Unit-8: Simulation of Computer System

Why simulations?

Simulation is a very important means in the field of power electronics, mechanical engineering, mathematical formulations, and field of science and research etc.

Simulation leads to

- Saving of development time,
- Saving of costs ('burnt power circuits tend to be expensive'),
- Better understanding of the function,
- Testing and finding of critical states and regions of operation,
- Fast optimization of system and control.

Today it is difficult to imagine the task of power electronics development without the help of simulation.

Simulation Tools:

When someone is learning how to drive a car, the driving school teachers train them on a simulator before teaching them how to drive an actual car on the road. Knowing or testing a product or machine's behavior before manufacturing or using it is always good. You can know how the product will look and behave, and if it isn't working according to your expectations, you can always make changes. When the cost of improper operations is high, a mock-up of the actual control panel is connected to real-time simulator tools.

Simulation tools are used to understand and design equipment that will be close to the desired outcome. Simulation tools help in predicting the behavior of the system. You can utilize simulation tools to create and evaluate new designs, spot issues in the existing designs and test a system under certain conditions.

For example, electronic simulation tools are used to replicate the actual electronic device. Similarly mathematical simulation tools shows how mathematical equations works.

Properties of good simulation tools:

Some properties of a good tool are

- Comfortable
- Correct models for the elements (as simple as possible but, as good as necessary)
- Correct error messages
- Robust (*powerful/strong*) execution of the simulation.
- Sophisticated integration algorithm (various algorithms to be chosen for the type of model)
- Good output of the simulation results (formats which can be exported to other programs)
- Support from the manufacturer
- Portability of models from one program version to succeeding one

Different Simulation Tools:

Simulation tools are used in various industries, and they are highly beneficial. There are a number of powerful simulation tools available. All of them have advantages and disadvantages.

Following are popular simulation tools available:

Matlab / Simulink / SimPowerSystems:

Matlab is a mathematical tool that has been established for a long time. Toolboxes for various applications exist. One of them is Simulink, a graphical tool for the entering of functions. Simulink itself can be expanded with another toolbox: SimPowerSystem. This toolbox is designed for the simulation of electrical power systems including power electronics. The elements of the various toolboxes can be combined.

PSpice:

PSpice has been on the market for a long time. It started as a simulation tool for low-power electronic circuits. A large library with PSpice models for various electronic components exists. Further models can be added. Today it can simulate analog and

digital circuits with a lot of features. The representation of numerical blocks and controllers is difficult.

AnyLogic:

AnyLogic is the first simulation tool to introduce multimethod simulation modeling. The tool can convert flowcharts into interactive movies with 2D and 3D graphics. Users can present their models to their stakeholders in a self-explanatory and visually attractive manner. The tool provides an extensive set of graphical objects to visualize equipment, staff, vehicles, buildings, and many more items.

AutoDesk:

AutoDesk is a simulation tool that lets users test their designs efficiently to ensure they survive in the real world. Users can simulate their product digitally by reducing the cost of prototyping. The tool provides a feature named Fusion 360, a cloud simulation tool. It eliminates the need for any hardware.

Users can test for 8 different criteria, which include events, nonlinear and more. The tool lets users remove unnecessary bulk from the existing design. Users can achieve advanced design refinements by using the 3D mesh outputs as their guide.

Ansys:

Ansys, as a simulation tool, provides more time to optimize and innovate product performance. Users can create advanced physics models and analyze various fluid phenomena. The tool provides an intuitive and customizable space.

Users can accelerate their design cycle with the tool. The tool provides the best physics models and efficiently solves complex models. Users can accelerate their pre and post-processing as the tool provides a streamlined workflow for application and meshing setups.

Arena Simulation:

Arena Simulation provides discrete event simulation. It allows users to quickly analyze a system's behavior or process over time. Users can use the flowchart modeling methodology. It provides a vast library of pre-defined building blocks to model the process without custom programming. The tool lets users complete a range of statistical distribution options to accurately model process variability.

The tool has the ability to define routes and paths for simulation. It provides statistical reports and analysis. Users can see realistic 3D and 2D animations to visualize results beyond numbers.

Incontrol:

Incontrol is a simulation tool that helps users cope with costs, time, resources, reliability, productivity, sustainability, and safety. The tool provides two software- Supply Chain Simulation and Crowd Management Simulation software.

The supply chain simulation software is a comprehensive platform that offers users a fully scalable, easy-to-use, and configurable simulation environment. With data-driven answers, it helps users solve complex processes, people, infrastructure, and technology-related challenges. Users can virtually test and improve scenarios through the system lifecycle without disturbing the actual system.

The crowd management simulation is designed for the execution and creation of large crowds simulation in complex infrastructures. The users can use the tool to evaluate the safety and performance of the environment.

