SHARE PROGRAM LIBRARY AGENCY

PROGRAM NUMBER

151008

University of Miami

1365 MEMORIAL DRIVE - CORAL GABLES, FLORIDA
(305) - 284-6257

SHARE PROGRAM LIBRARY SUBMITTAL FORM

SHARE PROGRAM LIBRARY AGENCY
Triangle Universities Computation Center
Post Office Box 12076
Research Triangle Park, North Carolina
27709 USA

SPLA CONTROL NUMBER: 202

This form should be completed and submitted with the program package to the SHARE Program Library Agency at the address shown above. Standards and instructions for submitting programs are in the "SHARE Reference Manual".

(1) Program Number (to be filled in by SPLA)..... 360D-15.1.008

(2) System Type (machine)..... S/36Ø, S/37Ø

(3) Search Key..... SOL-37Ø SIMULATION-SYSTEM

(4) Programming Systems/Languages..... OS/PL/1

(5) Author's Name and Address..... HORST E. ULFERS
DCEC, Code R83Ø
186Ø Wiehle Avenue
Reston, Virginia 22Ø9Ø

(6) Direct Technical Inquiries to Name & Address (if different than Author)

(7) Title of Program..... SOL-37Ø Simulation System

(8) Submitter's Installation Membership Code..... DCEC

(9) Submitter's Own Program Identification and Suffix(Optional).. U2MKØ.P1591.SOL37Ø

(10) Primary Subject Code..... 11 Ø

(11) Minimum System Requirements OS36Ø/OS37Ø (TSO OPTIONAL)

(12) New or Revision Code (if revision, show prior Program Number in Item 1) N

(13) Year Completed..... 1976

(14) Date of Submittal..... 6/30/76

(15) Documentation (number of original pages submitted)..... 56

(16) Abstract (should contain sufficient information for a reader to determine the value of the program). Listed on the reverse side of this form are subjects which may serve as a guide for a descriptive abstract.

Revised 4/74

SHARE PROGRAM LIBRARY SUBMITTAL FORM

Subject Guide:

a. **Purpose**
b. **Programming Language used**
c. **Version and modification level or release number**
d. **Field of application**
e. **Type of routine (main program, subroutine, etc.)**
f. **Specific description of machine requirements**

ABSTRACT

The SOL-370 Simulation System is a General Purpose Simulator for discrete modeling and simulation. The source language is English like and has been implemented as an extension to PL/1. The system produces object code and provides for extensive interactive post-simulation analysis.

The system is compatible with all versions of the PL/1-F, PL/1-Opt., and PL/1 - checkout compilers and can be used in the OS/MVT and OS/MVT-TSO environment. It can be operated in the batch or TSO mode.

Minimum system requirements for SOL-370, Release 1/76 are 200K of core, 250 tracks of 3330 disk or equivalent. In TSO mode minimum region is 170K.

The documentation consists of the "SOL-370 Language Reference Manual and a User's Guide," TN 25-75, and the "SOL-370 Installation and Error Tracing Guide," TN 23-76. Both documents are available also through the Defense Documentation Center (DDC).

DISCLAIMER

Triangle Universities Computation Center (TUCC) serves solely as the distribution agent for contributed programs and does not test or maintain them. They are distributed essentially in the original form submitted by the author. Neither TUCC nor SHARE, INC., makes any warranty, expressed or implied, as to the documentation, function, or performance of the contributed programs.

(Please attach additional pages if necessary).....Total pages attached

Permission to Publish

"I hereby give the SHARE Program Library Agency permission to reprint, reproduce, and distribute this program."

(17) **Signature of Submitter and Date** Signed & dated by letter attachment

(18) **Signature of Installation Addressee** _____

TECHNICAL NOTE NO. 25-75

SOL-370

LANGUAGE REFERENCE MANUAL AND USERS' GUIDE

December 1975

Prepared by:

Horst Ulfers

Approved for Publication:

ROBERT E. LYONS
Chief, Computer Systems Division

FOREWORD

The Defense Communications Engineering Center (DCEC) Technical Notes (TN's) are published to inform interested members ot the defense community regarding technical activities of the Center, completed and in progress. They are intended to stimulate thinking and encourage information exchange; but they do not represent an approved position or policy of DCEC, and should not be used as authoritative guidance for related planning and/or further action.

Comments or technical inquiries concerning this document are welcome, and should be directed to:

Director
Defense Communications Engineering Center
1860 Wiehle Avenue
Reston, Virginia, 22090

TABLE OF CONTENTS

I. INTRODUCTION

SOL-370 is a dialect of the SOL-simulation language as developed by D. E. Knuth and J. L. McNeley (ref 1). The original SOL language has been extended to accomodate algorithms essential for the simulation of communications networks and systems. Furthermore, SOL-370 has been implemented as an extension of the PL/I language. This allows for a free intermixing of SOL-370 and PL/I language statements.

SOL-370 is an algorithmic language used to construct models of general systems for simulation in a readable form. The model builder describes his model in terms of PROCESSES whose number and detail are completely arbitrary and definable within the constraints of the language elements. A SOL model consists of a number of statements and declarations which have a character similar to that found in programming languages such as PL/I or ALGOL.

The model is not built to be executed in a sequential fashion, as ordinary programming languages require. Rather, the processes are written and executed as though all were running in parallel. Control between processes is maintained by the interaction of GLOBAL ENTITIES and by control and communications instructions within the different processes. At the initiation of the simulation all processes are begun simultaneously.

Variables declared within a process are called LOCAL VARIABLES. Within a given process it is possible to have several actions occurring at once; therefore, to visualize the process, we may think of several objects on which the action takes place, each in its own place in the process at any given time. These objects will be referred to as TRANSACTIONS. A set of local variables corresponding in number to those declared in the process is "carried with" each transaction of that process. Transactions situated within one process may not refer to the local variables of another process nor to the local variables of another transaction in the same process.

GLOBAL ENTITIES are of four major types: GLOBAL VARIABLES, FACILITIES, STORES, and TRUNKS. Global variables can be referenced or changed by any transaction from any process in the system, and the variable possesses only one value at any given time.

II. SOL SYNTAX

1. THE SYNTAX NOTATION

The Backus Normal Form (BNF) is used to describe the syntax of the SOL language. The following rules explain the use of this notation.

a. The Notation Variable

A Notation Variable is the name for a class of elements used in a programming language. It consists of letters and hyphens and is enclosed in "less than" and "greater than" symbols.

EXAMPLES:

<digit> This denotes the occurrence of a digit, which may assume values of 0 through 9.

<facility name> This denotes the occurrence of a Notation Variable of the class Facility Name.

<do statement> This denotes the occurrence of a DO statement.

b. The Notation Constant

A Notation Constant is the literal occurrence of a string of characters. It is represented in capital letters.

Example:

STORE This denotes the literal occurrence of the word "STORE".

c. The Syntactical Unit

A Syntactical Unit is defined as a single variable, a constant, or any collection of notation variables, notation constants, and syntax language symbols. The vertical stroke "|" separating two Syntactical Units indicates a choice, which can be made between the two. Anything enclosed in brackets denotes an option. The syntax within the brackets may be used or left out.

EXAMPLES:

<identifier> FIXED|FLOAT	This denotes an identifier which may have the attribute FIXED or FLOAT.
<identifier>[(<number>)]	This denotes an identifier which may optionally be subscripted by a number.

d. General Format

The general format of a syntax statement uses the definition symbol "::=" to define a notation variable.

EXAMPLE:

<go to statement> ::= GO TO <label>;

2. THE MODEL STRUCTURE

When coding a SOL model the format below should be followed:

```
<global variable declarations>
<resource declarations>          >GLOBAL DECLARATIONS
<table declarations>

<process declaration>
<process statements>             > FIRST PROCESS
<end statement>                                    > MODEL

<process declaration>
<process statements>             > SECOND PROCESS
<end statement>
  .  .  .  .
  .  .  .  .
<end statement>
```

First, all global declarations should be listed. The order of the global declarations is arbitrary. The order in which processes are listed should be selected carefully, since the first process will be started first and if any values are read for global variable initialization, this should be done in the first process. The program should be followed by an END statement. If not included, the system will provide this statement.

a. Process Declaration

```
<process description>::=PROCESS <identifier>
                        [,T=<number>] [,R=<number>];
```

Each process is bracketed with the process declaration and an END statement. The two optional parameters allow the modeler to specify the maximum number of simultaneous transactions T and the maximum number of resources R (facilities, stores, and trunks) encountered by any one transaction. This feature is important in optimizing the model's core requirements.

