MANUAL
FOR A GENERAL PURPOSE SIMULATOR
USED TO EVALUATE RELIABILITY
OF DIGITAL SYSTEMS

Peter Alan Thompson

Technical Report No. 132

August 1977

This work was supported in part by NASA Grant NGR-05-020-699, Sup. 1.

Acknowledgment: Acknowledgment is made to the NASA Ames Research Center, Moffett Field, California for the use of their CDC-7600 computing facility.
MANUAL

FOR A GENERAL PURPOSE SIMULATOR
USED TO EVALUATE RELIABILITY
OF DIGITAL SYSTEMS

Peter Alan Thompson

Technical Report No. 132

August 1977

DIGITAL SYSTEMS LABORATORY
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California

This work was supported in part by NASA Grant NGR-05-020-699, Sup. 1.

Acknowledgment: Acknowledgment is made to the NASA Ames Research Center,
Moffett Field, California for the use of their CDC-7600 computing facility.
ABSTRACT

A simulation technique has been developed for the reliability evaluation of arbitrarily defined computer systems. The main simulation program is written in FORTRAN IV, and requires no changes to simulate many different systems. The user defines a model for a particular system by supplying a set of short FORTRAN subroutines, and a specially formatted block of numerical parameters. The subroutines specify the functional behavior of various subsystems comprising the model, while the numerical parameters describe how the subsystems are interconnected, their time delays, what faults occur in each one, etc. The main simulation program uses this model to perform a Monte-Carlo-type evaluation of the systems' reliability.

This report supplements a basic description of the technique by
supplying all the details necessary for writing subroutines, specifying numerical parameters, and using the main simulation program. The simulation is event-driven, and automatically generates pseudo-random faults and time delays according to parameters given by the user. Some problems typical of event simulators, such as ambiguities arising from random time-delay generation, can be solved by taking advantage of special facilities built into the simulation package. A complete source listing of the main program is included for reference.

INDEX TERMS: Simulation, Reliability, Monte-Carlo.
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title Page</td>
<td>i</td>
</tr>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Contents</td>
<td>iv</td>
</tr>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Unit Type and Data-Store Subroutines</td>
</tr>
<tr>
<td>3</td>
<td>System Parameter Data-Deck</td>
</tr>
<tr>
<td>3.1</td>
<td>Segment Descriptions</td>
</tr>
<tr>
<td>3.2</td>
<td>Card Type Descriptions</td>
</tr>
<tr>
<td>4</td>
<td>Flow of Control During Simulation</td>
</tr>
<tr>
<td>5</td>
<td>Generation of Randomly Distributed Faults and Time Delays</td>
</tr>
<tr>
<td>6</td>
<td>Resolving Time Delay Ambiguities</td>
</tr>
<tr>
<td>7</td>
<td>Appendix A: FORTRAN Source Listing of Simulation Package</td>
</tr>
<tr>
<td>References</td>
<td>39</td>
</tr>
</tbody>
</table>
1. Introduction

A simulator technique has been developed which can be used to evaluate the reliability of digital systems. To study a particular system, the user must specify it as a collection of subsystems, called units, which transfer data between one another on a network of links. For any one type of unit, the user must supply a FORTRAN subroutine which observes that unit's input-link data, and computes its output-link data. The subroutine must consider the effects of various fault conditions which might occur within the physical subsystem which is represented by that unit. In addition to the unit/link configuration and subroutines, the user also specifies probability density functions for faults and time-delays, initial values for state variables within each unit, and various other simulation parameters. All of these specifications together constitute a complete model of the system to be simulated.

The user-supplied FORTRAN subroutines are appended to the main simulation program (which is the same for all models), and are called by the main program during the simulation process. The unit/link configuration and all other numerical parameters are punched in a "data deck" using special pre-defined formats. The main program reads in the data-deck once, then repeatedly simulates the system from some initial simulated time up to the time at which the system reaches a "failed" state. The simulator is event-driven, with randomly occurring "failures" being generated within units automatically by the main program, according to the probability density functions.
specified by the user. After the model is simulated-to-failure many times, a Monte-Carlo analysis is done to obtain a reliability vs. time curve for the system.

A separate report [Thompson, 1976] describes this process in more detail, and gives several examples of its application to different kinds of systems. Example a, Section 4.1, of that report is occasionally referred to here.

This report provides the details needed to define a model for the simulator program, including both the FORTRAN subroutines and the parameter data-deck. Specifics of the simulation process relating to the design of a model are also discussed.

Appendix A of this report is a listing of the main simulation program. The various sections of code have the following functions:

- MAIN - main simulation program.
- SETUP - reads data deck.
- EDUMP - prints event list.
- ADDU - adds a unit number to affected-unit list.
- INSERT - inserts an event into next-event-list.
- FLTGEN - generates a new fault change event.
- FTIME - generates random variable from given distribution.
- URAND - generates uniform \((0,1)\) random value.
- MSTOP - generates a stop event for the current mission.
- RSTOP - halts simulation after the current mission.
2. Unit Type and Data-Store Subroutines

The user must supply one subroutine for each type of unit in the model. In the current implementation, the type subroutines are written in standard ANSI FORTRAN IV, non-extended. The simulator main program passes all essential information to these FORTRAN subroutines through arrays and variables which have been pre-defined in a labeled COMMON area. By including the standard COMMON statements at the beginning of each subroutine, the user can directly access the values at the input ports, the fault state of that unit, the state variables, and other parameters.

The name of each subroutine is standardized so that the main program may call the subroutine corresponding to a specific unit type. Each type of unit is assigned a number from 1 to 30, and the user must name the subroutine 'TYPEnn,' where 'nn' is that type number. The subroutine names for types 2, 8, and 25 are thus TYPE:!, TYPE8, and TYPE25, respectively. The subroutine declarations have no arguments.

The COMMON statements are shown below, with detailed explanations for the use of some variables. All upper bounds for array indices are shown in the COMMON declaration.

```
COMMON/USER/ IN(8,8),OUTT(8,8),UV(8,40),UTD(8),FAULT(24,40)
1   ,TIME,LIMIT,TITLE(20),SYSIN,SYsOUT,DATOUT
2   ,MISSION,NMSSN,ISEED,IPRINT(5),UNIT
REAL IN,OUTT,UV,UTD,TIME,LIMIT,TITLE
INTEGER FAULT(SYSIN,SYsOUT,DATOUT,MISSION,NMSSN,ISEED,IPRINT,UNIT
```
IN(I,J) Real number which is the value of the I\(^{th}\) element of the link vector at the J\(^{th}\) input port of this unit. Before calling the subroutine, the simulation substrate loads the IN array with all the current values of the vectors on each input link. Entries for I or J out of range are undefined. The subroutine may use the IN array for a scratch memory area without affecting the current link values.

UNIT Integer which is equal to the unit number (not the unit type) of the unit currently being processed. For example, if the model consists of 12 interconnected units, then UNIT will have a value from the set \{1, 2, ..., 12\}. The user should not change the value of UNIT.

FAULT(I,UNIT) Integer which equals 0 if FEC I of this unit is not currently active, and is greater than 0 if the FEC is active. The FAULT array entries are automatically changed by the simulation program according to the arrival and duration parameters supplied in the model data deck, so the FORTRAN programs should not change the array entries.

OUTT(I,J) Real number which is the value of the I\(^{th}\) element of the link vector at the J\(^{th}\) output port of this unit. All array elements are initially undefined when the subroutine is called. The user must compute all output values and store them in the proper OUTT locations. When the subroutine does a RETURN to the simulation package all necessary entries of each column are stored as an event in the next-event-list. The time of the event for each column is computed by generating a time delay to that output (specified by the user in the data deck) and adding it to the current time. When the event "happens" the proper link vector gets the column values which had been stored in the next-event list. This process is automatic; the subroutine must do nothing but compute the proper values, store them in the OUTT array, and return to the main program.
UV(I,UNIT) Real number which is the \(i\)\(^{th}\) state-variable of this unit. This array is a general storage location for each unit. All variables declared within the subroutine will be shared by all units of this same unit type, but each unit has its own column of the UV array to store state information or other general parameters which might differ among units of the same type. In the data deck the user specifies how many column entries are needed for each unit and the initial values of each entry. The UV array is reinitialized before each mission. Values within each column should be changed only by the corresponding UNIT, and except for reinitializing are never changed by the simulation main program.

TIME Real number which is the current simulated time within the current mission. The user should not change this.

LIMIT Real number which is the maximum value of simulated time for any one mission. This value, along with the initial value of TIME, is given by the user in the data deck. Again, the user must not change this.

U_TD(I) Real number which is the time delay to be used for output I when the new link values for that output are stored in the next-event list. For proper use of this variable, see Section 6.

In addition to these variables, there are certain functions available for general use. These are listed below:

REAL FUNCTION URAND(I)

\[ I = \text{any integer; its value is ignored} \]

\[ \text{URAND} = \text{uniformly distributed random value in the range} (0,1). \text{Successive calls to URAND result in independent identically distributed random values. The pseudo-} \]

\[ \text{random number generator seed is specified in the data deck.} \]

REAL FUNCTION FTIME(TIME,ID,FP)

\[ \text{TIME} = \text{current simulated TIME} \]

\[ \text{ID} = \text{integer from set \{0,1,...,9\} specifying the type of probability distribution required (see page 14).} \]

\[ \text{FP} = \text{real vector of length 3 containing the parameters used in the probability distribution (see page 14).} \]

\[ \text{FTIME} = \text{real value randomly distributed as specified. Successive calls to FTIME give independent results. The current version of FTIME doesn't use the value of TIME for any purpose.} \]
SUBROUTINE MSTOP(TM)

TM = simulated time at which the current mission should stop.
The user uses this subroutine to store a STOP event into the next-event list. If several calls are made, the current mission will stop at the smallest of LIMIT and the minimum TM used.