Introduction to Simulation Language:

A simulation language is a special purpose language structured to meet the programming requirements of the simulation models of a specific class of situations. The analyst develops the simulation model, employing this special purpose language, which is applicable over a class of applications.

In the sense, these language are comparable to FORTAN, C#, VB.NET, or Java but also include specific features to facilitate the modeling process.

For example, the simulation language **GPSS** is applicable to queuing like situations while SIMAN (**SIM**ulation **AN**alysis) and SLAM II are more appropriate for simulating the manufacturing and material handling systems. Some other modern simulation languages are GPSS/H, GPSS/PC, SLX and SIMSCRIPT III.

Features of Simulation Language:

The important features of simulation language can be identified as follows:

- Generation of large streams of random numbers
- Generation of random variates from a large number of probability distributions
- Determining the length of simulation run and length of wrapping (covering) up period
- Advancing the simulation clock
- Scanning the event list to determine the next earliest event to occur
- Collecting data
- Analyzing data and setting confidence intervals

Objectives of Simulation Language:

Simulation software packages are designed to meet the following objectives:

- To conveniently describe the elements, which commonly appear in simulation, such as the generation of random deviates.
- Flexibility of changing the design configuration of the system so as to consider alternate configuration.
- Internal timing and control mechanism, for book keeping of the vital information during the simulation run.
- To obtain conveniently, the data and statistics on the behavior of the system.

Advantage of Simulation Language:

- Since most of the features to be programmed are in-built, simulation languages take comparatively *less programming time and effort.*
- Since simulation language consists of blocks, specially constructed to simulate the common features, they *provide a natural framework for simulation modelling*.
- The simulation models coded in simulation language can *easily be changed and modified*.

- The *error detection and analysis* is done automatically in simulation languages.
- The simulation models developed in simulation languages, especially the specific application packages, called simulators, are very *easy to use*.

Simulation Language- GPSS:

GPSS, which stands for ***General Purpose Simulation System***. This language was developed primarily by Geoffrey Gordon at IBM around 1960, and has contributed important concepts to every commercial discrete event Computer Simulation Language developed ever since.

GPSS is a discrete time simulation general-purpose programming language, where a simulation clock advances in discrete steps. A system is modeled as transactions enter the system and are passed from one service (represented by blocs) to another. User translates his problem into a conceptual model, which is a block diagram. Then GPSS software package: Processes this block diagram, Executes the simulation run, and Produces statistics.

This is particularly well-suited for problems such as a factory. **GPSS** is less flexible than simulation languages such as Simulate and SIMSCRIPT but it is easier to use and more popular.

GPSS was designed especially for analysis who were not necessarily computer programmer. It is particularly suited for modeling traffic and queuing systems. A **GPSS** programmer does not write a program in the same sense as SIMSCRIPT programmer does instead, he construct a block diagram – a network of interconnected blocks, each performing a special simulation oriented function.

GPSS provides a set of 48 different blocks to choose from each of which can be used repeatedly. Each block has a name and specific task to perform. Moving through the system of one block to another block are called transaction and entities are customer, messenger, machine parts, vehicle, etc.

Basic Structure

A transaction is a GPSS object with a number of attributes. A transaction is like a customer entering into the process for service.

A single transaction may represent several individual entities. Each transaction has to be generated either one at a time or in batches. Once the transactions are appear into the system, they must be contained exactly in one action Block. However, a Block may contain many transactions.