EXAMPLE:

```
PROCESS SWITCH, T=100, R=4;
```

b. Variable Declaration

Variables are declared either at the beginning of the program outside the processes as GLOBAL VARIABLES, or at the beginning of each process as LOCAL VARIABLES.

<global variable declaration>:=<variable declaration>

<variable declaration>:=INTEGER <identifier list> ;
| REAL <identifier list>;

When declared INTEGER, their internal representation will be fixed binary; when declared REAL, floating decimal.

EXAMPLES:

```
INTEGER A, B, C, D;
REAL X, Y, Z;
INTEGER AR(16);
INTEGER BAR(0:100); %
```

c. Resource Declarations

<resource declaration> ::= <facility declaration>;
| <store declaration>;
| <trunk declaration>;

All resources must be declared ahead of the process declarations. For details see the discussion for the specific resource.

3. IDENTIFIERS AND CONSTANTS

<letter>::=A|B|C|D|...|Z
<digit>::=0|1|2|3|...|9
<number>::=<constant>|<decimal constant>
<constant>::=*<digit>*
<decimal constant>::=<constant>.<constant>
<identifier>::=<letter>|<identifier><letter>|
<letter><digit>

Identifiers are used as the names of variables, statistical tables, stores, facilities, processes, procedures, and statements. A specific identifier can be used for only one purpose in a program. Constants are used to represent integer numbers; decimal constants represent real numbers. Identifiers must be declared before they are used elsewhere. All SOL commands are reserved words and must not be used as identifiers. No identifier can start with the character '$'.

4. EXPRESSIONS AND RELATIONS

```
<name>::=<identifier>|<identifier>|<expression>| %
```

By variable name, facility name, etc., we will mean that the identifier in the name has appeared in a variable declaration, facility declaration, etc., respectively.

```
<primary>::=<variable name>|<store name>|
 <constant>|<decimal constant>|TIME|
 (*<expression>*)|abs(<expression>)|
 DISTRIBUTION((a)x,(b)y,...(c)z)|
 NORMAL(<expression>,<expression>)|
 EXPONENTIAL(<expression>)|poisson(<expression>)
 GEOMETRIC(<expression>)|RANDOM
<term>::=<primary>|<term>X<priimary>|
        <term> <primary>|<term>/<primary>|
        <term>MOD<primary>
<sum>::=<term>|+<term>|-<term>|<sum>+<term>|
        <sum>-<term>
<unconditional expression>::=<sum>|<sum>:<sum>
<expression> ::= <unconditional expression> |
              if <relation> THEN <expression> ELSE <expression>
```

The meaning of an arithmetical operations inside an expression is identical to the meaning in PL/I.

The new elements here are "a MOD b," the positive remainder obtained upon dividing a by b ; "MAX(e ,.....,e)" and "MIN(e ,...,e)," which denote the maximum and minimum values, respectively, of the n expressions; and there are also notations for expressing random values. The expression "(e ,...,e)" indicates that a random selection is made from among the n expressions with equal probability of choosing any expression. The expression DISTRIBUTION ((a)X,(b)Y,...(c)z)) can be used as a shorthand notation of "(e ,....,e)" as shown in the following example.

```
A=(1,1,1,1,2,2,4,4,4,9,9,9,9,9,9,9,9,9); can be written as
A=DISTRIBUTION((4)1,(2)2,(3)4,(9)9);
```

The expressions NORMAL (M, S), POISSON (M), GEOMETRIC (M), and EXPONENTIAL (M) indicate random values with special distributions which occur frequently in applications. A random number drawn from the normal distribution with mean M and standard deviation S is denoted by NORMAL (M, S) and is a real (not necessarily integer) value. A number drawn from the exponential distribution with mean M is denoted by EXPONENTIAL (M) and is also of type real. The poisson distribution signified by POISSON (M), on the other hand, yields only integer values; the probability that POISSON (M) = n is (e +M /n). The geometric distribution with mean M, denoted by GEOMETRIC (M), also yields integer values, where the probability that GEOMETRIC(M)=N-1 is 1/m(1-1/M). The

symbol RANDOM denotes a random real number between 0 and 1 having a uniform distribution. Finally,the notation a:b denotes a random integer between the limits a and b. The normal, exponential, poisson, and geometric distributions are mathematically expressible in terms of random distributions as follows:

```
NORMAL(M,S) = S * V -2 ln(random) * sin(2 *random) + M

EXPONENTIAL(M) = - M ln(random)

POISSON(M) = n if e  (1+M+M /2 +.....+M  /(n-1) )
             <= random < e   (1+M + ...  +M /n )

GEOMETRIC(M) = (1 + ln(random)/ln(1-1/M)).
```

(The poisson distribution should not be used for values of M greater than 10.) As examples of the use of these distributions, consider a population of customers coming to a market with an average of one customer every M minutes. The distribution of waiting time between successive arrivals is EXPONENTIAL(M). On the other hand, if an average of M customers come in per hour, the distribution of the actual number of customers arriving in a given hour is POISSON(M). If an individual performs an experiment repeatedly with a chance of success, I/M on each independent trial, the number of trials needed until he first succeeds is GEOMETRIC(M).

The special symbol "TIME" indicates the current time; initially, time is zero. The value of a store name is the current number of occupants of the store.

```
<relational operator> ::= = | ~= | <= | >= | > | <
<relation primary>::= <unconditional expression>
                <relational operator> <unconditional expression>|
                <facility name> BUSY |<facility name> NOT BUSY |
                <store name> FULL | <store name> NOT FULL |
                <store name> EMPTY | <store name> NOT EMPTY |
                 pr(<expression>) | (<relation>)
<relation> ::= <relation primary>|
                <relation primary> | <relation primary>|
                <relation primary> & <relation primary>|
                ~<relation primary>
```

These relations have obvious meanings except, for the construction "pr(e)", which stands for a random condition that is true with probability e. (Here e must be less than or equal to 1.)

```
IF PR(0.12) THEN (12% of the time)
 ELSE (88% of the time).
```

5. FACILITIES AND ASSOCIATED COMMANDS

A FACILITY is a global element which can be controlled by only one transaction at a time. Associated with each request for the facility is a "control strength," and if a requesting transaction has a higher strength than the transaction controlling the facility, an interrupt will occur. Interrupts may be nested to any depth. If the requesting transaction is not of greater strength than the controlling transaction, then the requesting transaction stops and waits for the facility until the controlling transaction releases its control.

The following declarations and commands are associated with facilities.

a. Facility Declaration

```
<facility declaration> ::= facility <facility name list>;
<facility name list>::=<facility name>[,<facility name list>]
<facility name>::= <name>[(<number>)]
```

Facilities are declared at the beginning of the program ahead of the processes. Facilities may be declared as one-dimensional arrays.

```
EXAMPLES:      FACILITY TERMINAL;
               FACILITY LINE (16);
```

b. SEIZE Statements

```
<seize statement>::= SEIZE <facility name>;|
                     SEIZE <facility name>, <expression>;
```

The first form is equivalent to "SEIZE <facility name>, 0." This statement is usually rather simple, but there are situations when complications arise. If the facility is not busy when this statement occurs, then it becomes busy at this point and remains busy until later released by this transaction. (Note: If this transaction creates another transaction, the new transaction does not control the facility.)

The <expression> in the SEIZE statement represents the "control strength" which is normally zero. Allowance is made, however, for one transaction to interrupt another. E.g. the facility is busy when the seize statement occurs, let CS be the control strength with which the facility was seized and let HS be the control strength of this seize statement. If HS <= CS , the transaction waits until the facility is not busy. If HS > CS , however, interrupt occurs. The preempted transaction is handled according to the last INTERRUPT statement executed. The transaction, A, which had control of the facility, is stopped wherever it was in its program, and the present transaction, B, seizes the facility. When B releases the facility, the following occurs:

(1) If A was executing a wait statement when interrupted the time of wait is increased by the time which passed during the interrupt.

(2) There may be several transactions waiting but not attempting to seize this facility. If any of these has a higher control strength than CS, then A is interrupted again. The transaction which interrupts is chosen by the normal rules for deciding who obtains control of a facility upon release, as described in the section for the RELEASE STATEMENT.