SUBROUTINE RSTOP

This forces the simulation to halt after the current mission. The post-run analysis subroutine is then called.

In the current implementation any standard FORTRAN function may be called, including READ and WRITE commands. System input and output must be sent through FORTRAN units SYSIN and SYSOUT, respectively, which are set by the simulation program to the values specified by the user in the data deck. The user should bear in mind that in the current implementation any records read by a READ command may never be read again in that run unless at some point in the subroutines the file unit being used is rewound.

Statistics gathering should be done by a user-written subroutine defined as

SUBROUTINE STASH(R,I).

This subroutine is called at various times to initialize tables, store default values, perform statistical analysis, and dump results.

The user should write STASH to perform the following operations, depending on the value of I for which it is called:

STASH(0.0,-1) called by main simulation program before the first mission. The data-gathering routine should initialize itself, then RETURN.
STASH(R,I) \( I > 0 \), the user calls STASH in this way from one of his own subroutines, with statistical data R to be stored or used in some way. The action taken, or the meaning of R, is signified with integer I.

STASH(0.0,-2) called by main simulation program at the end of each mission, before all arrays are reinitialized for the next mission. The user should gather data, test default conditions, and prepare for the next mission at this point.

STASH(0.0,0) called by main simulation program after all missions are completed. The user may analyze data and WRITE and PUNCH any desired statistics.

The STASH subroutine is given several parameters through the labeled COMMON area. These are

- **SYSIN** integer, FORTRAN input unit number,
- **SYSOUT** integer, FORTRAN output unit number,
- **NMSSN** integer, the total number of missions which the user specified the run should include,
- **MISSION** integer, which mission is the current mission, \( 1 < \text{MISSION} < \text{NMSSN} \),
- **TITLE** real vector of length 20 containing the 80-character title of this run,
- **ISEED** integer, the current pseudorandom number generator seed,
- **IPRINT** integer vector of length 5, entries are 0 or 1 depending on what type of output is desired. These values are specified in the data deck. In general, only the 5th entry is available for user-defined meanings in the STASH subroutine,
- **TIME** current simulated time for current mission,
- **LIMIT** maximum value of TIME specified by user.

The user should take care not to change the values of any of these variables.
The Simple Dual Computer System, described in Section 4.1 of the main report [Thompson, 1976], illustrates the use of these variables and subroutines. The arbiter (TYPE2, Figure A4) uses the function URAND to make a random choice. The monitor subroutine (TYPE3, Figure A5) stores times-to-failure through the STASH subroutine, and calls MSTOP when it determines the simulation of that mission should stop. The STASH routine (Figure A6) stores the times-to-failure in its own declared array, TSAVE. At the end of the simulation run, the STASH routine uses a special subroutine, OUTPUT, to sort the times-to-failure in numerical order. This subroutine plots the probability of failure (proportion failed) versus time to obtain the reliability curve for the system. This example does not use all the facilities offered by the simulator package, but does demonstrate the manner in which such facilities are used in general.

In the current implementation, all data analysis is left to the person who programs STASH. As the simulation package is developed, there will be a variety of subroutines made available to the user for data reduction, special input/output, statistical analysis, and graph plotting. In particular, there should be a subroutine which produces a graph of mission reliability given a vector of mission times, which will involve special sorting algorithms and some type of optimal-estimation curve fitting. This subroutine would perform the same function as the OUTPUT routine called from STASH in the example above.
3. System Parameter Data-Deck

3.1 Segment Descriptions

The network definition of the model to be simulated is encoded in a data deck that describes each link and unit of the network and any preset events that are included in the simulation.

The data deck has three major sections separated by blank cards:

- LINK: mission information and link initial values
- UNIT: unit parameters
- EVENT: preset events

Each segment is a sequence of card types usually divided into subsequences.

**LINK Segment**

<table>
<thead>
<tr>
<th>Card #</th>
<th>Card Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>file identification</td>
</tr>
<tr>
<td>2.</td>
<td>title</td>
</tr>
<tr>
<td>3.</td>
<td>time</td>
</tr>
<tr>
<td>4.</td>
<td>link i values</td>
</tr>
<tr>
<td>5.</td>
<td>link j values</td>
</tr>
<tr>
<td></td>
<td>N cards for the N links of the model. Order is unimportant, but each link must be described.</td>
</tr>
<tr>
<td>N+3</td>
<td>link w values</td>
</tr>
<tr>
<td>N+4</td>
<td>blank</td>
</tr>
</tbody>
</table>

**UNIT Segment**

A descriptive sub-sequence for a particular unit consists of:

<table>
<thead>
<tr>
<th>Card #</th>
<th>Card Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>unit i descriptors (includes IU, OU, FU)</td>
</tr>
<tr>
<td>2.</td>
<td>input link descriptors</td>
</tr>
<tr>
<td>3.</td>
<td>unit i output card</td>
</tr>
<tr>
<td></td>
<td>One card for each output of the unit. (OU is the number of outputs).</td>
</tr>
<tr>
<td>OU+2</td>
<td>unit i output card,</td>
</tr>
<tr>
<td>Card #</td>
<td>Card Type</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>OUi+3</td>
<td>unit i FEC m card 1 } 2 cards</td>
</tr>
<tr>
<td>OUi+4</td>
<td>unit i FEC m card 2 } per fault</td>
</tr>
<tr>
<td></td>
<td>2 × FUi cards</td>
</tr>
<tr>
<td>OUi+2FU1+1</td>
<td>unit i FEC k card 1</td>
</tr>
<tr>
<td>OUi+2FU1+2</td>
<td>unit i FEC k card 2</td>
</tr>
<tr>
<td>OUi+2FU1+3</td>
<td>state variables</td>
</tr>
</tbody>
</table>

For both the faults and outputs, the order in which they are specified is unimportant. The last unit descriptor subsequence must be followed by a blank card.

**EVENT Segment**

When there are pre-specified events, there will be one or two cards per event depending on the type of event. (See the card type descriptions.) The sequence, which may be empty, is terminated by a blank card.

### 3.2 Card Type Descriptions

<table>
<thead>
<tr>
<th>File Identification Card</th>
<th>Format (3I4)</th>
<th>Fortran I/O unit number on which the rest of the data deck is found.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Fortran I/O unit number on which to print the system configuration and run-time messages.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fortran I/O unit number on which to store any output data.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title Card</th>
<th>Format (20A4)</th>
<th>title to be printed for this run</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Time Card</th>
<th>Format (2E12.6, 18, 512, 130)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col. 1-12</td>
<td>starting simulated time for this run</td>
</tr>
<tr>
<td>Col. 13-24</td>
<td>simulated time upper bound (LIMIT)</td>
</tr>
<tr>
<td>Col. 25-32</td>
<td>how many missions to simulate, maximum (NMSSN)</td>
</tr>
<tr>
<td>Col. 34</td>
<td>= 1 for list of all events after each cycle</td>
</tr>
<tr>
<td>Col. 36</td>
<td>= 0 no list</td>
</tr>
<tr>
<td></td>
<td>= 1 for ordered event list for each run</td>
</tr>
<tr>
<td></td>
<td>= 0 no list</td>
</tr>
</tbody>
</table>

10
### Time Card

**Col. 38**

Format \(2\text{E}12.6, 18, 512, 130\) (Cont'd)

- \(= 1\) table of mission times
- \(= 0\) no table

**Col. 40**

- \(= 1\) listing of system configuration and parameters
- \(= 0\) no listing

**Col. 42**

- User-specified print option control
- Integer seed for random number generator

**Col. 43-72**

### Link Values Card

**Col. 1-4**

Format \(214, 6\text{F}12.6/(8\text{X}, 6\text{F}12.6)\)

- Number of link \((L_1 \rightarrow 1, L_4 \rightarrow 4, \text{etc.})\)
- Length of vector required to store the values on this link.
- Initial value of first link element
- Initial value of second link element
- Etc.

### Unit Descriptors

**Col. 1-4**

Format \(614\)

- Number of unit \((U_1 \rightarrow 1, U_4 \rightarrow 4, \text{etc.})\)
- Unit type
- Number of link input ports to the unit
- Number of link output ports from the unit
- Number of faults specified for that unit
- Number of state-variables for the unit

The numbers in columns 9-24 determine how many values are read in on the immediately succeeding cards. If any of these numbers is 0, the corresponding data cards should not be there (i.e., do not put a blank card in sequence).

### Unit Input Link Descriptors Card(s)

Format \(2014\)

If more than 20 links are input to the unit, 2 or more cards will be used.

**Col. 1-4**

Number specifying which link is connected to input port \#1 of this unit. If a change of the link value always implies that the unit must be processed to recompute its state or outputs, then this field holds the link number. If a change of the link value does not force the unit to be processed, then this field holds the negative of the link number.
Col. 5-8: same as above, for input #2 of this unit
Col. 9-12: same as above, for input #3 of this unit
etc.

Unit Output Card(s) Format (3I4, 3F12.6)

One card for each output of this unit:

Col. 1-4: number of output port for this unit type
Col. 5-8: link which is connected to this output port
col. 9-12: type of probability distribution of time delay
If the time delay should be distributed
probabilistically for different missions
but equal to a constant for any one mis-
sion, this integer entry should be the
negative of the index number specifying
the probability distribution.
Col. 13-24: parameter #1 of prob. dist.
Col. 25-36: parameter #2 of prob. dist.
Col. 37-48: parameter #3 of prob. dist.