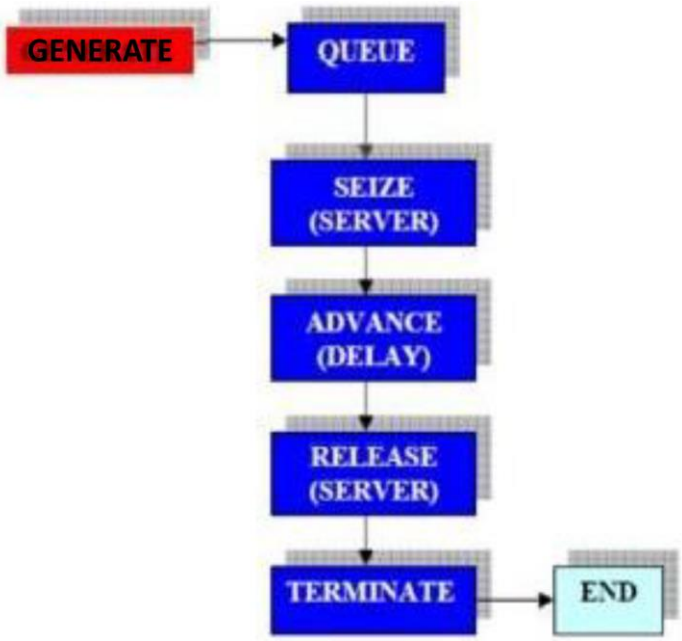


Fig: Basic Structure GPSS

- A **GENERATE** Block ***generates a stream of transactions with a specific set of behavior***. No transaction may again enter this block. Behavior could be deterministic, stochastic, functional, etc. A Transaction leaving a GENERATE Block descends into the next available Block it finds. The entering Block shouldn't deny entries to transactions. Otherwise, system backups may result.
- A **QUEUE** Block never ***refuses any transaction***. If a transaction cannot enter into the next Block, it stays at the current Block. Therefore, a QUEUE simulates an infinitely long buffer.
- A transaction attempts to **SEIZE** ***a facility (server, router, CPU) for service.*** If it succeeds, it would leave the current Block and start using the facility. If

not, it stays where it is until the next time. As long as a facility is occupied, it cannot allow another transaction to SEIZE it.

- An **ADVANCE** Block captures the transaction and imposes a ***delay on it wherever it is***. The delay could be deterministic, probabilistic, etc.
- A **RELEASE** Block forces ***a transaction to release its facility***. For every successful SEIZE, there must be a RELEASE.
- A **TERMINATE** Block ***kills the entering transaction*** here.

TABULATE, tabulates the time in it took the transaction to reach that point from the time it entered the simulation system. GPSS handles the advancement of time by a block called ADVANCE. When a transaction enters the block an action time is computed and added to the current time to produce a block departure time. When the time reaches the departure time the transaction will be moved, if possible, to the next block in the chart. Transaction might process certain attributes which are used to make logical decision within block.

GPSS - Basic Commands

- **START:** set the termination count and begin a simulation
- **EXIT:** end the GPSS world session
- **CLEAR:** reset statistics and remove transactions
- **CONTINUE:** resume the simulation
- **HALT:** stop the simulation and delete all queued commands
- **INCLUDE:** read and translate a secondary model file
- **INTEGRATE:** automatically integrate a time differential in a use variable
- **REPORT:** set the name of the report file or request an immediate report
- **RESET:** reset the statistics of the simulation
- **SHOW:** evaluate and display expression
- **STEP:** attempt a limited number of block entities
- **STOP:** set a stop condition based on block entry attempts
- **STORAGE:** define a storage entity
- **TABLE:** define a table entity
- **VARIABLE:** define a variable entity

Transaction:

Transaction in GPSS is a process that represents the real-world system you are modeling which is executed by moving from block to block. Each transaction in the model is contained in exactly one block, but one block may contain many transactions.

The Various Blocks of a GPSS are as follows:

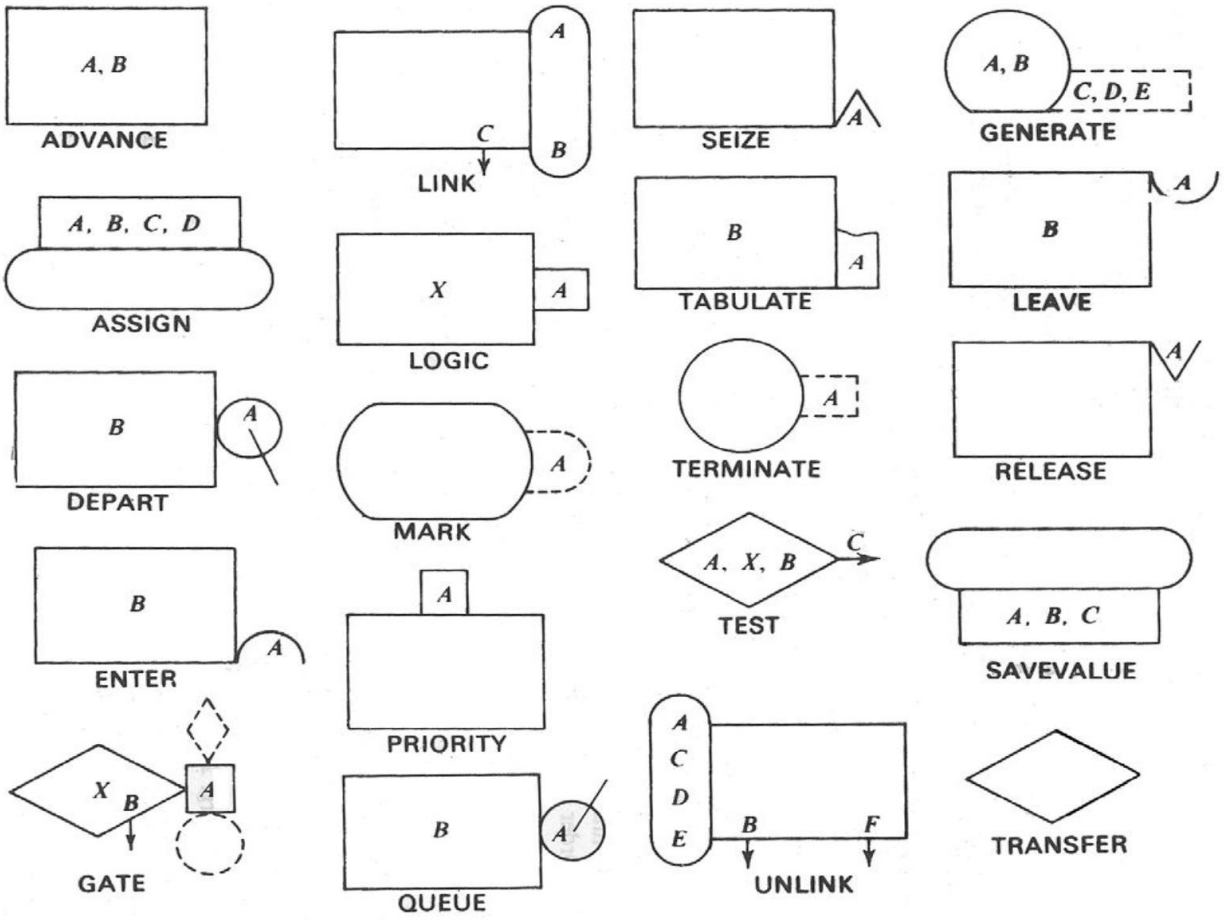
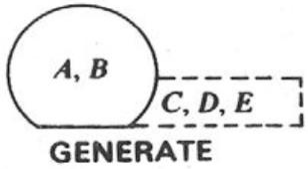