The control strength in the present implementation of SOL must be an integer between 0 and 15. This allows interrupts to be nested up to 15 deep.

EXAMPLES:

```
SEIZE TERMINAL, PRIORITY_CLASS:
SEIZE LINE, 10;
SEIZE PUMP;
SEIZE TERMINAL, 1:10;
SEIZE LINE, EXP(15);
SEIZE LINE, A*(B-C);
```

c. RELEASE Statements

<release statement>::=RELEASE <facility name>;

This statement is permitted only when the transaction is actually controlling the facility because of a previous seizure. When the facility is released, there may be several other transactions waiting because of seize statements. In this case, the one which gets control of the facility next is chosen by a consideration of the following three quantities in order:

(1) Highest control strength

(2) Highest PRIORITY

(3) First to request the facility.

EXAMPLES:

```
RELEASE TERMINAL;
RELEASE LINE;
RELEASE PUMP;
```

d. Testing the Status of a Facility

```
<facility status> = BUSY | NOT BUSY
```

The status of a facility can be tested for the condition BUSY or NOT BUSY. The facility status can be used in any compound statement as a relation primary.

EXAMPLES:

```
IF TERMINAL BUSY THEN CANCEL:
IF LINE NOT BUSY THEN GO TO LOAD:
WAIT UNTIL TERMINAL NOT BUSY:
```

6. STORES AND ASSOCIATED COMMANDS

STORES are space-shared rather than time-shared global elements and they are assigned a specific storage capacity. As long as there is sufficient storage to accommodate the requesting transaction the request for space is satisfied; otherwise, the transaction waits for the space regarded as a store which has a capacity of one unit only, except for the fact that no interrupt capability is provided for stores.

The following declarations and control statements are associated with manipulating stores:

a. STORE Declaration

```
<store declaration> ::=STORE <capacity> <store name list>;
<store name list>   ::= <store name>[,<store name list>]
<store name>        ::= <name>[(<number>)]
<capacity>          ::= <number>
```

STORES are declared at the beginning of the program ahead of the processes. STORES may be declared as one-dimensional arrays.

```
EXAMPLES:     STORE 10 STACK;
              STORE 512 (CORE, BUFFER(5));
```

b. ENTER Statement

```
<enter statement>::= ENTER <store name>;|
                     ENTER <store name>, <expression>;
```

The first form is an abbreviation for "ENTER <store name>, 1." The value of the expression, rounded to the nearest integer, gives the number of units requested of the store. The transaction will remain at this statement until that number of units becomes available and until all other transactions of greater or equal priority which have been waiting for storage space have been serviced.

EXAMPLES:

```
ENTER STACK;
ENTER CORE, 256;
ENTER CORE, BYTE * LENGTH;
```

c. LEAVE Statement

```
<leave statement> ::= LEAVE <store name> ; |
                      LEAVE <store name>,<expression>;
```

The first form is an abbreviation for "LEAVE <store name>, 1." This statement returns the number of units equivalent to the value of the (rounded) expression.

EXAMPLES:

```
LEAVE STACK:
LEAVE CORE, 128;
LEAVE BUFFER (NODE), LENGTH;
```

d. Testing The Status Of A Store

```
<store status> = FULL | NOT FULL | EMPTY | NOT EMPTY;
```

The status of a store can be tested for the following conditions

FULL, NOT FULL, EMPTY, NOT EMPTY.

In combination with other SOL or PL/I statements a variety of compound statements may result.

EXAMPLES:

```
IF SWITCH FULL THEN WAIT PAUSE:
IF SWITCH NOT FULL THEN ENTER SWITCH:
```

7. TRUNKS AND ASSOCIATED COMMANDS

TRUNKS are space-shared global elements similar to STORES. However, in contrast to stores, trunks allow for preemption. As long as there is sufficient storage to accommodate the requesting transaction, the request for space is satisfied without further action. Each transaction holding space in a trunk is assigned a specific holding strength, which may be different from the preemption strength. Thus, a transaction with a low preemption strength once assigned space in a trunk can have a very high holding strength; therefore preemption of it becomes unlikely.

a. TRUNK Declaration

```
<trunk declaration>::=TRUNK <capacity><trunk name list>;
<trunk name list>::= <trunk name>[,<trunk name list>]
<trunk name>::= <name>[(<number>)]
```

TRUNKS are declared at the beginning of the program ahead of the processes. TRUNKS may be declared as one-dimensional arrays.

EXAMPLES:

```
TRUNK 96 SWITCH(16)
TRUNK 1000 CORE;
```

b. DEMAND Statement

```
<demand statement>::= DEMAND <trunk name>, <capacity>,
                       <demand strength>, <hold strength>;
<trunk name>::= <name>[(<number>)] |<name>(<variable>)
```

The demand statement is a request for a number of units of a trunk. If the units requested are available in the trunk, they are assigned to the transaction. Associated with the resource allocation is the hold strength specified in the demand statement.

If the required number of units are not available, then the following takes place:

(1) The number of units needed through preemption are calculated.

(2) The sum of the space held by other transactions at a hold strength less than demand strength of the demanding transaction is determined.

(3) If the total available and preemptable space is sufficient to satisfy the demand, transactions are preempted as required to free enough space. The demanding transaction is then allocated the space with the associated hold strength and continues in sequence. Note that the interrupted transactions are handled according to the setting of the interrupt action indicator.

(4) If there is not enough preemptable space in the trunk, the transaction is queued up on demand strength.

EXAMPLES:

```
DEMAND LINK(15), TRANSM_RATE,  4, 10;
DEMAND CORE(NODE), 512, A, (A + C)/B;
```

c. YIELD Statement

<yield statement> ::= YIELD <trunk name> , <capacity>,
<hold strength>;
<trunk name> ::= <name>[(<number>)] |<name>(<variable>)

The yield statement releases the specified number of units in the trunk at the specified hold strength. If the number to be released is greater than the number currently held by the transaction at that hold strength, the simulation goes to error terminate.

EXAMPLES: YIELD LINES(5), 400, 10;

d. CAPACITY Function

<primary>::=CAPACITY (<trunk name>, <demand strength>)

The capacity function is provided to test the status of a trunk. The capacity function returns the number of units available for a demand of the capacity of the specified demand strength. No resources are allocated and the current state of the trunk is not touched. This function allows the simulation to interrogate the state of the trunk prior to attempting a demand statement upon it.

8. TRANSACTIONS AND ASSOCIATED STATEMENTS

Transactions represent discrete elements "flowing" through the model. They are local to a particular process and may have a number of descriptors (local variables). For example, in a road network simulation a transaction may represent individual vehicles. The properties of these vehicles, such as speed, number of passengers, fuel consumption, etc., are described by the local variables. Each transaction has its own set of local variables. The following statements directly control the creation, disappearance, queuing, or transfer of transactions.

a. Creation of Transactions.

At the beginning of simulation there is one transaction present for each process described. Each of these initial transactions starts at time zero and is positioned at the beginning of the process. More transactions may be created by using "start statements."

<start statement>::= NEW TRANSACTION TO <label>;

This statement, when executed, creates a new transaction (whose local variables are the same in number and value as those of the transaction which created it). The new transaction begins executing the program at label while the original transaction continues in sequence.

b. Disappearance Of Transactions

Transactions "die" when they execute a cancel statement.

<cancel statement>::=CANCEL:

An implied cancel statement is at the end of every process, so cancel statements need not always be explicitly written. Transactions are also cancelled when they are preempted and the global variable INTERRUPT has been set to CANCEL (see discussion of 'Interrupt').

c. Queuing Of Transactions

Whenever a transaction encounters a blocked resource such as a full store, a busy facility or a full trunk, it automatically enters the queue associated with this resource. Besides these situations the following wait conditions may be programmed for:

(1) WAIT Statements

<wait statement>::=WAIT <expression>;

The expression is rounded to the nearest integer, and then this statement advances "time" by MAX(0, <expression>, as far as this transaction is concerned. All time delays in a simulated process are, in the last analysis, specified by using wait statements.

EXAMPLES:

```
WAIT 400;
WAIT SIMULATION_TIME:
WAIT (A + B)/C;
```

(2) WAIT UNTIL Statements

<wait-until statement>::=WAIT UNTIL <relation>;

This causes the transaction to freeze at this point until the <relation> becomes true (because of action by other transactions). The relation must not involve expressions which have a random value; e.g., it is not legal to write "WAIT UNTIL pr(10)" or "WAIT UNTIL A|1:4|=0," etc.

