. If, for example, there are three terminals, each represented by an ORIGINATE block, the entry of messages into the system would be controlled as shown in Figure 14. One logic switch is associated with each terminal, and transactions leaving the ORIGINATE blocks are

Figure 14 Polling terminals with equal times



checked by a GATE block looking for the switch to be set. Transactions can enter the computer only when the switch is set.

In a small closed loop there is a single transaction cycling around that opens and closes each logic switch in turn. This is done by making parameter number 1 of this control transaction represent the number of the terminal to be checked. At the beginning of a cycle, this parameter is set to 3, and it is then used at a LOGIC block to set one of the switches by indirect addressing. The switch remains set for a fixed length of time T and is then reset when the control transaction moves to a second LOGIC block. The LOOP block decrements the parameter by 1 and sends the transaction to the next switch unless the parameter has been reduced to zero, in which case it restarts the cycle. In this way, the three logic switches are each opened, in turn, for a fixed interval T, thereby giving each terminal in sequence access to the computer.

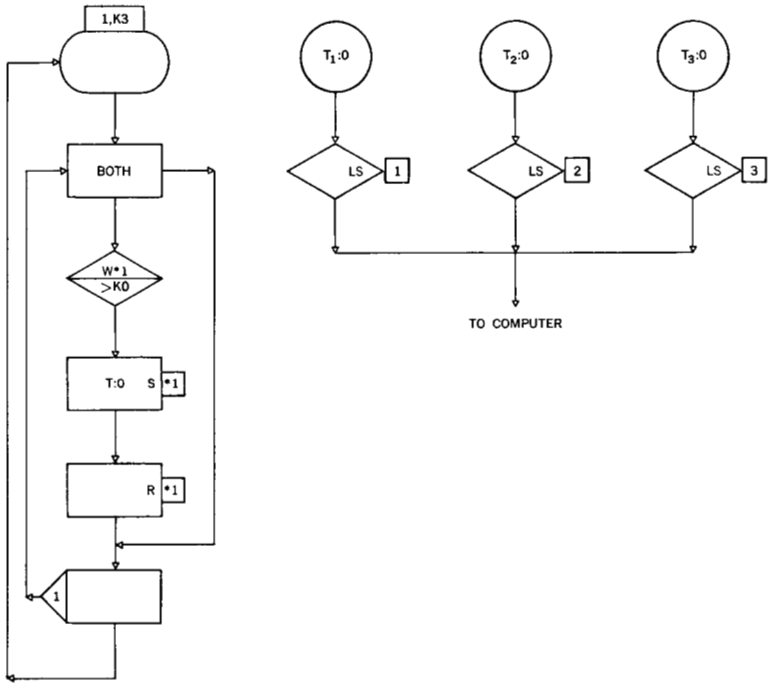
system variables

The control of transaction flow in the examples described so far has depended upon the state of facilities, storages, and logic switches. In addition, the program can make use of various items of data known collectively as *system variables*. These are numbers that describe the state of the system, and they can be referenced by the program through the use of symbols. For example, the number of transactions at block 20 is represented by W20; the contents of storage 15 by S15; the length of queue number 5 by Q5, and so on. In addition to system properties, reference to an absolute number, such as 6, can be made by using the symbols K 6. It is also possible to form combinations of system variables using simple mathematical operators to form what are called *variable statements*. For example, variable statement V1 might be defined as V1 = S 6 + Q 5/K 2. This provides a system variable V1 whose value is the contents of storage number 6, plus half the contents of queue number 5.

more advanced polling system

To illustrate how system variables are used, suppose in the previous example concerned with polling terminals that the system is arranged to skip a terminal if no messages are waiting at that terminal. The loop controlling the polling would then look as shown in Figure 15. A COMPARE block has been added to check whether a terminal is empty or not. This block type operates like the GATE block, but the condition tested can be a comparison between any two system variables. References to system variables can be made indirectly. Here the COMPARE block is used to check whether the number of transactions at the block indicated by parameter number 1 of the control transaction is zero. This parameter represents the number of the terminal to be checked next in the polling sequence. If this number is zero, there are no messages waiting at that terminal and the control transaction does not enter the COMPARE block. Instead, it is diverted directly to the LOOP block to step on to the next terminal. If there are messages waiting, the control transaction passes through the COMPARE block and switches a logic switch in the manner described before.

Figure 15 Polling terminals—skipping empty terminals



Systems variables greatly enhance the logical ability of the program. They also extend the ability to derive statistical data about the performance of the system, since any system variable can be tabulated by a TABULATE block to derive a statistical distribution or it can be printed out to give a chronological record. In addition, any system variable may be used as the input variable of a function. The random number, clock, storage, and parameter modes of operating functions are examples of system variables used for this purpose.

some other block types

Brief descriptions are given of some of the other block types. The INTERRUPT block (Figure 16) represents a higher-level use of the facility by a transaction. A transaction is admitted to the block only if the facility it is to interrupt is not already interrupted by another transaction. The facility remains interrupted until the transaction exists from the INTERRUPT block. If another transaction is using the facility at the first level of usage, that transaction is suspended in its progress through the block diagram until the interrupt concludes. The suspension is not unconditional, but the various special conditions can be ignored by the user, since the program maintains all the necessary records and automatically takes the proper action in every case. The PREEMPT block also represents a higher-level use of a facility by a transaction, but differs from the INTERRUPT block in that a separate block, the RETURN block, is used to signal the conclusion of the interrupt.

Figure 16

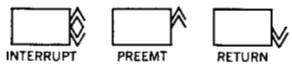
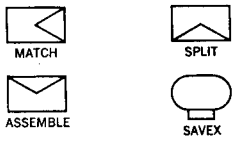


Figure 17



The SPLIT, MATCH, ASSEMBLE, and SAVEX blocks are shown in Figure 17. The SPLIT block creates a duplicate of each transaction that enters the block. The transactions thus created are said to be members of an *assembly set.* Further creation of transactions by splitting adds members to the set. Since the duplicate transaction may be synchronized with the original, the SPLIT block is useful in representing simultaneous events in a system. The MATCH blocks, used in pairs, synchronize the progress of two transactions of an assembly set. The transactions do not join, but continue to advance independently through the block diagram. An ASSEMBLE block joins a specified number of transactions from an assembly set into a single transaction. The final merging of independently manufactured parts is frequently represented by an ASSEMBLE block. The SAVEX block permits the user to gather and print information from the block diagram, and to transmit information from one transaction to another. Entry to a SAVEX block causes storage of the value of a specified system variable in certain memory locations—referred to as *savex* locations.

Not all the block types and features of the GPSS II program have been described, but it is hoped that enough information has been given to illustrate the principles of the program and its operation.

CITED REFERENCES

1. K. Blake and G. Gordon, "Systems Simulation With Digital Computers," *IBM Systems Journal* **3**, No. 1, 14 (1964).
2. G. Gordon, "A General Purpose Systems Simulator," *IBM Systems Journal* **1**, 18 (1962).
3. *General Purpose Systems Simulator,* Program Library, Reference 7090-CS-05X, International Business Machines Corporation.
4. *General Purpose Systems Simulator II, Reference Manual,* International Business Machines Corporation.
5. *General Purpose Systems Simulator II,* Program Library, Reference 7090-CS-13X, International Business Machines Corporation.