Fig: Different Blocks used in GPSS

1. GENERATE BLOCK:

This block will produce or create a transactions for future entry into the simulation. It create with inter-arrival times determined by the attribute values. The Label is optional. The distribution of inter-arrival times follows a uniform probability distribution.



A - Average value of uniform distribution (mean)

B - Inter generation time half-range or Function Modifier

C - Start delay before first transaction is generated

D – Maximum number of transaction generated

E - Priority allocated to transaction. Zero is the default

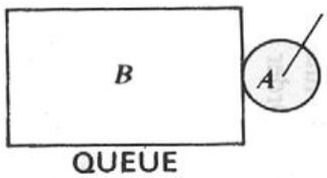
Syntax:

Line Number Label GENERATE A, B, C, D, E (*Line Number and Label Optional)*

Example: GENERATE 5, 2

2. QUEUE BLOCK:

This block will instruct GPSS to start gathering queuing statistics on the queue named in its attribute value. The Label is optional but may be necessary if you have to refer to this line from somewhere else in the program.



Syntax:

Line Number Label QUEUE A

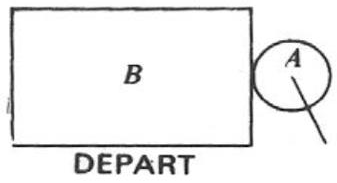
A = Name of queue (For Example: Garage)

Example: QUEUE Garage

If a transaction arriving at the queue block cannot proceed because it is blocked by the next stage, then it will stay in the queue block until it can gain entry to the next stage.

3. DEPART BLOCK:

This block instructs GPSS that a transaction is leaving the queue named in its attribute value. This is necessary in order to compile the statistics on the queue. The Label is optional.



Syntax:

Line Number Label DEPART A

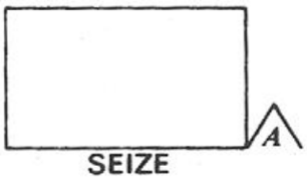
A = name of the queue (For Example: Garage)

Example: DEPART Garage

Note: *If **QUEUE** block is used in the model then the corresponding **DEPART** block should follow it.*

4. SEIZE BLOCK:

This block allows the transaction to seize (*acquire*) a **Facility** if it is free. Thus it may be a car "seizing" a "facility" such as a petrol pump or a customer in a supermarket "seizing" a "facility" such as the checkout assistant. When the car or customer is being serviced by the facility, then it is said to "own the facility". The Label is optional.



Syntax:

Line Number Label SEIZE A

A = Name of ***Facility*** *(For Example: Pump)*

Example: SEIZE PUMP

Note: *A transaction can only seize a facility if it is free or else wait until the owning transaction releases it.*

5. RELEASE BLOCK:

A transaction entering this block informs GPSS that it is giving up ownership of the **Facility** named in its attribute value. The Label is optional.