EXAMPLES:

```
WAIT UNTIL SWITCH EMPTY;
WAIT UNTIL TIME  SIMULATION_TIME;
```

d. Transfer Of Transactions

(1) GO TO Statements

<go to statement> ::= GO TO <label>;

This statement is used to transfer to another point in the program; statements are usually executed sequentially.

(2) INTERRUPT Statement

INTERRUPT = WAIT;|CANCEL;|<label>;

INTERRUPT is a global variable which specifies the action to be taken for a preempted transaction. Whenever the interrupt variable has been set, the action for all subsequent preemptions any place within the program is specified unless the interrupt variable is reset.

INTERRUPT = WAIT;

If the interrupted transaction is executing a WAIT statement when interrupted, the wait time is increased by the time which passed during the interrupt. If the interrupted transaction was executing anything other than a WAIT, the transaction is cancelled.

INTERRUPT = CANCEL;

The interrupted transaction is unconditionally cancelled. (Refer to cancel statement).

INTERRUPT = <label>;

The interrupted transaction is started immediately at the statement specified by label and the transaction no longer controls the preempted facility.

9. SPECIAL SOL STATEMENTS

a. PRIORITY Statement

If by coincidence two transactions attempt to do something at precisely the same time, they may be in conflict: that is, they may both want to seize a facility, to change the value of the same global variable, or one may want to change it while the other is using its value. Actually, in such cases of conflict, the simulator does choose a specific order for execution; no two things actually happen at the same instant, as we deal more properly with infinitesimal differences of time between the discrete units. The choice of order is fairly arbitrary except when a difference of priority is specified; in that case, the transaction with higher priority will be acted on first. Each transaction has a priority, which is initially zero; priority is changed by the statement

PRIORITY = <expression>;

The declaration "integer PRIORITY" is implied at the beginning of each process; i.e., PRIORITY is treated as a local variable. In the present implementation of SOL, the priority must be between 0 and 15.

b. STOP Statement

<stop statement> ::= STOP;

A stop statement causes simulation to terminate immediately, and all transactions cease. The statistics for all stores, tables, and facilities are output as in the output statement, as well as the final time, the number of times each labeled statement was referenced, and the number of transactions which appeared in each process.

c. Transaction Input-Output Statements

<transaction read statement> ::= READ TO <label>;
<transaction write statement> ::= WRITE;

The read statement inputs a set of values of local variables for a transaction of the same type as the one executing the read statement. This set of values is used in the creation of a new transaction which begins executing the program at the statement mentioned. The write statement writes the current values of the local variables of the transaction onto the unit specified and does not cancel the present transaction. The output is associated with the file named 'DISK', while the input is read from file 'CARD'.

The tabulate statement in conjunction with the table declaration is the vehicle for collecting data to be displayed in histogram format.

(1) <table declaration> ::= TABLE(<MIN> BY <INC> TO <MAX>)
<table name>(<number>);

The table is used in conjunction with the TABULATE statement to collect data for histographical representation. The histogram can be specified by its dimensions, MIN, INC, MAX.

EXAMPLE: TABLE (0 BY 100 TO 1000) TIMETABLE(5) ;

(2) TABULATE Statements

<tabulate statement>::=TABULATE <expression> IN <table name>;

The value of the expression is recorded as a statistical observation in the table specified.

EXAMPLES:

TABULATE TIME IN TIMETABLE;

TABULATE (STARTIME-TIME) IN DIFFTABLE;

10. COMPOUND AND CONDITIONAL STATEMENTS

Both of those statements are legal in SOL as well as in PL/I. Because of their relative importance and frequent use, they are listed separately.

a. Compound Statements

Several statements may be combined into one, as follows:

<statement list>::=<statement>; [<statement list>];

<compound statement>::=begin <statement list> end;|
(<statement list>)

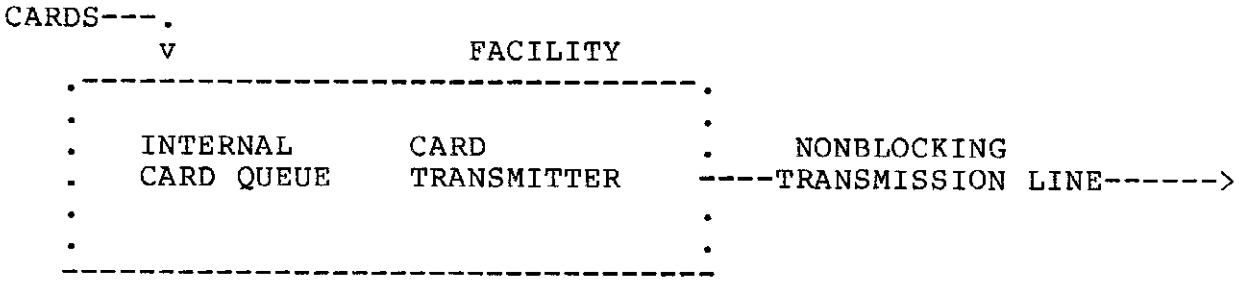
b. Conditional Statements

<condition>::= if <relation> then <unconditional statement>;|
if <relation> then <unconditional statement> else <statement>

III. SAMPLE MODELS

This section contains a description of nine simple models, including listings of the source language code. Most listings are self-descriptive. However, a more detailed description has been provided for MODEL 1E.

1. MODEL 1A: Single Server Queuing Model With ConstantAarrival Rate



PROBLEM: Punched cards are arriving at a card transmitter station with a constant interarrival interval of 36 seconds. The transmitter can handle only one card at a time and needs 40 seconds to process one card. The simulation is to stop after 5 minutes.

2. MODEL 1B: Single Server Queuing Model with Poisson Distributed Arrivals.

PROBLEM: As in Model 1A, except punched cards are arriving poisson distributed with an average arrival rate of 100 cards/minute.

3. MODEL 1C: Single Server Model with Parameterized Input.

PROBLEM: Same as in 1B except time constants are to be replaced by variables which are to be assigned values from a data set.

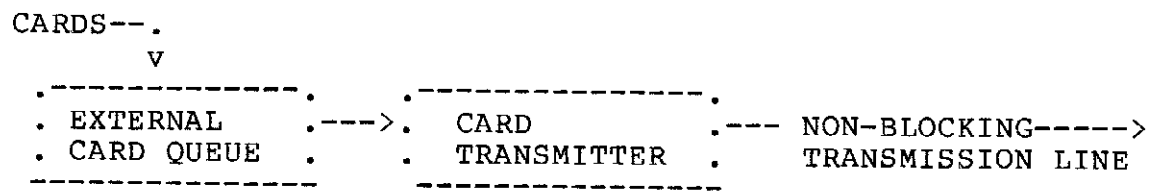
4. MODEL 1D: Single Server Queuing Model with Priority Handling.

PROBLEM: Same as in 1C, except cards are assigned priorities between 1 and 8 on a random basis. The transmitter is to select cards from the input queue according to its priority strength. A message is to be printed, when card is received.

5. MODEL 1E: Single Server Queuing Model with Preemption.

PROBLEM: Same as in ID, except preempt levels between 1 to 4 are to be assigned randomly. The preempted messages are to be cancelled after a message has been printed.

6. MODEL 1F: Single Server Queuing Model with External Queue



PROBLEM: Same as in 1E, except the STORE resource 'QUEUE' is used to model a physical queue ahead of the transmitter. This QUEUE is used to monitor the queue buildup during the simulation, since the internal queue of the facility is not accessible to the user. Furthermore, a separate process 'CONTROL' is used to read in variable values and control the length of the simulation.

7. MODEL 1G: Network Model - 5 Nodes Fully Connected, but Nonblocking Links.

PROBLEM: Each node is modeled as a combination of a card transmitter as described in 1G and a card receiver of a similar type. The network is fully connected and nonblocking. The originating nodes and the terminating nodes are picked at random. Each message is to be assigned an identification number.

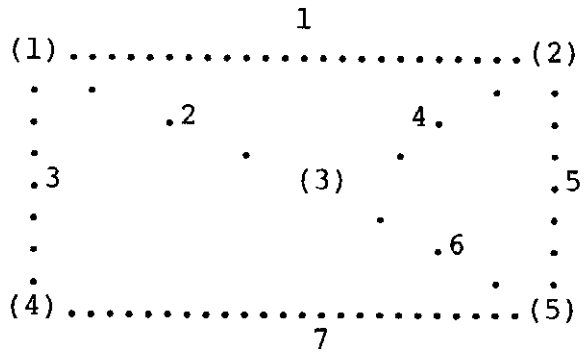
8. MODEL 1H: Network Model with Blocking Links.