Unit Fault Cards

Two cards required for each FEC:

Card 1: fault description card, Format (514)

Col. 1-4: number used to identify the FEC for
this unit
Col. 5-8: type of distribution time to occurrence
of next fault in the FEC:
= 0 never occurs
= 1 exponential (Poisson)
= 2 gamma
= 3 beta
= 4 Erlang
= 5 Weibull
= 6 constant
= 7 normal
= 8 uniform
= 9 Pascal
Col. 9-12: type of distribution of fault duration
= 0 never disappears
others same as above
Col. 13-16: initial state of FEC
$\geq 1$ = active fault
0 = not active
Col. 17-20 Multiplicity of FEC. This must be 0 or 1 if the occurrence and distribution types are not 0 or 1. The multiplicity of an FEC is the number of independent elements assumed to have the given distributions, and the value of FAULT(I,UNIT) is then an integer between 0 and the multiplicity specifying how many of these independent elements have active faults at that time. See Section 6 for a more complete explanation.

- Card 2: distribution parameters, Format (6E12.6)

| Col. 1-12 | parameter $\alpha_1$ for occurrence distribution |
| Col. 13-24 | parameter $\alpha_2$ for occurrence distribution |
| Col. 25-36 | parameter $\alpha_3$ for occurrence distribution |
| Col. 37-48 | parameter $\beta_1$ for duration distribution |
| Col. 49-60 | parameter $\beta_2$ for duration distribution |
| Col. 61-72 | parameter $\beta_3$ for duration distribution |

State-Variable Card(s) Format (6E12.6, 8X)

| Col. 1-12 | state-variable #1 of this unit |
| Col. 13-24 | state-variable #2 of this unit |
| etc. | etc. |

State-Variables are only used by the user-defined subroutine for the unit type. The $I^{th}$ state-variable of the currently processed UNIT is $UV(I,UNIT)$.

Event Card Format (E12.6,4I4)

| Col. 1-12 | time of event |
| Col. 13-16 | type of event: |
| | 1 link values change |
| | 2 unit FEC state change |
| | 3 unit type changes |
| | 4 stop mission |
| Col. 17-20 | a link number for type 1, a unit number for types 2 and 3, leave blank for event-type 4 |
| Col. 21-24 | FEC number for event-type 2, new unit type for event-type 3; otherwise, leave blank |
| Col. 25-28 | new value of fault state (non-negative integer) for event-type 2; otherwise, leave blank |
Event Values Card(s) Format (6E12.6, 8X)

This card is not used for event types 2, 3, and 4.

Col. 1-12 new value #1 of link vector
Col. 13-24 new value #2 of link vector
e tc.
e tc.

The probability distributions are generated as the type specified with the following parameters $\alpha_1, \alpha_2, \alpha_3$ (or $\beta_1, \beta_2, \beta_3$):

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>P.D.F.</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>exponential</td>
<td>$e^{-\alpha_1 t}$</td>
<td>rate = $\alpha_1$</td>
</tr>
<tr>
<td>2</td>
<td>gamma</td>
<td>$\frac{(\alpha_1 t)^{\alpha_2}}{t^\Gamma(\alpha_2)} e^{-\alpha_1 t}$</td>
<td>rate = $\alpha_1$, order = $\alpha_2$</td>
</tr>
<tr>
<td>3</td>
<td>beta</td>
<td>$\frac{1}{B(\alpha_1, \alpha_2)} t^{\alpha_1-1}(1-t)^{\alpha_2-1}$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>binomial</td>
<td>$\binom{\alpha_2}{K} \alpha_1^{1-\alpha_1} \alpha_2^{-K}$</td>
<td>where $K = \lfloor t/\alpha_3 \rfloor$, $\alpha_3 =$ time increment</td>
</tr>
<tr>
<td>5</td>
<td>Weibull</td>
<td>$e^{-\alpha_1 t^{\alpha_2}}$</td>
<td>rate = $\alpha_1$, aging factor = $\alpha_2$</td>
</tr>
<tr>
<td>6</td>
<td>constant</td>
<td>$\delta(t-\alpha_1)$</td>
<td>$t \equiv \alpha_1$</td>
</tr>
<tr>
<td>7</td>
<td>normal</td>
<td>$\frac{1}{\sqrt{2\pi\alpha_2}} \exp\left(-\frac{1}{2} \frac{(t-\alpha_1)^2}{\alpha_2} \right)$</td>
<td>mean = $\alpha_1$, variance = $\alpha_2$</td>
</tr>
<tr>
<td>8</td>
<td>uniform</td>
<td>$\frac{1}{\alpha_2-\alpha_1}$</td>
<td>min $(\alpha_1, \alpha_2) &lt; t &lt; \max (\alpha_1, \alpha_2)$</td>
</tr>
<tr>
<td>9</td>
<td>Pascal</td>
<td>$\alpha_1 (1-\alpha_1)^{K-1}$</td>
<td>$\alpha_1 =$ step probability</td>
</tr>
</tbody>
</table>

$K = \lfloor t/\alpha_2 \rfloor$, $\alpha_2 =$ time increment per step.

(* → not yet implemented)
4. Flow of Control During Simulation

The execution of the simulation package is one "run", which usually will consist of many missions. For any one mission all the arrays are initialized to values specified by the user in the data-deck, the fault events are generated according to their specified distribution, and then events are processed according to their simulated times of occurrence. When the mission processes a stop event, the mission is over, and everything is reinitialized for the next mission. The basic flow chart of the simulation package is shown in Figure 1. The seed for the random number generator is set by the user at the beginning of the run and is purposely not reset at the start of each mission. Thus, the generation of fault times will be different for each mission.

As indicated in the data-deck description, it is possible to specify time delays such that the time delay is constant for any one mission, but the value of that constant is randomly distributed for different missions. All such constants are generated immediately after the initial fault events are created at the start of each mission.

The basic flow chart for simulation of any one mission is shown in Figure 2. There are two loops in the flowchart. The upper loop processes the next-event and all succeeding events which take place at the same time as the first next-event. Most events will be a link value change, fault change from active to inactive or vice versa, or stop event for the mission. All events which affect a unit in some way will cause that unit number to be put in an affected-unit list, and any one unit number is only put in the list once for the
START

READ IN DATA DECK

STORE ALL INITIAL VALUES

SET RANDOM SEED

CALL STASH(0.0,-II)

MISSION + II

MISSION = MISSON + II

NO

MISSON ? > NMSSN ?

YES

CALL STASH(0.0,0)

STOP

RESTORE ALL INITIAL VALUES

GENERATE ALL FAULT EVENTS

GENERATE ALL TIME DELAYS CONSTANT FOR THIS MISSION

SIMULATE ONE MISSION

CALL STASH(0.0,-2)

RSTOP ? CALLED ?

NO

YES

Figure 1. Flow Chart for One Simulation Run.
BEGIN
SIMULATION OF
ONE MISSION
CLEAR LIST OF AFFECTED UNITS
MSTOPS = 0
GET NEXT EVENT
SET TIME = TIME OF THIS EVENT

1. STORE NEW LINK VALUE
2. STORE FAULT VALUE
3. STORE NEW UNIT TYPE
4. MSTOP + 1

GENERATE NEW FAULT EVENT

YES

? CHANGE FROM PREVIOUS VALUE

NO

ADD ALL AFFECTED UNITS TO LIST

REMOVE THIS EVENT FROM NEXT-EVENT-LIST

NEXT EVENT

? AT CURRENT TIME?

YES

NO

? MSTOP = 1?

YES

CALL STASH(0.0,-2)

continue (see Fig. 1)

NO

? AFFECTED UNIT LIST EMPTY?

YES

GET NEXT AFFECTED UNIT
TRANSFER INPUT LINK VALUES TO IN ARRAY
GENERATE TIME DELAY FOR EACH OUTPUT
CALL USER'S SUBROUTINE FOR THAT UNIT TYPE
INSERT OUT ARRAY COLUMNS INTO NEXT-EVENT-LIST
REMOVE THIS UNIT FROM AFFECTED-UNIT LIST

Figure 2. Flow Chart for Simulation of One Mission.
same simulated time. If the event sets a parameter equal to the same value as it had immediately before the event then nothing really changes, and such an event cannot affect any units. In particular, a link change event (event type 1) will affect all units which have that link as an input if and only if at least one of the values of the new link vector is different from that entry in the old link vector. Event type 2 also puts the specified unit on the affected-units list. After all events at the same time have been processed, all units in the affected-units list are processed in an arbitrary order. The main processing for each unit is accomplished by calling the FORTRAN subroutines written by the user for that unit type. When the subroutine returns to the main simulation program, the current simulated time is added to the time delays left in the UTD array. The result is the time at which each output link will change its value, and is used to order these future events in the next-event list.

Event type 4 causes the mission to stop at the time of the event. All changes which would occur at the same time are processed, such as link value changes, but no affected units are processed. Thus, at the time of the end of mission, the second (bottom) loop is not entered.

When the state of a FEC (active/inactive) changes, the simulator automatically generates a new fault event which at some future time will change the FEC to its next state. The time delay to this new event is pseudorandomly generated according to the probability distribution specified by the user in the data-deck. There are exceptions to this rule which are discussed in Section 5.
5. Generation of Randomly Distributed Faults and Time Delays

The time to next fault event and the time delays to unit outputs can be specified by one of several probability distributions (see end of Section 3). Given a probability distribution \( f(t, a) \), where \( t \) is the random variate and \( a \) is a vector of fixed parameters for the function, it is possible to pseudorandomly generate independent values of \( t \) which are identically distributed by \( f(\cdot, a) \). This is done by the subroutine FTIME described in Section 2, which in turn makes frequent calls to the subroutine URAND for uniform \((0,1)\) variates. URAND is written in such a way as to optimize its own operation to the characteristics of the computer on which the simulator is being executed, so as to insure a very high degree of randomness in the results [Knuth, 1969]. For this reason the simulation could generate different (although identically distributed) random values from the same initial seed on different computers if the computers do not have the same word length and arithmetic conventions. The cycle length for the pseudorandom generator will be close to the highest integer represented in the computer, but since most applications will use FTIME for several different types of distribution and meaning, the effective cycle length is much longer. This is true because each call to FTIME or URAND takes the integer random seed through one or several steps of the complete cycle.