Syntax:

Line Number Label RELEASE A

A = Name of Facility (For Example: Pump)

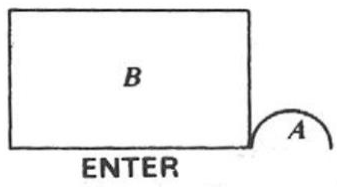
Note: *By giving up ownership of the facility, the transaction makes it available for another transaction that may be waiting to use it. If* ***SEIZE*** *block is used in the model then the corresponding* ***RELEASE*** *block should follow* it.

6. ENTER BLOCK:

This Block instructs GPSS that a transaction has entered STORAGE.

- The first attribute value gives the name of storage (Storage Contains Multiple Service Points)
- The second attribute value gives the amount the storage will be incremented by.

When the transaction enters the ENTER block. STORAGE must be declared at the beginning of a program.



Syntax:

Line Number Label ENTER A, B

A = name of the storage (for Example: warehouse)

B = increment storage by this value

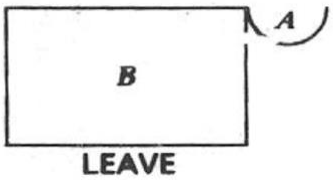
Example:

ENTER Warehouse, 25

In this example the Active Transaction demands 25 storage units from the storage units available at the Storage Entity named Warehouse. If there are not enough storage units remaining in the Storage Entity, the Transaction comes to rest on the Delay Chain of the Storage Entity.

7. LEAVE BLOCK:

This block instructs GPSS that a transaction is leaving a STORAGE. The first attribute gives the name of the STORAGE and the second attribute decrements the storage by the value of the attribute.



Syntax:

Line Number Label LEAVE A, B

A = name of the storage (for Example: warehouse)

B = Decrement storage by the value

Note: *If* ***ENTER*** *block is used in the model then the corresponding* ***LEAVE*** *block should follow it.*

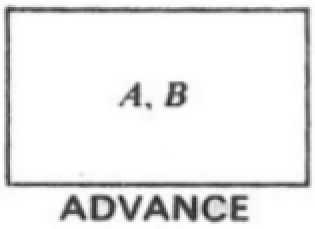
Example:

LEAVE Warehouse, 25

In this example, when a Transaction enters the LEAVE Block, the available storage units at the Storage Entity named Warehouse is decrement by 25 and total storage is increment by 25 units.

8. ADVANCE BLOCK:

This block represents the servicing of a transaction. The servicing times follow a uniform probability distribution. The Label is optional.



Syntax:

Line Number Label ADVANCE A, B

A = average value of uniform distribution (mean)

B = Modifier

A transaction entering this block will be delayed by a time interval chosen at random from the specified probability distribution.

9. TRANSFER BLOCK:

This block will take transactions entering it and transfer them to each of two different destinations according to laid down proportions.

Syntax:

Line Number Label TRANSFER A, B, C

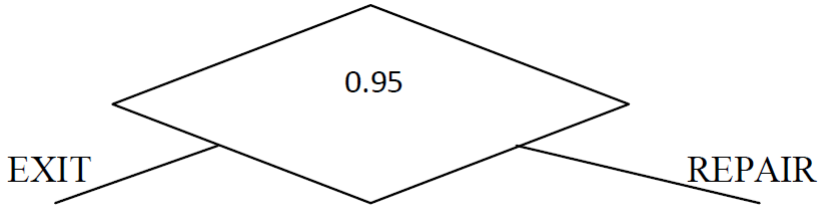
A = probability value (0 to 1)

B = proportion of (1-A) transactions transferred to this Labelled location

C = proportion A transactions transferred to this Labelled location

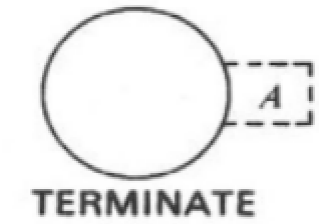
For Example: In this case 95% of all transactions entering the TRANSFER block will go to the program Line Labelled REPAIR and 5% will go to the program Line Labelled EXIT. If the second attribute "EXIT" is replaced by a "comma", then the 5% will go to the next block in the program.

TRANSFER 0.95, EXIT, REPAIR



10. TERMINATE BOLOCK:

This block destroys any transaction entering it and removes it from computer memory. Each time a transaction enters this block it decrements a counter by an amount equal to its attribute value. The counter is set by the user upon starting the simulation.



Syntax:

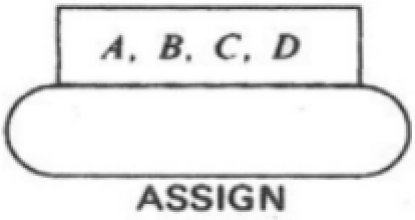
Line Number Label TERMINATE A

A = decrements simulation counter by this amount

When the counter, set at the beginning of the simulation, reaches zero then the simulation is complete and a statistical report is produced on the outcome of the simulation

11. ASIGN BLOCK:

ASSIGN Blocks are used to place or modify a value in a Transaction Parameter.