PROBLEM: Same as in 1G except that blocking on links is considered. All links are to have eight channels. No alternate routing will be considered. The connecting matrix 'CONN' provides for cross-reference between nodes and links.

	1	2	3	4	5
1	0	1	2	3	0
2	1	0	4	0	5
3	2	4	0	0	6
4	3	0	0	0	7
5	0	5	6	7	0

9. MODEL 1I: Network Model with Alternate Routing.

PROBLEM: Same as in 1H. Network has the following connectivity (Links are numbered from 1 to 7):



The alternate routing will be of the following type:

Each node has a primary plus two alternate next nodes to choose from for routing a message. The selection will simply be based on the blocking of the links. The modeler may assume that the routing algorithm has been precoded as a function named 'ROUTE' with two arguments representing the current node and the destination node.

<variable>= ROUTE (<orig. node>,<term. node>);

The function will return the next tandem node number of the value '0' if blocking occurs.

```
/* MODEL1A  -  SINGLE SERVER QUEUING MODEL */
/*          WITH UNIFORMLY DISTRIBUTED ARRIVALS */
FACILITY TRANSMITTER;
PROCESS TRANSMIT, T=10, R=1;
START:  IF TIME > 3000 THEN STOP;
        NEW TRANSACTION TO SEND;
        WAIT 360;
        GO TO START;
SEND:   SEIZE TRANSMITTER;
        WAIT 400;
        CANCEL;
END;

/* MODEL1B  -  SINGLE SERVER QUEUING MODEL */
/*          WITH POISSON DISTRIBUTED ARRIVALS */
FACILITY TRANSMITTER;
PROCESS TRANSMIT, T=10, R=1;
START:  IF TIME > 3000 THEN STOP;
        WAIT EXPONENTIAL(360);
        NEW TRANSACTION TO START;
        SEIZE TRANSMITTER;
        WAIT 400;
        CANCEL;
END;

/* MODEL1C  -  SINGLE SERVER QUEUING MODEL */
/*          WITH PARAMETERIZED INPUT VARIABLES */
INTEGER SIMTIME,INTTIME,SERTIME;
FACILITY TRANSMITTER;
PROCESS TRANSMIT, T=10, R=1;
GET FILE(CARD) LIST(SIMTIME,INTTIME,SERTIME);
START:  IF TIME > SIMTIME THEN STOP;
        WAIT EXPONENTIAL(INTTIME);
        NEW TRANSACTION TO START;
        SEIZE TRANSMITTER;
        WAIT SERTIME;
        CANCEL;
END;

/* MODEL1D  -  SINGLE SERVER QUEUING MODEL */
/*                  WITH PRIORITY HANDLING  */
INTEGER SIMTIME,INTTIME,SERTIME;
FACILITY TRANSMITTER;
PROCESS TRANSMIT, T=10, R=1;
GET FILE(CARD) LIST(SIMTIME,INTTIME,SERTIME);
START:  IF TIME > SIMTIME THEN STOP;
        WAIT EXPONENTIAL(INTTIME);
        NEW TRANSACTION TO START;
        PRIORITY = 1:8;
        PLIBEGIN;
        PUT EDIT ('CARD RECEIVED AT ',TIME)(A(17),F(6)) SKIP;
        PLIEND;
        SEIZE TRANSMITTER;
        WAIT SERTIME;
        CANCEL;
END;
```

```
/* MODEL1E  -  SINGLE SERVER QUEUING MODEL */
/*                  WITH PREEMTION          */
INTEGER SIMTIME,INTTIME,SERTIME;
FACILITY TRANSMITTER;
PROCESS TRANSMIT, T=10, R=1;
INTEGER STRENGTH;
GET FILE(CARD) LIST(SIMTIME,INTTIME,SERTIME);
INTERRUPT = FINISH;
START:  IF TIME > SIMTIME THEN STOP;
        WAIT EXPONENTIAL(INTTIME);
        NEW TRANSACTION TO START;
        PRIORITY = 1:8;
        STRENGTH = 1:4;
PLIBEGIN;
        PUT EDIT ('CARD RECEIVED AT ',TIME)(A(17),F(6)) SKIP;
PLIEND;
        SEIZE TRANSMITTER,STRENGTH;
        WAIT SERTIME;
        CANCEL;


FINISH:
PLIBEGIN;
PUT EDIT('PREEMPTION OCCURRED AT ',TIME) (A(23),F(6)) SKIP;
PLIEND;
CANCEL;
END;
```

EXPLANATION TO MODEL 1E. CODE

```
 STATEMENTS 10, 20.  /* MODEL 1E - SINGLE SERVER QUEUING MODEL*/
                     /*                  WITH PREEMPTION     */
```

The first two statements contain explanatory text. Since it is bracketted with /* . . . . */ it will be ignored by the translator.

```
STATEMENT 30.        INTEGER SIMTIME, INTTIME, SERTIME;
```

This statement declares the global variables SIMTIME, INTTIME, and SERTIME. Within the model these variables will assume the following meaning.

SIMTIME = Simulation time, the time after which the simulation is to be terminated.

INTTIME = Interarrival time for transactions.

SERTIME = Server time, the time a transaction will seize a facility until it is served.

STATEMENT 40. FACILITY TRANSMITTER;

This statement declares one nonsubscribed facility with the name TRANSMITTER.

STATEMENT 50. PROCESS TRANSMIT, T=10, R=1;

This statement declares the process with the name TRANSMIT. T=10 specifies that not more than 10 transactions will be active at any one time during the simulation. An active transaction is a transaction which has been created and not yet cancelled.

R=1 specifies that no transaction will use more than one resource. During model checkout, these parameters should be kept to a minimum to optimize core utilization.

STATEMENT 60. INTEGER STRENGTH;

This statement declares STRENGTH as a local variable within the process.

STATEMENT 70. GET FILE(CARD) LIST(SIMTIME, INTTIME, SERTIME);

This statement is a PL/1 statement which has been inserted into the SOL route to read the file named 'CARD' and assign the first three numerical values to the global variables SIMTIME, INTTIME, SERTIME.

STATEMENT 80. INTERRUPT = FINISH;

This statement specifies that any preempted transaction is to be sent to Label 'FINISH'.

STATEMENT 90. START: IF TIME > SIMTIME THEN STOP;

This compound statement, identified by the label 'START', tests the global variable SIMTIME against the built-in global variable TIME. TIME is a reserved word within SOL and represents the current time of the simulation. If TIME exceeds the value for SIMTIME, the simulation will be terminated, as specified by the STOP statement.

STATEMENT 100. WAIT EXPONENTIAL(INTTIME);

This statement specifies that the transaction is to be placed into the wait queue for the time specified by the built-in function EXP. The EXP function will sample a value from an exponential distribution with the average value INTTIME.

STATEMENT 110. NEW TRANSACTION TO START;

This statement specifies that a new transaction with the same local variables is to be created and to be sent to the label 'START'. The original transaction will continue to run until it encounters a wait status. Then the new transaction will start executing.

STATEMENT 120. PRIORITY = 1:8;

This statement assigns a random integer between 1 and 8 to the built-in local variable 'PRIORITY'. The local variable PRIORITY is used by the system to resolve any conflicts between transactions requiring the same action at the same time. In this case, it will control the seizing of the facility TRANSMITTER by transactions which have entered a queue because the facility was busy.

STATEMENT 130. STRENGTH = 1:4;

This statement assigns an integer value between 1 and 4 to the local variable STRENGTH.

STATEMENTS 140 to 160. PLIBEGIN; PUT EDIT ('CARD RECEIVED AT', TIME) (A(17), F(6)) SKIP; PLIEND;

These three statements represent a PL/1 block which was inserted to send a message to the SYSPRINT file. The PL/1 PUT statement has been bracketed by PLIBEGIN and PLIEND. In this way the entire PL/I block is bypassed by the translater and the associated text inserted unaltered. %

STATEMENT 170. SEIZE TRANSMITTER, STRENGTH;

One of the following actions takes place:

a. If the facility TRANSMITTER is not busy, the transaction simply seizes the facility and marks it busy. An entry is placed into the log file.

b. If the facility is busy with a transaction of holding strength equal or higher to the local variable STRENGTH of the calling transaction, the calling transaction enters the wait queue for this facility.

c. If the facility is busy with a transaction of lower holding strength, this transaction is preempted and sent to the action label FINISH as specified in the INTERRUPT statement.