At the beginning of a mission, the simulator generates a fault event for each FEC in the system. The event causes a change
of the active/inactive state of the FEC. Each time an event of this type is processed it automatically generates a new event which at some future time will change the FEC state to its next value. As an example, consider a system which is only one unit with no inputs and outputs, initial time TIME = 0.0 and maximum mission time LIMIT = 150.0. The single unit performs no operation, but has four FECs which become active and inactive according to prespecified distributions. Each FEC has a multiplicity of 1. All four FECs have an exponential distribution of occurrence with rate $\alpha_1 = 100.0$ and a constant "distribution" of duration with constant $\beta_1 = 80.0$. FECs 1, 2, and 4 are initially inactive, and FEC 3 is initially active. The data-deck cards describing the unit would be

1 = 1 2. 0 4 0
1 1 6 0 1
100.0 0.0 0.0 80.0 0.0 0.0
2 16 0 1
100.0 0.0 0.0 80.0 0.0 0.0
3 16 1 1
100.0 0.0 0.0 80.0 0.0 0.0
4 16 0 1
100.0 0.0 0.0 80.0 0.0 0.0

At TIME = 0.0 four events would be inserted into the next-event list, one for each FEC:

<table>
<thead>
<tr>
<th>Time Order</th>
<th>Time of Event</th>
<th>Type of Event</th>
<th>Unit</th>
<th>FEC</th>
<th>New Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>97.8</td>
<td>fault</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>126.3</td>
<td>fault</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>32.2</td>
<td>fault</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>150.0</td>
<td>stop</td>
<td>-</td>
<td>-</td>
<td>(stop event)</td>
</tr>
</tbody>
</table>
The times 97.8, 126.3, and 32.2 were generated by the FTIME subroutine from the exponential distribution with $\alpha_1 = 100.0$. The system immediately jumps to TIME = 32.2 and changes FEC number 4 of unit number 1 to have new value 1 (active fault), then inserts the event which changes that FEC back to value 0. The next-event list now looks like this:

<table>
<thead>
<tr>
<th>Time Order</th>
<th>Time of Event</th>
<th>Type of Event</th>
<th>Unit</th>
<th>FEC</th>
<th>New Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>97.8</td>
<td>fault</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>126.3</td>
<td>fault</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>80.0</td>
<td>fault</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>150.0</td>
<td>stop</td>
<td>-</td>
<td>-</td>
<td>(stop event)</td>
</tr>
<tr>
<td>3</td>
<td>112.2</td>
<td>fault</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

The new time 112.2 was obtained by adding the fault duration time (80.0) to the time when the fault became active. A chart of FEC states vs. time for each FEC is shown in Figure 3.

When FEC 3 became inactive at TIME = 80.0, a call to FTIME gave a delay of 28.4, so that FEC 3 became active again at TIME = 108.4.
When FEC 4 became inactive at TIME = 112.2, the random variate generated was 49.8, so next-event for FEC 4 would have occurred at TIME = 162.0 if the mission had not been forced to stop at LIMIT = 150.0. This entire process is handled automatically by the simulation program.

When the occurrence distribution is simple exponential (or never occurs) and the duration distribution is simple exponential (or never disappears) for a set of faults which all have the same effect, the fault generation can be made more efficient by including these into one FEC with a multiplicity greater than 1. One entry of the FAULT array would then correspond to a set of M single faults, where M is the multiplicity specified in the data deck. The integer value of that entry is the number of single faults active at that time. For example, suppose faults in a 16-bit bus driver are to be simulated, and each single driver has fault occurrence and duration distributions which are simple exponential with parameters $\lambda_0$ and $\mu_0$, respectively. The value $F$ of that entry in the FAULT array follows a sequence generated by the Markov chain shown in Figure 4 for $M=16$. The rates of change of the $F$ values are shown on the arcs between states. Rather than generating a separate event for each of the 16 faults, only one event is generated which changes the value of $F$ by plus or minus 1. Thus, $F$ varies from 0 to 16, and $F$ equals 0 only when all 16 bus drivers are fault free. Note that this facility can only be used with exponentially distributed faults; all other FECs must be specified with multiplicity equal to 1 or 0.
Fig. 4. Markov Chain Generating Multiple-Fault States For One Fault-Equivalent Class

- $F$ = value of variable associated with this FEC,
- $M$ = multiplicity (number of single faults) for this FEC,
- $\lambda_0$ = constant rate of occurrence for each single fault,
- $\mu_0$ = constant rate of repair for each single fault.

The arcs are labelled with the relative rates of transition from each state.


6. Resolving Time Delay Ambiguities

It is possible that certain ambiguities might occur in the generation of events due to the dynamic use of probabilistic time delays. For example, consider a unit with one FEC, one input, one output, and a constant time delay to the output of 10.0. Suppose that at TIME = 100.0 the input changes, so an event is stored to change the output at future TIME = 110.0. However, at TIME = 105.0 the FEC becomes active and affects the output in such a way that the new link values which would have been set TIME = 110.0 are no longer valid. Instead, the user wants the output link of the unit to be set to some different values at an earlier time, say at TIME = 107.0, and the previously stored event should be removed from the next-event list. A similar problem occurs when an FEC becomes inactive before the effect of the FEC being active has had a chance to cause an error indication at the unit's output.

In order to allow the user to resolve these ambiguities, two facilities have been added to the simulation package. The first is that each type subroutine is given the time delay generated to each output and is free to override them with its own computed values. Specifically, the following array is available:

\[ \text{UTD}(I) \]

Real number which is the time delay to be used for output I when the new link values for that output are stored in the next-event list. The time of the event will be \( \text{TIME} + \text{UTD}(I) \). Before the unit subroutine is called, all UTD entries are generated according to the time delay distributions specified in the data deck. The subroutine may replace entries with any real values to change the time delay in that instance.
The other facility is the rule used by the simulation package when storing events into the next-event-list. If the event is a change of link value then all other previously stored events which are value changes of that link are removed from the event list if and only if those previously stored events would have occurred at a later simulated time than the event now being stored. In the example above the subroutine should replace the value 10.0 in UTD(1) with the value 2.0. When the event to occur at TIME = 107.0 is stored, the simulation program will automatically remove from the list the event which would have occurred at TIME = 110.0. This procedure should eliminate the restrictions caused by various timing anomalies.

7. APPENDIX A: FORTRAN Source Listing of Simulation Package

The following programs are written in ANSI FORTRAN IV (non-extended). When using the simulator on a particular computer, be sure to adjust the URAND function according to that computer's arithmetic conventions (see comments inserted in the function listing). If array sizes need to be changed to accommodate a larger model, change the corresponding declarations in all subroutines. Changing array sizes might also require changing the length of vectors ABLES, ANET, AUSER, and ALUES. These vectors are used in SETUP and the MAIN program, and are equivalenced to the various labeled COMMON areas. See the SETUP initialization sequence for a list of relevant array-size parameters.
C A SIMULATOR FOR THE EVALUATION OF RELIABILITY OF
C SYSTEMS COMPOSED OF USER-DEFINED UNITS WHICH ARE
C INTERCONNECTED IN AN ARBITRARY FASHION.
C
C PETER A. THOMPSON 1976
C DIGITAL SYSTEMS LABORATORY, STANFORD UNIVERSITY
C
C FOR USER MANUAL, SEE DSL T.N.#132
C FOR BASIC DESCRIPTION, SEE DSL T.R.#119.
C FOR EXAMPLE OF USE FOR A LARGE DUAL COMPUTER SYSTEM,
C SEE DSL T.R.#121
C
C REQUIRES DECLARATION OF FORTRAN TAPE UNIT NO. 4
C ALL CODE SUITABLE FOR ANSI FORTRAN 4, NON-EXTENDED
C
C INTEGER EORDER, ETYPE, ELORU, EHEAD, ELNGTH, ENVAL
C COMMON /NET/ET(300),ETYPE(300),ELORU(300),EVAL(16,300)
C EORDER, EHEAD, ELNGTH, NEVT
C ET,ETYPE,ELORU, EVAL ARE PARALLEL TABLES WHICH HOLD EVENT
C TIME, TYPE, LINKORUNIT, AND VALUES RESPECTIVELY.
C EORDER GIVES TRUE SEQUENCE OF THE EVENT ENTRIES.
C COMMON/USER/
C IN(8,8),OUTT(8,8),UV(8,40),UTD(8),FAULT(24,40)
C TIME, LIMIT, TITLE(20), SYSIN, SYSOUT, DATOUT
C REAL IN, OUTT, UV, UTD, TIME, LIMIT, TITLE
C INTEGER FAULT, SYSIN, SYSOUT, DATOUT, MSSION, NMSSN, ISEED, IPRINT, UNIT
C COMMON /TABLES/
C UIN(8,40), UOUT(8,40), UT(40), UTD(6,24,40), EUAFFC(40), PFT(3)
C COMMON /VALUES/
C NLK, NUAFFC, NU, NUOUT, NUTDP, NUV, NFLT, NLY, NL, NUTD
C DIMENSION ABLES(13467), ANET(6303), AUSER(1428), ALUES(14)
C EQUIVALENCE (ABLES(1), UIN(1,1))
C EQUIVALENCE (ANET(1), ET(1))
C EQUIVALENCE (AUSER(1), IN(1,1))
C EQUIVALENCE (ALUES(1), NLK)
C INTEGER ADDU, INSERT
C REAL FLTGEN, FTIME