Syntax:

ASSIGN A, B, C

A - Parameter number of the Active Transaction. Required.

B - Value. Required.

C - Function number. Optional.

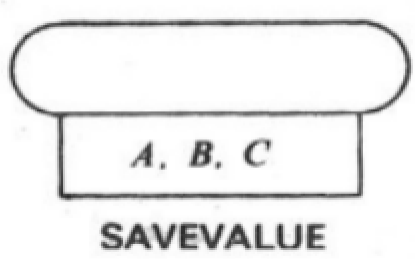
Examples:

ASSIGN 2000, 150.6

This is the simplest way to use the ASSIGN Block. The value 150.6 is assigned to Parameter number 2000 of the entering Transaction. If no such Parameter exists, it is created.

12. SAVEVALUE BLOCK:

A SAVEVALUE Block changes the value of a Savevalue Entity.



Syntax:

SAVEVALUE A, B

A - Savevalue Entity number. Required. May be followed by + or - to indicate addition or subtraction to existing value.

B - The value to be stored, added, or subtracted. Required.

Example 1:

SAVEVALUE Account, 99.95

In this example, the Savevalue Entity named Account takes on the value 99.95.

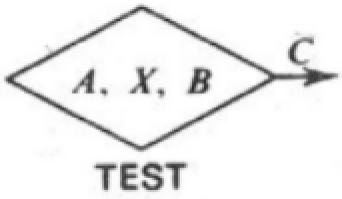
Example 2:

SAVEVALUE Kantipur, "The old name of Kathmandu"

In this example, the Savevalue Entity named Kantipur is assigned a string "The old name of Kathmandu". If the Savevalue Entity does not exist, it is created.

13. TEST BLOCK:

A TEST Block compares values, normally SNAs, and controls the destination of the Active Transaction based on the result of the comparison.



Syntax:

TEST O A, B, C

O - Relational operator. Relationship of Operand A to Operand B for a successful test. Required. The operator must be E, G, GE, L, LE, or NE (E=Equal, G=Greater than, GE=Greater than or equal, L=Less than, LE=Less than or equal, NE=Not equal)

A - Test value. Required.

B - Reference value. Required.

C - Destination Block number. Optional.

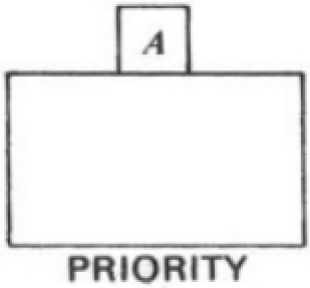
Example:

TEST G C1, 70000

In this example of a "Refuse Mode" TEST Block, the Active Transaction enters the TEST Block if the relative system clock value is greater than 70000. Otherwise, the Transaction is blocked until the test is true.

14. PRIORITY BLOCK:

A PRIORITY Block sets the priority of the Active Transaction.



Syntax:

PRIORITY A, B

A - New priority value. Required.

B - Buffer option. Places Active Transaction behind priority peers on CEC. Optional.

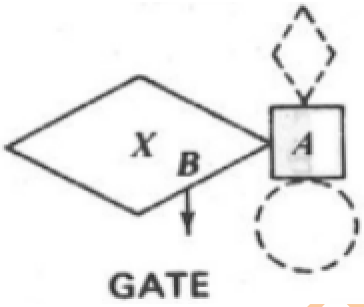
Example:

PRIORITY 10

In this example any entering Transaction is made to have a priority of 10.

15. GATE BLOCK:

A GATE Block alters Transaction flow based on the state of an entity.



Syntax:

GATE O A, B

O - Conditional operator. Condition required of entity to be tested for successful test. Required. The operator must be LS, LR, U, NU, SF, SNF, SE, SNE etc. (LS=Logic Switch Set, LR=Logic Switch Reset, U=Facility in Use, NU=Facility in Not Use, SF=Storage Full, SNF=Storage Not Full, SE=Storage Empty, SNE=Storage Not Empty)

A - Entity name or number to be tested. The entity type is implied by the conditional operator. Required.

B - Destination Block number when test is unsuccessful. Optional.

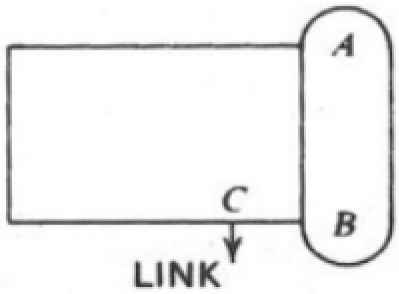
Example:

GATE SNF MotorPool

In this example of a "Refuse Mode" GATE Block, the Active Transaction enters the GATE Block if the Storage Entity named MotorPool is not full (i.e. if at least 1 unit of storage is available). If the Storage is full, the Active Transaction is blocked until 1 or more storage units become available.