STATEMENT 180. WAIT SERTIME;

The transaction encountering this statement will enter the wait queue for the period specified by the value of SERTIME. The next transaction in the time queue will then start executing.

STATEMENT 190. CANCEL;

This statement will cause all resources the transaction is using to be freed and the transaction to be deactivated. Corresponding entries are made in the log file.

STATEMENT 200. FINISH;

This is a simple label statement that has been specified as the action label for an interrupt.

STATEMENTS 210 to 230. PLIBEGIN; PUT EDIT ('PREEMPTION OCCURRED AT',TIME) (A(23), F(6)) SKIP; PLIEND;

These three statements represent a PL/I block, similar to statements 140 to 160, which will cause a message to be sent to the SYSPRINT file whenever a transaction is preempted.

STATEMENT 240. CANCEL;

This statement will deactivate the transaction.

STATEMENT 250. END;

The END statement identifies the end of the process and is ignored by the transactions.

```
/* MODEL1F  -  SINGLE SERVER QUEUING MODEL */
/*                    WITH EXTERNAL QUEUE      */
INTEGER SIMTIME,INTTIME,SERTIME;
FACILITY TRANSMITTER;
STORE 1000 QUEUE;
PROCESS CONTROL,T=1,R=0;
GET FILE(CARD) LIST(SIMTIME,INTTIME,SERTIME);
WAIT SIMTIME;
STOP;
END;
PROCESS TRANSMIT, T=10, R=2;
INTEGER STRENGTH;
INTERRUPT = FINISH;
START;
       WAIT EXPONENTIAL(INTTIME);
       NEW TRANSACTION TO START;
       PRIORITY = 1:8;
       STRENGTH = 1:4;
       PUT EDIT ('CARD RECEIVED AT ',TIME)(A(17),F(6)) SKIP;
       ENTER QUEUE;
       SEIZE TRANSMITTER, STRENGTH;
       WAIT SERTIME;
       CANCEL;
FINISH:  PUT EDIT('PREEMPTION OCCURRED AT ',TIME) (A(23),F(6)) SKIP;
CANCEL;
END;

/* MODEL1G  -  SINGLE SERVER QUEUING MODEL WITH EXTERNAL QUEUE */
INTEGER SIMTIME,INTTIME,SERTIME;
FACILITY TRANSMITTER(5),RECEIVER(5);
STORE 1000 SENDQUEUE(5), 1000 RECEIVEQUEUE(5);
PROCESS CONTROL, T=1, R=0;
GET FILE(CARD) LIST(SIMTIME,INTTIME,SERTIME);
WAIT SIMTIME;
STOP;
END;
PROCESS TRANSMIT, T=10, R=4;
INTEGER STRENGTH,ORIG,DEST,NUMBER;
INTERRUPT = FINISH;
NUMBER=0;
START;
        WAIT EXPONENTIAL(INTTIME);
        NEW TRANSACTION TO START;
        NUMBER = NUMBER+1;
        PRIORITY = 1:8;
        STRENGTH = 1:4;
        ORIG = 1 : 5;
        DEST = 1 : 5;
PLIBEGIN;
        PUT EDIT ('CARD ',NUMBER,' RECEIVED AT NODE ',ORIG,
        ' AT TIME ',TTIME) (A(5),F(4),A(18),F(2),A(9),F(6)) SKIP;
PLIEND;
            ENTER SENDQUEUE(ORIG);
```

```
            SEIZE TRANSMITTER(ORIG),STRENGTH;
            ENTER RECEIVEQUEUE(DEST);
            SEIZE RECEIVER(DEST),STRENGTH;
            WAIT SERTIME;
            CANCEL;
FINISH;
PLIBEGIN;
   PUT EDIT('CARD ',NUMBER,' PREEMPTED AT TIME ',TIME)
           (A(5),F(4),A(19),F(6)) SKIP;
PLIEND;
CANCEL;
END;


INPUT PARAMETER LIST FOR STATISTICS.     SOL.DATA(STATIN)

                    0,
                 'NO',
                 'NO',

CLASS(Z) OUTPUT OF STATISTICS STEP 'SOL(S)' FOR MODEL1G

  NAME OF FACILITY        TIME        FRACTION OF TIME IN USE
TRANSMITTER (  1)         20866              0.0958
TRANSMITTER (  2)         20866              0.1114
TRANSMITTER (  3)         20866              0.1534
TRANSMITTER (  4)         20866              0.1342
TRANSMITTER (  5)         20866              0.2210
   RECEIVER (  1)         20866              0.1725
   RECEIVER (  2)         20866              0.1279
   RECEIVER (  3)         20866              0.0958
   RECEIVER (  4)         20866              0.1150
   RECEIVER (  5)         20866              0.1725
    NAME OF STORE         TIME   CAPCTY   MAX USD   TOTAL OCCP   AVG UTL
  SENDQUEUE (  1)        20866     1000         1         2000    0.0001
  SENDQUEUE (  2)        20866     1000         2         2324    0.0001
  SENDQUEUE (  3)        20866     1000         1         3200    0.0002
  SENDQUEUE (  4)        20866     1000         1         2800    0.0001
  SENDQUEUE (  5)        20866     1000         2         4933    0.0002
RECEIVEQUEUE(  1)        20866     1000         2         3655    0.0002
RECEIVEQUEUE(  2)        20866     1000         1         2669    0.0001
RECEIVEQUEUE(  3)        20866     1000         1         2000    0.0001
RECEIVEQUEUE(  4)        20866     1000         1         2400    0.0001
RECEIVEQUEUE(  5)        20866     1000         2         4222    0.0002
```

```
CLASS(Y) OUTPUT OF MODEL1G

CARD      1 RECEIVED AT NODE  1 AT TIME     470
CARD      1 RECEIVED AT NODE  3 AT TIME     716
CARD      1 RECEIVED AT NODE  5 AT TIME     962
CARD      1 RECEIVED AT NODE  5 AT TIME    1051
CARD      1 RECEIVED AT NODE  2 AT TIME    1185
CARD      1 RECEIVED AT NODE  4 AT TIME    1525
CARD      1 RECEIVED AT NODE  4 AT TIME    2261
CARD      1 RECEIVED AT NODE  5 AT TIME    3501
CARD      1 RECEIVED AT NODE  4 AT TIME    3855
CARD      1 RECEIVED AT NODE  3 AT TIME    4066
CARD      1 RECEIVED AT NODE  1 AT TIME    4601
CARD      1 RECEIVED AT NODE  3 AT TIME    5137
CARD      1 RECEIVED AT NODE  3 AT TIME    5987
CARD      1 RECEIVED AT NODE  5 AT TIME    6319
CARD      1 RECEIVED AT NODE  3 AT TIME    6869
CARD      1 RECEIVED AT NODE  5 AT TIME    7065
CARD      1 RECEIVED AT NODE  5 AT TIME    9217
CARD      1 RECEIVED AT NODE  2 AT TIME    9835
CARD      1 RECEIVED AT NODE  5 AT TIME    9957
CARD      1 RECEIVED AT NODE  2 AT TIME   10104
CARD      1 RECEIVED AT NODE  3 AT TIME   10282
CARD      1 RECEIVED AT NODE  3 AT TIME   11004
CARD      1 RECEIVED AT NODE  2 AT TIME   11465
CARD      1 RECEIVED AT NODE  4 AT TIME   12102
CARD      1 RECEIVED AT NODE  2 AT TIME   12447
CARD      1 RECEIVED AT NODE  1 AT TIME   13239
CARD      1 RECEIVED AT NODE  1 AT TIME   14493
CARD      1 RECEIVED AT NODE  4 AT TIME   14932
CARD      1 RECEIVED AT NODE  5 AT TIME   14933
CARD      1 RECEIVED AT NODE  4 AT TIME   15340
CARD      1 RECEIVED AT NODE  3 AT TIME   15512
CARD      1 RECEIVED AT NODE  4 AT TIME   16097
CARD      1 RECEIVED AT NODE  5 AT TIME   16307
SIMULATION TERMINATED - I/O ERROR
```

```
  CONTENTS OF FILE SOL.DATA(GOIN): 2000, 500, 400

/* MODEL1H - 5-NODE FULLY CONNECTED NETWORK WITH BLOCKED LINKS */
INTEGER SIMTIME,INTTIME,SERTIME,NUM;
PLIBEGIN;
DCL CONN(5,5) FIXED BIN(31);
PLIEND;
FACILITY TRANSMITTER(5),RECEIVER(5);
STORE 1000 SENDQUEUE(5), 1000 RECEIVEQUEUE(5), 8 LINK(10);
PROCESS CONTROL, T=1, R=0;
GET FILE(CARD) LIST(SIMTIME,INTTIME,SERTIME,CONN);
WAIT SIMTIME;
STOP;
END;
PROCESS TRANSMIT, T=50, R=5;
INTEGER STRENGTH,ORIG,DEST,NUMBER;
INTERRUPT = FINISH;
NUM=0;

START:
       WAIT EXPONENTIAL(INTTIME);
       NEW TRANSACTION TO START;
       NUMBER,NUM = NUM+1;
       PRIORITY = 1:8;
       STRENGTH = 1:4;
       ORIG = 1 : 5;
RET:   DEST = 1 : 5;
       IF DEST = ORIG THEN GO TO RET;
     PUT EDIT ('CARD ',NUMBER,' RECEIVED AT NODE ',ORIG,' AT TIME ',
TIME ', TO =',DEST) (A(5),F(4),A(18),F(2),A(9),F(6),A(5),F(2)) SKIP;
       ENTER SENDQUEUE(ORIG);
       SEIZE TRANSMITTER(ORIG),STRENGTH;
       ENTER LINK(CONN(ORIG,DEST));
       ENTER RECEIVEQUEUE(DEST);
       SEIZE RECEIVER(DEST),STRENGTH;
       WAIT SERTIME;
        CANCEL;
FINISH:   PUT EDIT('CARD ',NUMBER,' PREEMPTED AT TIME ',TIME)
          (A(5),F(4),A(19),F(6))  SKIP;
CANCEL;
END;


Input Dataset SOL.DATA(GOIN):

20000,100,500,
0,1,2,3,4,
1,0,5,6,7,
2,5,0,8,9,
3,6,8,0,10,
4,7,9,10,0,
```