C ** END SPECIFICATION SECTION ** BEGIN DATA SECTION **
C
C NEVT=300
C NEVT IS THE MAXIMUM NUMBER OF PENDING EVENTS
C
C DO 5 I=1,NEVT
C ET(I)=O.O
C 5 EORDER(I)=O
C EHEAD=1
C ELNGTH=O
C SIGNF=1.E-6
C
C DELTA=10.0
C
C***********************
C SETUP THE RUN********************
C CALL SETUP
C SYSTM=TIME
C MSSION=O
C MISTOT=O
C X=CHOICE(-1,1,0.0)
C IUSED=O
C CALL STASH(0.0,-1)
C**SIMULATE ONE MISSION***************
55 IF(USED.EQ.0) GO TO 50
REWIND 4
READ (4)ABLES,ANET,AUSER,ALUES
X=CHIOCR(-1,1,0.0)
IUSED=0
50 MISSION=MISSION+1
MISTOT=MISTOT+1
IF(MISSION.GT.NMISSION) GO TO 51
TIME=SVTIME
C**GENERATE INITIAL FAULT EVENTS***************
DO 600 I=1,NU
DO 600 J=1,NFLT
IF(FLTD(3,J,I).LE.0) GO TO 600
X=FLTGEN(I,J,1)
600 CONTINUE
C**GENERATE ALL CONSTANT TIME DELAYS
DO 105 I=1,NU
DO 105 J=1,NUOUT
IF(UTDD(J,I).GE.0) GO TO 105
DO 106 K=1,NUTDP
PFT(K)=UTDP(K,J,I)
106 UTDP(K,J,I)=0.0
UTDP(1,J,I)=FTIME(TIME,-UTDD(J,I),PFT)
105 CONTINUE
C**GET ALL EVENTS AT NEXT SMALLEST TIME
NUAFFC = 0
IF(IPRINT(1).EQ.1).OR.(IPRINT(2).EQ.1))WRITE(SYSOUT,103)MISSION
103 FORMAT(13H1++***MISSION ,18,6H ****//)
WRITE(SYSOUT,6000)MISTOT
1000 FORMAT(2H /,110)
102 IF(IPRINT(1).EQ.1) CALL EDUMP
103 CONTINUE
104 CONTINUE
C
105 NUAFFC= 0
106 IF(IPRINT(1).EQ.1).OR.(IPRINT(2).EQ.1)) WRITE(SYSOUT,103)MISSION
103 FORMAT(13H1++***MISSION ,18,6H ****//)
WRITE(SYSOUT,6000)MISTOT
1000 FORMAT(2H /,110)
102 IF(IPRINT(1).EQ.1) CALL EDUMP
103 CONTINUE
104 CONTINUE
105 NUAFFC= 0
106 IF(IPRINT(1).EQ.1).OR.(IPRINT(2).EQ.1)) WRITE(SYSOUT,103)MISSION
103 FORMAT(13H1++***MISSION ,18,6H ****//)
WRITE(SYSOUT,6000)MISTOT
1000 FORMAT(2H /,110)
102 IF(IPRINT(1).EQ.1) CALL EDUMP
103 CONTINUE
104 CONTINUE
C
105 NUAFFC = 0
106 IF(IPRINT(1).EQ.1).OR.(IPRINT(2).EQ.1)) WRITE(SYSOUT,103)MISSION
103 FORMAT(13H1++***MISSION ,18,6H ****//)
WRITE(SYSOUT,6000)MISTOT
1000 FORMAT(2H /,110)
102 IF(IPRINT(1).EQ.1) CALL EDUMP
103 CONTINUE
104 CONTINUE
C
105 NUAFFC = 0
106 IF(IPRINT(1).EQ.1).OR.(IPRINT(2).EQ.1)) WRITE(SYSOUT,103)MISSION
103 FORMAT(13H1++***MISSION ,18,6H ****//)
WRITE(SYSOUT,6000)MISTOT
1000 FORMAT(2H /,110)
102 IF(IPRINT(1).EQ.1) CALL EDUMP
103 CONTINUE
104 CONTINUE
C
105 NUAFFC = 0
106 IF(IPRINT(1).EQ.1).OR.(IPRINT(2).EQ.1)) WRITE(SYSOUT,103)MISSION
103 FORMAT(13H1++***MISSION ,18,6H ****//)
WRITE(SYSOUT,6000)MISTOT
1000 FORMAT(2H /,110)
102 IF(IPRINT(1).EQ.1) CALL EDUMP
103 CONTINUE
104 CONTINUE
C
105 NUAFFC = 0
106 IF(IPRINT(1).EQ.1).OR.(IPRINT(2).EQ.1)) WRITE(SYSOUT,103)MISSION
103 FORMAT(13H1++***MISSION ,18,6H ****//)
WRITE(SYSOUT,6000)MISTOT
1000 FORMAT(2H /,110)
102 IF(IPRINT(1).EQ.1) CALL EDUMP
103 CONTINUE
104 CONTINUE
C
CONTINUE
C FAULT CHANGE
IUSED = 1
IU = ELORU(ICUR)
IM = IFIX(EVAL(1, ICUR))
IF = IFIX(EVAL(2, ICUR))
TEST IF NEW DIFFERENCE FROM OLD
K = FAULT(IF, IU) - IM
IF(K .EQ. 0) GO TO 131
ADD UNIT TO AFFECTED UNIT LIST
IF(ADDU(IU).GT.0) GO TO 132
STOP
132 FAULT(IF, IU) = IM
C GENERATE NEW FAULT CHANGE EVENT
X = FLTGEN(IU, IF, l)
131 GO TO 200
CONTINUE
C UNIT TYPE CHANGE
IUSED = 1
IU = ELORU(ICUR)
UT IS INTEGER, BUT THE NEW VALUE IN EVAL IS FLOATING
TEMP = EVAL(1, ICUR)
TEMP2 = UT(IU)
C CHECK FOR UNIT TYPE CHANGE
TEMP2 = ABS(TEMP) - ABS(TEMP2)
UT(IU) = TEMP
ICHNG = 0
IF (ABS(TEMP2) .GT. SIGNF) ICHNG = 1
IF ((ICHNG .LE. 0) .AND. (ADDU(IU).GT.0)) GO TO 200
GO TO 200
CONTINUE
C STOP THIS RUN
ISTOP = 1
GO TO 200
CONTINUE
C ELIMINATE EVENT AT HEAD OF LIST
EHEAD = EORDER(EHEAD)
EORDER(ICUR) = 0
ELNGTH = ELNGTH - 1
C IF THERE IS ANOTHER EVENT AT THE SAME TIME PROCESS IT
TEMP = ABS(ET(INXT)) - ABS(TIME)
IF((ELNGTH .GT. 0) .AND. (ABSCTEMP) .LE. SIGNF) GO TO 101
700 CONTINUE
C ALL EVENTS AT THE CURRENT TIME HAVE BEEN PROCESSED, HANDLE AFFECTED UN
IF(ISTOP.EQ.1) GO TO 52
IF(NUAFFC.NE.0) GO TO 703
IF(IPRINT(1).EQ.1) WRITE(SYSOUT, 702)
702 FORMAT(17HOO UNITS AFFECTED)
GO TO 800
703 IF(IPRINT(1).EQ.1) WRITE(SYSOUT, 704) NUAFFC, (EUAFFC(KX), KX = 1, NUAFFC)
704 FORMAT(1H0, I4, 16H UNITS AFFECTED,,2014)
DO 799 NUNIT = 1, NUAFFC
UNIT = EUAFFC(NUNIT)
C*******GENERATE OUTPUT TIME DELAYS
IQ = 0
715 IQ = IQ + 1
IL = UOUT(IQ, UNIT)
IF (IL) 710, 710, 705
710 OUTF(IQ) = LK(1, IL)
DO 706 IF=1,3
706 PFT(IP) = UTDP(IP, IQ, UNIT)
UTD(IQ) = PTIME(TIME, UTD(IQ, UNIT), PFT)
715 GO TO 715
710 OOUT = IQ - 1
216. C#####TRANSFER INPUT BUFFER
219. IQ=0
220. 730 IQ=IQ+1
221. IL=UNIQ(IQ,UNIT)
222. IF (IL) 725, 725, 720
223. 720 IP=LK(1, IL)
224. DO 721 IR= 1, IP
225. 721 IN(IR, IQ>= LV(IR, IL)
226. GOTO 730
227.
228. CONTINUE
229.
230. 740 ITYPE= UT(UNIT)
231. GO TO (801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828), ITYPE
232. 801 CALL TYPE1
233. GOTO 760
234. 802 CALL TYPE2
235. GOTO 760
236. 803 CALL TYPE3
237. GOTO 760
238. 804 CALL TYPE4
239. GO TO 760
240. 805 CALL TYPE5
241. GO TO 760
242. 806 CALL TYPE6
243. GO TO 760
244. 807 CALL TYPE7
245. GO TO 760
246. 808 CALL TYPE8
247. GO TO 760
248. 809 CALL TYPE9
249. GO TO 760
250. 810 CALL TYPE10
251. GO TO 760
252. 811 CALL TYPE11
253. GO TO 760
254. 812 CALL TYPE12
255. GO TO 760
256. 813 CALL TYPE13
257. GO TO 760
258. 814 CALL TYPE14
259. GO TO 760
260. 815 CALL TYPE15
261. GO TO 760
262. 816 CALL TYPE16
263. GO TO 760
264. 817 CALL TYPE17
265. GO TO 760
266. 818 CALL TYPE18
267. GO TO 760
268. 819 CALL TYPE19
269. GO TO 760
270. 820 CALL TYPE20
271. GO TO 760
272. 821 CALL TYPE21
273. GO TO 760
274. 822 CALL TYPE22
275. GO TO 760
276. 823 CALL TYPE23
277. GO TO 760
278. 824 CALL TYPE24
279. GO TO 760
280. 825 CALL TYPE25
281. GO TO 760
282. 826 CALL TYPE26
283. GO TO 760
284. 827 CALL TYPE27
285. GO TO 760
286. 828 CALL TYPE28
287. GO TO 760
288. 829 CALL TYPE29
289. GO TO 760
290. continue
291.
C***
177 DO 780 IQ=1, IOUT
785 IF(INSERT(UNIT, IQ), 1, UOUT(IQ, UNIT), VAL, IP))
1 780 WRITE(SYSOUT, 781) UNIT, TIME
781 FORMAT(23H INSERT ERROR WITH UNIT, 13,.qq9H AT TIME , E13.6)
STOP
770 CONTINUE
776 CONTINUE
799 CONTINUE
306.
C**********************************************************************
307. CONTINUE
308. IF (IPRINT(1).EQ.1) CALL EDUMP
309. IF(ELNGTH.GT.0) GO TO 100
310.
C***END OF MISSION************
311. 52 CALL STASH(0.0,-2)
312. GO TO 55
313.
C***END OF RUN**************
314. 51 CALL STASH(0.0,0)
315. WRITE(SYSOUT,999)ISEED
316. 999 FORMAT(/12H FINAL SEED ,130)
STOP
END