16. LINK BLOCK:

A LINK Block controls the placement of the Active Transaction on the User Chain of a Userchain Entity.



Syntax:

LINK A, B, C

A - Userchain number. The Userchain Entity which may receive the entering Transaction. Required.

B - Chain ordering. The placement of new Transactions on the Userchain. Required.

C - Next Block location. The destination Block for Transactions which find the Link Indicator of the Userchain in the off state (reset). Optional.

Example:

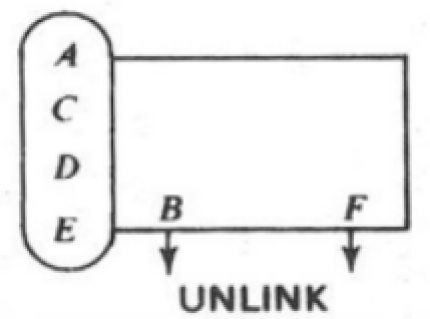
LINK OnHold, FIFO

In this example, the Active Transaction is placed at the end of the User Chain Entity named OnHold. It is removed from all chains except Transaction Groups and Interrupt Chains. In other words, preemptions are not cleared. The Transaction remains on the User Chain until

some other Transaction enters an UNLINK Block and specifically removes it. In the present example, the Transaction is placed at the end of the User Chain named OnHold.

17. UNLINK BLOCK:

An UNLINK Block removes Transactions from the User Chain of a Userchain Entity.



Syntax:

UNLINK O A, B, C, D, E, F

O - Relational operator. Relationship of D to E for removal to occur. Optional. The operator must be Null, E, G, GE, L, LE or NE.

A - User Chain number. User Chain from which one or more Transactions will be removed. Required.

B - Block number. The destination Block for removed Transactions. Required.

C - Removal limit. The maximum number of Transactions to be removed. If not specified, ALL is used. Optional.

D - Test value. The member Transaction Parameter name or number to be tested, a Boolean variable to be tested, or BACK to remove from the tail of the chain. Optional.

E - Reference value. The value against which the D Operand is compared. Optional.

F - Block number. The alternate destination for the entering Transaction. Optional.

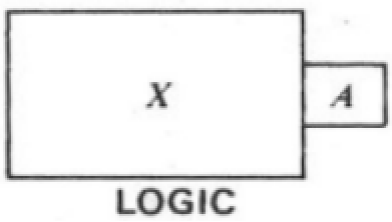
Example:

UNLINK OnHold, Reentry, 1

This is the simplest way to use the UNLINK Block. The first Transaction at the head of the Userchain Entity named OnHold, if any, is taken off the chain and is directed to the Block labeled Reentry. It is put on the CEC behind Transactions of the same priority. The Transaction entering the UNLINK Block proceeds to the Next Sequential Block.

18. LOGIC BLOCK:

A LOGIC Block changes the state of a Logicswitch entity.



Syntax:

LOGIC O A

O - Logic operator. Required. The operator must be S, R, or I (S=Set, R=Reset, I=Inverse)

A - Logicswitch Entity number. Required.

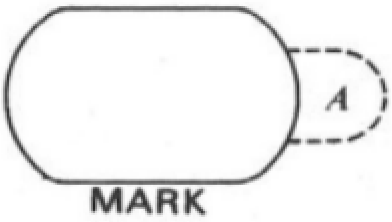
Example:

LOGIC S PowerSwitch

In this example, the Logicswitch Entity named PowerSwitch is left in the true or "set" state.

19. MARK

It marks the absolute clock time that the Transaction first entered the simulation or entered a MARK Block.



Syntax:

MARK (Param No)

Example:

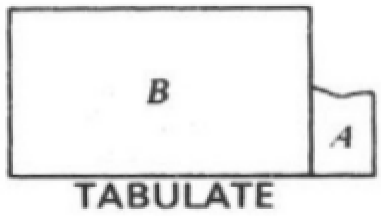
MARK Beginning

In this example, when a Transaction enters the MARK Block, its Transaction Parameter named Beginning is given a value equal to the value of the absolute system clock, AC1.

20. TABULATE:

A Table Entity is a set of integers used to accumulate data for a histogram. Each integer represents a frequency class in a histogram. A Table Entity is defined by a TABLE Command.

TABULATE Blocks update the histogram data accumulated in a Table Entity.



Syntax:

TABULATE A, B

A - Table Entity name or number. Required.

B - Weighting factor. Optional.

Example:

TABULATE Sales

When a Transaction enters this TABULATE Block, the Table Entity named Sales is found. Sales must have been defined in a TABLE Command. Then the statistics associated with the table are updated with no weighting.