For background execution a number of keywords may be entered optionally. These keywords can be used to override the standard default parameters for job execution. These are listed for each command in Figure A1.

The SOL Simulation System can be operated via the remote terminal in the foreground or by remote job entry. Figure A2 illustrates the use of individual commands to execute the job steps separately or as a whole.

There are four job steps:

1. Translate
2. Compile
3. Go
4. Statistics Generation.

The execution of the commands directly to the left or to the right of the box representing this step will generate the necessary code to execute the step. The commands to the left will initiate foreground jobs, while the commands to the right initiate background jobs. In either case, positional parameters and keywords must be passed to the command procedure. If positional parameters are left out, the system will prompt the user to enter the positional parameters. For foreground execution only one parameter - the model name - must be entered; e.g.,

```
SOLSTAT MODELIA
```

For Remote Job Entry it is necessary to enter four positional parameters in the following sequence:

1. Last name of user
2. Badge number of user
3. Task number
4. Model name

For instance:

```
SOLFCG MILLER 1217 8N15 MODEL1A
```

To allow execution of several job steps in one run combined commands have been created. Their range is indicated by a bracket.

Whenever a command is entered, a Command List is executed. This list either allocates the files and data sets and calls the load modules (foreground execution), or generates the necessary job control cards and submits the job for execution.

A special help routine is available on TSO to inquire about the SOL operating procedures. To invoke this rouitne enter the command

SOLHELP

To inquire about the structure and keywords of a command enter the following command

SOLHELP C(<command name>)

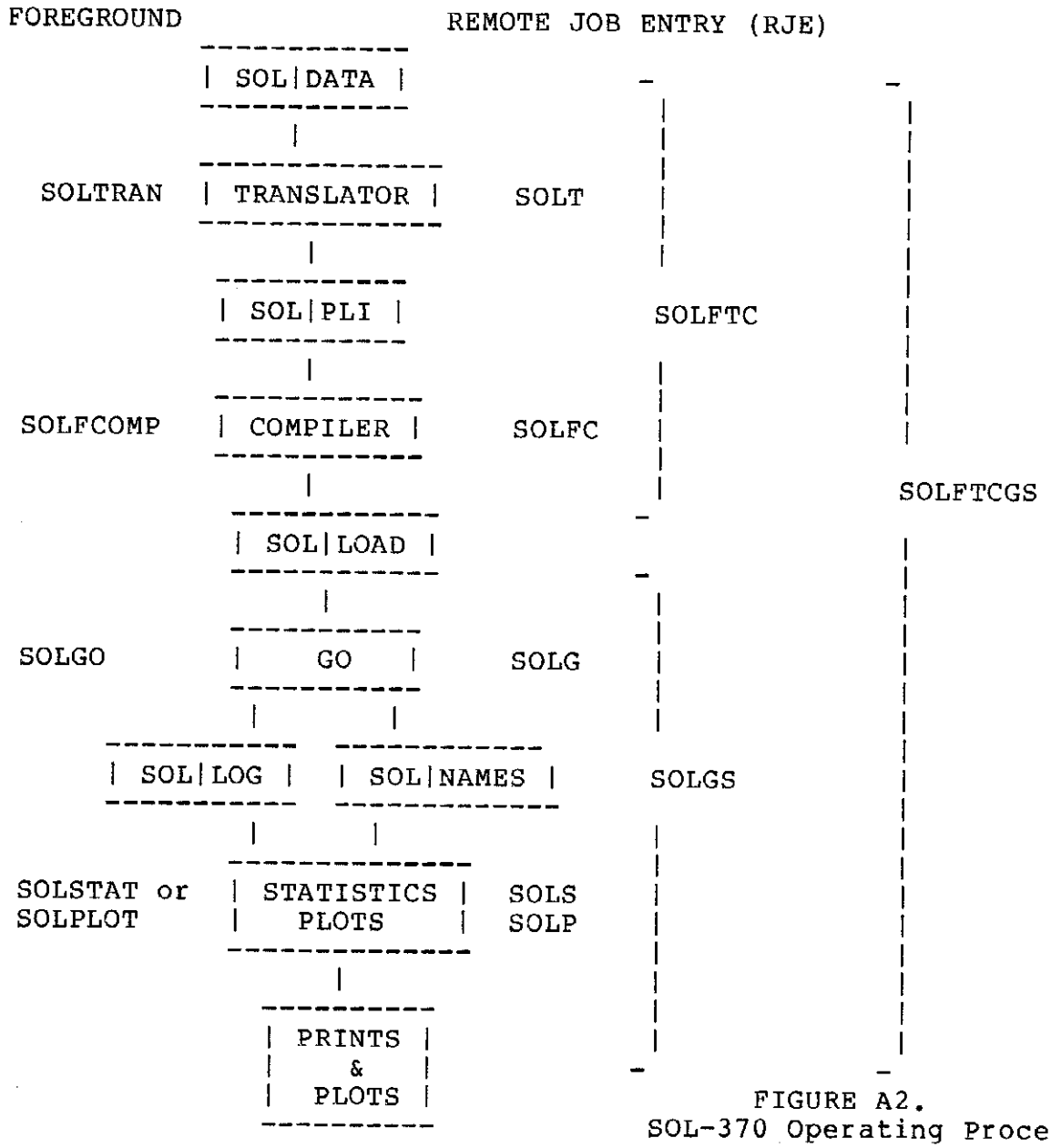


FIGURE A2.
SOL-370 Operating Procedures

OPTIONAL KEYWORDS HAVE DEFAULTS WHICH ARE LISTED IN PARENTHESIS. TABLE ENTRIES WITHOUT DEFAULTS ARE FIXED PARAMETERS THAT CANNOT BE CHANGED.