SUBROUTINE SETUP
C READS FORMATTED DATA DECK AND CONFIGURES MODEL
C
COMMON /TABLES/ UIN(8,4O),UOUT(8,4O),UT(4O),UTDD(8,4O),
1 FLDP(3,24,4O),FLTP(6,24,4O),EUAFFC(4O),PFT(3)
2 ,OUTN(8)
3 REAL UTDP,VAL,LV,FLTP,PFT
INTEGER UIN,UOUT,UT,UTDD,LK,FLTD,EUAFFC,OUTN
1 REAL IN,OUTT,UV,UTD,TIME,LIMIT,TITLE
2 INTEGER ISEED,IPRINT,UNIT
COMMON/USER/ IN(8,8),OUTT(8,8),Uv(8,4O),UTD(8),FAULT(24,4O)
1 TIME,LIMIT,TITLE(20),SYSIN,SYSOUT,DATOUT
2 COMMON /VALUES/ NLK,NUAFFC,NU,NUOUT,NUDIP,NUV,NFLT,NLV,NL,NUTD
1 NUIN,NVAL,NFLT,PFT,NUTDD
3 INTEGER EORDER,ETYPE,ELORU,EHEAD,ELNGTH,ENVAL
COMMON /NET/ET(300),ETYPE(300),ELORU(300),EVAL(16,300)
1 ,ENVAL(300),EORDER(300),EHEAD,ELNGTH,NEVT
C ET,ETYPE,ELORU, EVAL ARE PARALLEL TABLES WHICH HOLD EVENT
C TIME,TYPE,LINKORUNIT, AND VALUES RESPECTIVELY.
C EORDER GIVES TRUE SEQUENCE OF THE EVENT ENTRIES.
DIMENSION ABL(13467),ANET(6303),AUSER(1428),ALUES(14)
EQUIVALENCE (ABLES(1),UIN(1,1))
EQUIVALENCE (ANET(1),ET(1))
EQUIVALENCE (AUSER(1),IN(1,1))
EQUIVALENCE (ALUES(1),NLK)
MLV=8
NL=100
NU=40
NUV=8
NUOUT=8
NUIN=8
NFLT=16
NLK=20
NFLP=6
NUDIP=8
NUDDP=3
30
DELTA=10.0
NFLT=24

C NUTD = NUMBER OF TIME DELAYS POSSIBLE FOR EACH UNIT
C NLV = NUMBER OF VALUES PER LINK
C NL = NUMBER OF LINKS
C NU = NUMBER OF UNITS
C NUV = NUMBER OF UNIT STATE VARIABLES
C NUTDP = NUMBER OF PARAMETERS FOR A PROB DISTRIBUTION
C NUIN = MAXIMUM NUMBER OF INPUTS PER UNIT
C NOUT = MAXIMUM NUMBER OF OUTPUTS PER UNIT
C NFLT = MAX NUMBER OF FAULTS FOR EACH UNIT
C NFLTP = 2*(MAX NUMBER OF PARAMS FOR EACH FAULT DISTRIBUTION)

C INITIALIZE ALL TABLES

DO 5100 I=1,NL
C LINK TABLES
DO 5105 J=1,NLK
LK(J,I)=O
DO 5110 J=1,NLV
LV(J,I)=O
5110 CONTINUE
DO 5115 I=1,NU
C UNIT TABLES
UT(I)=O
DO 5120 J=1,NUIN
UIN(J,I)=O
DO 5122 J=1,NUOUT
UTDD(J,I)=O
DO 5127 K=1,NUTDP
UTDP(K,J,I)=O
5122 CONTINUE
DO 5125 J=1,NUV
UV(J,I)=O
C FAULT TABLES
DO 5121 J=1,NFLT
FAULT(J,I)=O
FLTD(1,J,I)=O
FLTD(2,J,I)=O
FLTD(3,J,I)=O
DO 5121 K=1,NFLTP
FLTP(K,J,I)=O
5115 CONTINUE

C ASSIGN I/O PORTS FOR RUN
READ(5,5130)SYSIN,SYSOUT,DATOUT
5130 FORMAT(314)

READ AND PRINT TITLE
READ(SYSIN,5150)TITLE
5150 FORMAT(20A4)
WRITE(SYSOUT,5151)TITLE
5151 FORMAT(31H,THE LOWER/UPPER TIME LIMITS ARE ,E12.6,3H/,,E12.6//)

C READ TIME BOUNDARIES, NUMBER OF MISSIONS, RANDOM SEED, PRINT CONTROL
C
READ(SYSIN,5260)TIME,LIMIT,NMSSN,IPRINT,ISEED
5260 FORMAT(2E12.6,18,512,130)
WRITE(SYSOUT,5261)TIME,LIMIT
5261 FORMAT(2H,THE LOWER/UPPER TIME LIMITS ARE ,E12.6,3H/,,E12.6)

C READ TIME BOUNDARIES, NUMBER OF MISSIONS, RANDOM SEED, PRINT CONTROL
C
READ(SYSIN,5260)TIME,LIMIT,NMSSN,IPRINT,ISEED
5260 FORMAT(2E12.6,18,512,130)
WRITE(SYSOUT,5261)TIME,LIMIT
5261 FORMAT(2H,THE LOWER/UPPER TIME LIMITS ARE ,E12.6,3H/,,E12.6)

C READ TIME BOUNDARIES, NUMBER OF MISSIONS, RANDOM SEED, PRINT CONTROL
C
READ(SYSIN,5260)TIME,LIMIT,NMSSN,IPRINT,ISEED
5260 FORMAT(2E12.6,18,512,130)
WRITE(SYSOUT,5261)TIME,LIMIT
5261 FORMAT(2H,THE LOWER/UPPER TIME LIMITS ARE ,E12.6,3H/,,E12.6)
C READ NUMBER OF VALUES FOR EACH LINK
C
5155 READ(SYSIN,5160)I,J,(LV(K,I),K=1,J)
5160 FORMAT(2I4,6F12.6)
5165 CONTINUE
C READ SPECIFICATIONS FOR EACH UNIT
C
5175 READ(SYSIN,5180)UNIT,J,K,L,M,N
5180 FORMAT(6I4)
5185 CONTINUE
C UNIT TYPE
C
5194 READ(SYSIN,5195)(UIN(J,UNIT),J=1,K)
5195 FORMAT(2I4)
5196 CONTINUE
C LINKS TO UNIT INPUTS
C
5205 FORMAT(3I4,3F12.6)
5206 CONTINUE
C LINKS TO UNIT OUTPUTS
C
5215 READ(SYSIN,5205)J,UOUT(J,UNIT),UTDD(J,UNIT),(UTDP(KK,J,UNIT),KK=1,3)
5220 CONTINUE
C
C  
C FAULTS PRESPECIFIED  
C  
495. IF (M) 5415, 5415, 5405  
496. DO 5412 JJ = 1, M  
497. READ(SYSIN, 5400) JJ, KK, LL, MM, MLL  
498.  
499. IF(JJ) 5415, 5415, 5402  
500. FLTD(1, JJ, UNIT) = KK  
501. FLTD(2, JJ, UNIT) = LL  
502. IF((KK.LE.1).AND.(LL.LE.1)) GO TO 5406  
503. MLL = 1  
504. MM = MINO(1, MM)  
505.  
506. FLTD(3, JJ, UNIT) = MLL  
507. FAULT(JJ, UNIT) = MM  
508. READ(SYSIN, 5403) (FLTP(I, JJ, UNIT), I = 1, 6)  
509. IF(IPRINT(4).EQ.O) GO TO 5408  
510. WRITE(SYSOUT, 5401) JJ, MM, KL, KK, (FLTP(I, JJ, UNIT), I = 1, 6)  
511.  
512. CONTINUE  
513. CONTINUE  
514. CONTINUE  
515. CONTINUE  
516.  
C STATE VARIABLES  
C  
517. IF(N) 5217, 5217, 5214  
518. READ(SYSIN, 5215) (UV(J, UNIT), J = 1, N)  
519. IF(IPRINT(4).EQ.O) GO TO 5217  
520. WRITE(SYSOUT, 5216) (J, UV(J, UNIT), J = 1, N)  
521. CONTINUE  
522. GO TO 5175  
523. CONTINUE  
524. CONTINUE  
525. CONTINUE  
526.  
C READ PRESPECIFIED EVENTS  
C  
527. IF(N) 5255, 5255, 5214  
528. READ(SYSIN, 5220) T, J, K, L, M  
529. IF(J) 5225, 5225, 5230  
530. IF(IPRINT(4).EQ.O) GO TO 5222  
531. WRITE(SYSOUT, 5221) (J, T, K, L, M)  
532. CONTINUE  
533. GO TO 5245  
534. CONTINUE  
535. CONTINUE  
536. CONTINUE  
537. CONTINUE  
538. CONTINUE  
539. CONTINUE  
540. CONTINUE  
541. CONTINUE  
542. CONTINUE  
543. CONTINUE  
544. CONTINUE  
545. CONTINUE  
546. CONTINUE  
547. JJ = T  
548. GO TO (5235, 5244, 5240, 5243), J  
549. L = LK(1, K)  
550. READ(SYSIN, 5236) (VAL(J), J = 1, L)  
551. IF(J) 5237, 5237, 5237  
552. IF(IPRINT(4).EQ.O) GO TO 5245  
553. WRITE(SYSOUT, 5237) (J, VAL(J), J = 1, L)  
554. CONTINUE  
555. GO TO 5245  
556. IF(IPRINT(4).EQ.O) GO TO 5239  
557. WRITE(SYSOUT, 5241) L  
558. CONTINUE  
559. CONTINUE  
560. VAL(1) = FLOAT(L)  
561. L = 1  
562. GO TO 5245  
563. VAL(1) = FLOAT(M)  
564. VAL(2) = FLOAT(L)  
565. IF(IPRINT(4).EQ.O) GO TO 5247  
566. WRITE(SYSOUT, 5246) L, M  