Standard Numerical Attributes (SNAs):

SNAs Contains

- One or two letter code
- A number

For example:

'Storage number 5' is denoted by S5

'Length of queue number 15' is denoted by Q15

SNA's combined with operators can form variable statements

myVariable VARIABLE S6+5*(Q12+Q17)

Similarly, to define floating point, FVARIABLE is used.

Value of SNA's are calculated at the time they are required. The value is never maintained for future use. To save value of SNA, SAVEVALUE is used.

Following table shows the list of some useful SNAs

C1 - Current value of clock

CHn - Number of transactions on chain 'n'

Fn - Current status of facility number 'n'. 1- if facility is busy, 0 otherwise

FNn - Value of function 'n'.

Kn - Integer 'n'

M1 - Transit time of a transaction

Nn - Total number of transactions that have entered block 'n'

Pxn - Parameter number 'n' of transaction, of type 'x'

Qn - Length of queue 'n'

Rn - Space remaining in storage 'n'

RNn - A computed random number having one of the values 1 to 999 with equal probability. 'n' = 1, 2, . . ., 8

Sn - Current occupancy of storage 'n'

Vn - Value of variable statement number 'n'

Wn - Number of transactions currently at block 'n'

Xxn - Value of savevalue location 'n' of type 'x'

Entities:

Transaction entities: GENERATE, SPLIT, TRANSFER, TERMINATE

Facilities entities: SEIZE, RELEASE

Queue entities: QUEUE, DEPART

Storage entities: ENTER, LEAVE

Format of GPSS Program

In GPSS each separate Line of the program will either be a statement or a block. (In a few rare situations the GPSS statement will be continued for two or more Lines). A general format of a GPSS block consists of four separate items. These are:

1. Label or location
2. Operation or block statement
3. Operands
4. Comments

Example 1:

A manufacturing shop is turning out parts at rate of one every 5 minutes. As they are finished, the parts go to an inspector, who takes 4±3 minutes to examine each one and rejects about 10% of the parts.

Model 1: Without inspection or wait

```
          GENERATE    5, 0
          ADVANCE     4, 3
          TRANSFER    0.1 ACC REJ
ACC       TERMINATE   1
REJ       TERMINATE   1
          START 100
```

Model 2: With 1 inspection

```
          GENERATE    5, 0
          SEIZE       1
          ADVANCE     4, 3
          RELEGE      1
          TRANSFER    0.1 ACC REJ
ACC       TERMINATE   1
REJ       TERMINATE   1
          START 100
```

Model 2: With 3 inspection

```
            GENERATE    5, 0
            ENTER       1
            ADVANCE     12, 9
            LEAVE       1
            TRANSFER    0.1 ACC REJ
ACC         TERMINATE   1
REJ         TERMINATE   1
            START 100
```

Controlling the Length of a Simulation Run (Time Driven):

The TERMINATE BLOCK can contain an attribute value that will make the simulation stop when the Number of transactions entering the TERMINATE block equals the attribute value.

In a GENERATE statement it is possible to specify the maximum Number of transactions that the GENERATE block will produce. Clearly the model will cease to function once this limit has been reached and the last transaction has been terminated.

It is possible to have a separate timing section in your program that will determine the exact time of a simulation.

For Example:

```
TIMER GENERATE 2000
TERMINATE 1
```

This will cause the GENERATE block to produce a single transaction at 2000 time units after the simulation starts and decrement the termination counter to zero. This

will immediately stop the simulation run. You must ensure there is no attribute value in any other TERMINATE block in the simulation model.

Example 1: Joe's Barbershop (Time Driven)

Suppose we want to simulate a typical day of nine hours' work. Since we have decided that one time unit is equal to one minute then we must convert nine hours in to minutes i.e. 9x60 = 540 minutes.

```
MEN     GENERATE    18, 6
        QUEUE       SEAT
        SEIZE       JOE
        DEPART      SEAT
        ADVANCE     15, 3
        RELEASE     JOE
        TERMINATE
TIMER   GENERATE    540
        TERMINATE   1
        START       1
```

Note: the first TERMINATE block does not have an attribute.

In the program above, after the command START 1 is issued, the simulation begins to run. Due to the fact that the first TERMINATE block does not have an attribute, the termination counter is not decremented and the simulation continues. At 540 time units after the simulation starts, however, a single transaction will be generated by generator which will decrement the termination counter to zero and stop the program.

Example 2: Joe's Barbershop (Event Driven)

Suppose we want to simulate the Barbershop for 50 Transactions.

```
MEN     GENERATE    18, 6
        QUEUE       SEAT
        SEIZE       JOE
        DEPART      SEAT
        ADVANCE     15, 3
        RELEASE     JOE
        TERMINATE   1
        START       50
```

End of Unit-8