33
5246 FORMAT(15H FAULT CLASS ,14,12H GETS VALUE ,14)
5247   L=2
5248   GO TO 5245
5249   L=1
5250   VAL(1)=0.0
5251   GO TO 5245
5252
C
C INITIALIZE NEXT EVENT TABLE
5253 IF(INSERT(T,JJ,K,VAL, L)) 5250, 5250, 5255
5254
C
5255 WRITE(SYSOUT,5251)
5256
5257 FORMAT(17H EVENT TABLE FULL)
5258
C STOP
5259
C 5225 CONTINUE
5260
C
5261 C CONSTRUCT LINK CONNECT TABLES
5262
C
5263 DO 5270 UNIT=1,NU
5264   I=0
5265
5266 5275 I=I+1
5267   L=UIN(I,UNIT)
5268   IF(L)5270,5270,5285
5269
5270 LK(2,L)=L
5271 GO TO 5275
5272
5275 CONTINUE
5276
5277 DO 5290 L=1,NL
5278 LK(2,L)=0
5279 DO 5300 UNIT=1,NU
5280 I=0
5281
5305 I=I+1
5282   L=UOUT(I,UNIT)
5283   IF(L)5300,5300,5310
5284
5300 LK(2,L)=UNIT
5285 GO TO 5305
5286
5305 CONTINUE
5287
C
C CONSTRUCT LINK CONNECT TABLES
5288
C
5289 WRITE(4)ABLES,ANET,AUSER,ALUES
5290
RETURN
5291
END
5292
SUBROUTINE EDUMP
5293 INTEGER EORDER, ETYPE, ELORU, EHEAD, ELNGTH, ENVAL
5294 COMMON /NET/ET(300),ETYPE(300),ELORU(300),EVAL(16,300)
5295 COMMON/USER/ IN(8,8),OUTT(8,8),UV(8,40),UDT(8),FAULT(24,40)
5296 INTEGER EORDER,GIVES TRUE SEQUENCE OF THE EVENT ENTRIES.
5297 COMMON/USER/ TIME,LIMIT,TITLE(20),SYSIN,SYSOUT,DATOUT
5298 2 ,MISSION,NMSSN,ISEED,IPRINT(5),UNIT
5299 WRITE(SYSOUT,1000)
5300
1000 FORMAT(1H0,15H CURRENT TIME = , E12.6)
5301 1 /8H EVENT LIST ENTRIES WAITING TO BE PROCESSED
5302 1 /8x, 4HTIME, 3X, 4HTYPE, 5H L/U , 4X, 6HVALUES)
5303 J = EHEAD
5304 DO 500 I= 1, ELNGTH
5305
J = EHEAD
5306 DO 500 I= 1, ELNGTH
5307
J = EHEAD
5308 WRITE (SYSOUT, 2000) J, ET(J), ETYPE(J), ELORU(J)
5309 1 ,(EVAL(K, J), K = 1, KK)
5310 2000 FORMAT(14, E12.6, 13, 13,2X, E12.6)
5311 J = EORDER(J)
5312 500 CONTINUE
5313 RETURN
5314 END
INTEGER FUNCTION ADDU(IU)
COMMON /TABLES/ UI8(8,40),UOUT(8,40),UT(40),UTDD(8,40)
                ,UTDP(3,8,40),VAL(16),LK(20,100),LV(8,100),
                FLTD(3,24,40),FLTP(6,24,40),EUAFFC(40),PFT(3)
                ,OUTN(8)
REAL UTDP,VAL,LV,FLTP,PFT
INTEGER UI8,UOUT,UT,UTDD,LK,FLTD,EUAFFC,OUTN
COMMON /VALUES/ NLK,NUAFFC,NU,UOUT,UTD,NUV,NFLT,NLV,NU,UTD
                NUAFFC(3,24,40),FLTP(6,24,40),EUAFFC(40),PFT(3)
                OUTN(8)
REAL UTDP,VAL,LV,FLTP,PFT
INTEGER UI8,UOUT,UT,UTDD,LK,FLTD,EUAFFC,OUTN
COt4MON
/VALUES/
NLK,NUAFFC,NU,UOUT,UTD,NUV,NFLT,NLV,NU,UTD
                NUAFFC(3,24,40),FLTP(6,24,40),EUAFFC(40),PFT(3)
                OUTN(8)
INTEGER UIN,UOUT,UT,UTDD,LK,FLTD,EUAFFC,OUTN
COt4MON
/VALUES/
NLK,NUAFFC,NU,UOUT,UTD,NUV,NFLT,NLV,NU,UTD
                NUAFFC(3,24,40),FLTP(6,24,40),EUAFFC(40),PFT(3)
                OUTN(8)
IF(NUAFFC.EQ.0) GO TO 900
DO 10 I = 1,NUAFFC
     IF(IU.EQ.EUAFFC(I)) GO TO 100
10 CONTINUE
900 CONTINUE
NUAFFC = NUAFFC + 1
IF (NUAFFC .LE. NU) GOTO 20
ADDU = -1
RETURN
20 CONTINUE
EUAFFC(NUAFFC) = IU
ADDU = NUAFFC
RETURN
950 CONTINUE
IF(TM.GT.LIMIT) RETURN
C ERROR - MORE UNITS TO BE PROCESSED THAN THERE ARE UNITS IN THE SYSTEM
RETURN
20 CONTINUE
EUAFFC(NUAFFC) = IU
ADDU = NUAFFC
RETURN
C DUPLICATE ENTRY
100 ADDU=1
RETURN
END

INTEGER FUNCTION INSERT(TM,TYPE,LORU,VAL,NVL)
 INTEGER TYPE
 REAL VAL(16)
 COMMON/USER/
IN(8,8),OUTT(8,8),UV(8,40),UTD(8),FAULT(24,40)
TIME,LIMIT,TITLE(20),SYSIN,SYSOUT,DATOUT
REAL IM,OUTT,UV,UTD,TIME,LIMIT,TITLE
INTEGER FAULT,SYSIN,SYSOUT,DATOUT,MSSION,MSSN,ISEED,IPRINT,UNIT
COMMON /NET/ET(300),ETYPE(300),ELORU(300),EVAL(16,300)
NEVT,ENVAL(300),EORDER(300),EHEAD,ELNGTH,NEVT
                ET,ETYPE,ELORU,
                EVAL ARE PARALLEL TABLES WHICH HOLD EVENT
                TIME,TYPE,LINKORUNIT,
                AND VALUES RESPECTIVELY.
                EORDER GIVES TRUE SEQUENCE OF THE EVENT ENTRIES.
                INSERT=1
                IF(TM.GT.LIMIT) RETURN
                C FIND AN EMPTY SLOT IN THE EVENT TABLES FOR THIS ENTRY (INDEX WILL BE K )
                DO 200 I=1,NEVT
                IF (EORDER(I) .NE. 0) GOTO 200
                K=I
                GOTO 900
                200 CONTINUE
                INSERT = -2
                RETURN
                C ERROR -2 MEANS THE EVENT TABLE IS FULL
                900 CONTINUE
                IF(TM .GT.ET(EHEAD)) GO TO 950
                C INSERT THIS EVENT BEFORE THE HEADOF THE LIST
                EORDER(K) = EHEAD
                EHEAD = K
                GOTO 2000
                950 CONTINUE
                C START AT HEAD OF LIST AND FIND THE FIRST EVENT TIME .GT. TIME
                I PLC = EHEAD
                IC = 0
C CBREAK The LIST LINK AND PatCh IN INoEx k
  EORDER(IPREV) = k
  EORDER(K) = IPLC
2000 CONTINUE
  EORDER(IPREV) = K
  EORDER(K) = IPLC
  EORDER(IPREV) = K
  EORDER(K) = IPLC
2000 CONTINUE
  EORDER(IPREV) = K
  EORDER(K) = IPLC
  610 IF(IPREV.EQ.IPRES)RETURN
  IF((ELORU(IPRES).NE.LORU).OR.(ETYPE(IPRES).NE.ETYPE(IPLC)))GO TO 600
  IC=EORDER(IPRES)
  EORDER(IPRES)=IC
  IF(IC.EQ.IPLC)
    EORDER(IPRES)=IPRES
  EORDER(IPLC)=O
  ELNGTH=ELNGTH-1
  IPLC=IPRES
600 IPLC =K
320 CONTINUE
C C THE LAST ELEMENT WILL POINT TO ITSELF
C EORDER(I) = 0 MEANS THE ENTRY IS EMPTY
9000 GO TO 1500
END

REAL FUNCTION FLTGEN(IU,IF,I)
C FLTGEN GENERATES NEW FAULT CHANGE EVENT, INSERTS IF I.GE.1
COMMON/USER/ IN(8,8),OUTT(8,8),UV(8,40),UTD(8),FAULT(24,40)
  1 TIME,LIMIT,TITLE(20),SYSIN,SYOUT,DATOUT
2 MSSION,NMSSN,ISEED,IPRINT(5),UNIT
3 REAL IN,OUTT,UV,UTD,TIME,LIMIT,TITLE
4 INTEGER FAULT,SYIN,SYOUT,DATOUT,MSSION,NMSSN,ISEED,IPRINT,UNIT
5 COMMON /VALUES/ NLK,NUAFCC,NU,NUOUT,NUTDP,NUV,NUFL,NLV,NU TD
6 1 NUIN,NVAL,NUFTP,NUTDD
7 COMMON /TABLES/ UIN(8,40),OUTT(8,40),UT(40),UTDD(8,40)
8 1 UTD(3,8,40),VAL(16),LU(20,100),LV(8,100),
9 2 FLTD(3,24,40),FLTP(6,24,40),EUAFFC(40),PFT(3)
10 3 OUTN(8)
11 REAL UTD,VAL,LU,FLTP,PFT
12 INTEGER UIN,OUTT,UT,UTDD,LU,FLTD,EUAFPC,OUTN
13 TEMP1=-1.0
14 IM=FAULT(IF,IU)
15 M=FLTD(3,IF,IU)
C M IS MULTIPLICITY OF FAULT
C TEST IF GENERATE FAULT RECOVERY EVENT

765. IF(IM.EQ.0) GO TO 133
766. ID=FLTD(2,IF,IU)
767. IF(ID.EQ.0) GO TO 133
768. PFT(1)=FLTP(4,IF,IU)*FLOAT(IM)
769. PFT(2)=FLTP(5,IF,IU)
770. PFT(3)=FLTP(6,IF,IU)
771. TEMP1=FTIME(TIME,ID,PFT)
772. VAL(1)=FLOAT(IM-1)
773. C TEST IF GENERATE FAULT OCCURRENCE EVENT
774. IF(IM.EQ.M) GO TO 135
775. ID=FLTD(1,IF,IU)
776. IF(ID.EQ.0) GO TO 135
777. PFT(1)=FLTP(1,IF,IU)*FLOAT(M-IM)
778. PFT(2)=FLTP(2,IF,IU)
779. PFT(3)=FLTP(3,IF,IU)
780. TEMP2=FTIME(TIME,ID,PFT)
781. C COMPARE TIMES TO CHOOSE SMALLER POSITIVE
782. IF(TEMP1.LT.0.0) GO TO 137
783. IF(TEMP1.LT.TEMP2) GO TO 135
784. 135 IF(TEMP1.LT.0.0) GO TO 131
785. 137 TEMP1=TEMP2
786. VAL(1)=FLOAT(IM+1)
787. 135 IF(TEMP1.LT.0.0) GO TO 131
788. IF(I.LT.1) GO TO 131
789. C INSERT EVENT INTO NEXT EVENT LIST
790. VAL(2)=FLOAT(IF)
791. IF((INSERT(TIME+TEMP1,2,IF,VAL,2).GT.O) GO TO 131
792. STOP
793. 131 FLTGEN=TEMP1+TIME
794. RETURN
795. END

REAL FUNCTION FTIME(TM,ID,FP)
796. C UPD AND (TIME, LIMIT, TITLE, ISEED, IPRINT, UNIT)
797. COMMON/USER/ IN(8,8), OUTT(8,8), UV(8,8), UTD(8), FAULT(24,40)
798. 1 TIME, LIMIT, TITLE(20), SYSIN, SYSOUT, DATOUT
799. 2 MSSION, NMSSN, ISEED, IPRINT, JJNIT
800. REAL IN, OUTT, UV, UTD, TIME, LIMIT, TITLE
801. INTEGER FAULT, SYSIN, SYSOUT, DATOUT, MSSION, NMSSN, ISEED, IPRINT, UNIT
802. REAL FP(3)
803. GO TO (100,200,300,400,500,600,700,800,900,1000)
804. C EXPONENTIAL
805. 100 FTIME=-ALOG(URAND(1))/FP(1)
806. RETURN
807. C GAMMA
808. 200 STOP
809. C BETA STOP
810. 300 STOP
811. C BINOMIAL STOP
812. 400 STOP
813. C WEIBULL
814. 500 FTIME=EXP(ALOG(-ALOG(URAND(1))/FP(1))/FP(2))
815. RETURN
816. C CONSTANT
817. 600 FTIME=FP(1)
818. RETURN
819. C NORMAL (GAUSSIAN)
820. 700 U=2.*URAND(1)-1.
821. V=2.*URAND(1)-1.
822. S=U*U+V*V
823. IF(S.GT.1.) GO TO 700
824. FTIME=U*SQRT(-FP(2)*2.*ALOG(S)/S)+FP(1)
825. IF((FTIME.LT.0.0) GO TO 700
826. RETURN
827. C UNIFORM
828. 800 FTIME=FP(1)+(FP(2)-FP(1))°URAND(1)
829. RETURN
830. C PASCAL
831. 900 FTIME=FP(2)* FLOAT(1+IFIX(ALOG(1-URAND(1))/ALOG(1+FP(1))))
832. RETURN
833. END

37
**REAL FUNCTION URAND(IZZZ)**

**COMMON USER/ IN(8,8),OUTT(8,8),UV(8,40),UTD(8),FAULT(24,40)**

**TIME, LIMIT, TITLE(20), SYSIN, SYSOUT, DATOUT**

**I, MSSSN, ISeed, IPRINT(5), UNIT**

**REAL IN, OUTT, UV, UTD, TIME, LIMIT, TITLE**

**INTEGER FAULT, SYSIN, SYSOUT, DATOUT, MISSION, NMSSN, ISeed, IPRINT, UNIT**

**C URAND RETURNS UNIFORM RANDOM REAL VALUE IN RANGE (0,1)**

**INTEGER IA, IC, ITWO, M2, M**

**DOUBLE PRECISION HALFM, DATAN, DSQRT**

**DATA M2/O/, ITWO*10/2/**

**IF(M2.NE.0) GO TO 20**

**M=1**

**10 M2=M**

**M=ITWO*M2**

**IF(M.GT.M2) GO TO 10**

**HALFM=M2**

**IA=8*INT(HALFM*DATAN(1.D0)/8.D0)+5**

**IC=2*INT(HALFM*(0.5DO-DSQRT(3.D0)/6.D0)+1)**

**S=0.5/HALFM**

**20 ISEED=ISEED*IA+IC**

**C INCLUDE NEXT STATEMENT FOR COMPUTERS WHERE WORD LENGTH FOR ADDITION IS GREATER THAN FOR MULTIPLICATION**

**IF (ISEED/2.GT.M2) ISEED=(ISEED-M2)-M2**

**IF (ISEED.LE.0) ISEED=(ISEED+M2)-M2**

**URAND=FLOAT(ISEED)**

**RETURN**

**END**

**SUBROUTINE MSTOP(TM)**

**C INSERT A STOP EVENT AT TIME=TM**

**COMMON /TABLES/ UIN(8,40), UOUT(8,40), UT(40), UTD(8,40)**

**UV(8,40), UTDP(3,8,40), VAL(16), LK(20,100), LV(8,100)**

**FLD(3,24,40), FLTP(6,24,40), EUAFPC(40), PFT(3)**

**REAL UTDP, VAL, LV, FLTD, EUAFPC, OUTN**

**INTEGER UIN, UOUT, UT, UTD, LK, FLTD, EUAFPC, OUTN**

**VAL(1)=0.0**

**I=INSERT(TM,4,0,VAL,0)**

**RETURN**

**END**

**SUBROUTINE RSTOP**

**C FORCE RUN TO STOP AFTER THIS MISSION**

**COMMON USER/ IN(8,8), OUTT(8,8), UV(8,40), UTD(8), FAULT(24,40)**

**TIME, LIMIT, TITLE(20), SYSIN, SYSOUT, DATOUT**

**MISSION, NMSSN, ISeed, IPRINT(5), UNIT**

**REAL IN, OUTT, UV, UTD, TIME, LIMIT, TITLE**

**INTEGER FAULT, SYSIN, SYSOUT, DATOUT, MISSION, NMSSN, ISeed, IPRINT, UNIT**

**NMSSN=MISSION**

**RETURN**

**END**

**** END OF LISTING ****
References