	SOLT	SOLOC SOLFTC SOLFC	SOLGS SOLG	SOLFCG SOLFTCGS SOLFCGS
JOB CLASS	B	CL(B)	CL(C)	CL(B)
REGION SIZE	150K	150K	R(250K)	R(250K)
JOB NUMBER	NO(0)	NO(0)	NO(0)	NO(0)
CPU TIME	5	5	TIME(5)	TIME(5)
OUTPUT	OUT(X)	OUT(X)	OUT(X)	OUT(X)
COMPILE OPTION	-	OPTION(NS)	-	-
COMPILE OPTIMIZE	-	OPT(1)	-	-
GO-STEP INPUT	-	-	GOIN(GOIN)	GOIN(GOIN)
STAT-STEP INPUT	-	-	STATIN(STATIN)	STATIN(STATIN)

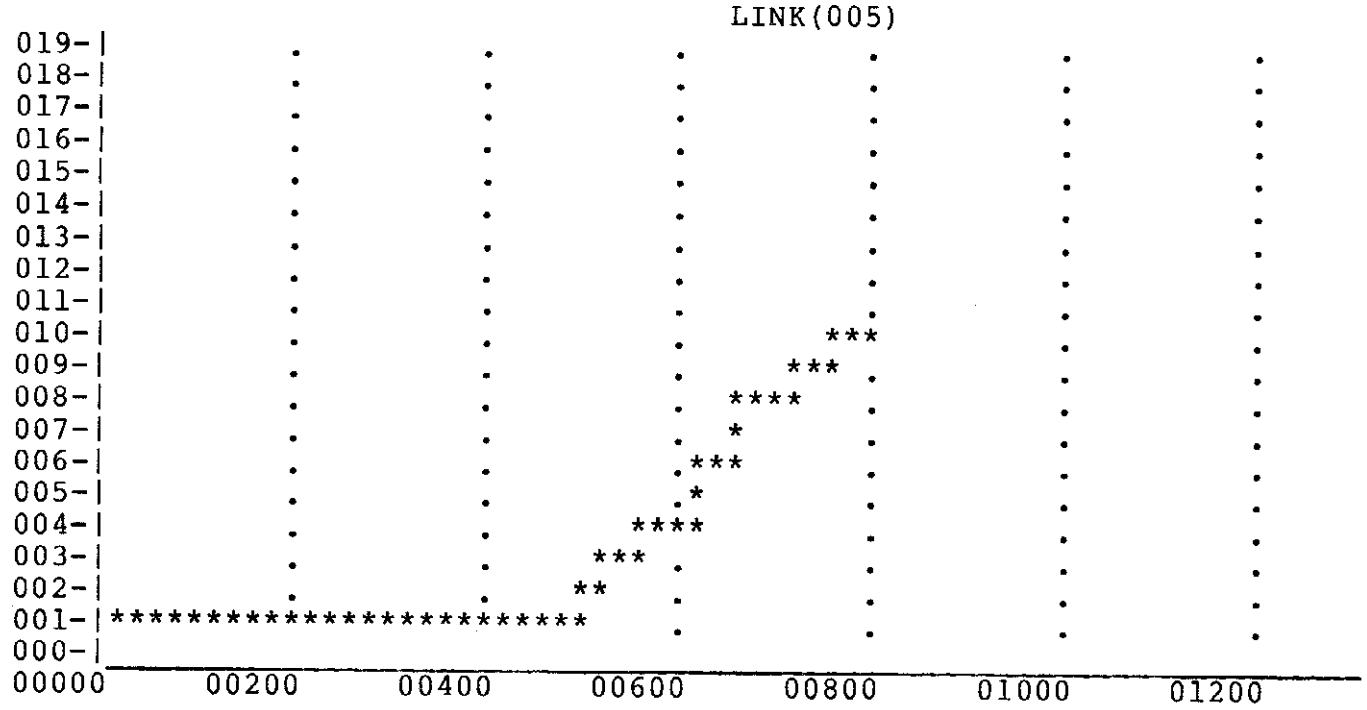
	SOLP SOLS	SOLTPS	SOLTCCG SOLCCG	SOLTPGS
JOB CLASS	CL(B)	CL(B)	CL(C)	CL(D)
REGION SIZE	200K	200K	R(250K)	R(350K)
JOB NUMBER	NO(0)	NO(0)	NO(0)	NO(0)
CPU TIME	5	TIME(5)	5	TIME(30)
OUTPUT	OUT(A)	A	OUT(X)	A
GO-STEP INPUT	-	-	GOIN(GOIN)	GOIN(GOIN)
STAT-STEP INPUT	STATIN(STATIN)	STATIN(STATIN)		STATIN(STATIN)
PLOT-STEP INPUT	PLOTIN(PLOTIN)	-	-	-
BLOCK	-	-	BL(0)	-
TAPE LIB.NUMBER	-	TP(?)	-	TP(SCRATCH)

Figure A1. Keyword List For SOL Commands

```
TSO SAMPLE SESSION


IKJ54012A ENTER LOGON
LOGON 1315 ACCT(9X40) P(SOLLOG)
 READY


******************  SOL-SYSTEM INITIALIZATION  **********************
SOL ALLOC
******* TEMPORARY SOL DATASETS ARE BEING CREATED ***********
            SOL.DATA
            SOL.PLI
            SOL.LOAD
            SOL.NAMES
            SOL.LOG
 READY
************************  MODEL CREATION ****************************

 READY                                          ++++++++++++++++++++++++
EDIT SOL(MODEL1C) DATA NEW                      + TO CREATE            +
10 ........FIRST LINE OF SOL CODE....           + SOL MODEL AS MEMBER +
20 ........SECOND LINE OF SOL CODE...           + IN TEMP. DATASET     +
30 ..................................           + SOL.DATA             +
130 .................................           ++++++++++++++++++++++++
140 (CARR.RET.)
SAVE
 SAVED
END
 READY
**********************  TRANSLATION STEP ****************************
SOLTRAN  MODEL1C                                 + TO TRANSLATE    +
                                                 + SOL SOURCE CODE +
   ***** END OF FILE ON SYSIN                     + TO PL/1 CODE    +
                                                 ++++++++++++++++++
   NUMBER OF INPUT CARDS:  13   NUMBER OF ERRORS:  0
READY

************************  COMPILE STEP  *****************************
                                                  +++++++++++++++
SOLFC MILLER 1315 9X40 MODEL1C                    + TO COMPILE  +
                                                  + & LINK-EDIT +
                                                  +++++++++++++++
PL1/F COMPILER & LINK-EDIT OUTPUT IS CLASS(X)
 JOB R1315c0 IS BEING EXECUTED



 IEF4041 R1315c0 ENDED        TIME=10.36.05
OUTPUT R1315c0 CLASS(R)
```

```
  ************************************************************
  *         JCL OF COMPILE & LINK-EDIT STEPS WILL BE DISPLAYED*     *
  ************************************************************
 READY
OUTPUT R1315c0 CLASS(X) PAUSE

  ************************************************************
  *                                                          *
  *      COMPILE AND LINK-EDIT LISTING WILL BE DISPLAYED     *
  *                                                          *
  *         TO SAVE ENTER SAVE 'DSNAME'                      *
  ************************************************************
END
 READY
*************************** GO STEP ********************************
SOLG  MILLER 1315 9X40 MODEL1C                          +++++++++++++++
SIMULATION OUTPUT I CLASS(X)                            + TO EXECUTE   +
                                                        + MODEL        +
                                                        +++++++++++++++

  JOB R1315G0 IS BEING EXECUTED

  IEF4041 R1315G0 ENDED       TIME=10.44.23
  READY
*********************** STATISTICS STEP ***************************
                                                        +++++++++++++++
SOLSTAT  MODEL1C                                        + TO ANALYSE   +
 ENTER STARTTIME                                        + SIMULATION   +
0                                                       + OUTPUT       +
        0 INIT TIME UNITS SUPPRESSED                    +++++++++++++++
 PLOT FUNCTION REQUESTED ? ("YES" or "NO")
'YES'
 PLOT FUNCTION HAS BEEN REQUESTED
 ENTER TYPE AND SEQUENCE NO. OF RESOURCE
1,3
                  LINK IS TO BE PLOTTED
 ENTER VERTICAL PARAMETERS-(MIN,INCR,MAX)
1,1,4
 VERTICAL PARAMETERS:  MIN= 1 INC= 1 MAX=4
 ENTER HORIZONTAL PARAMETERS-(MIN,INCR,MAX)
1,1,10
 HORIZONTAL PARAMETERS: MIN= 1  INC= 1  MAX= 10
 MORE RESOURCES TO BE PLOTTED ? ("YES" or "NO")
'NO'
 ARE SNAPSHOTS REQUESTED ? ("YES" or "NO")
'YES'
 ENTER TIME VALUE
5000
 SNAPSHOT TIME IS     5000
        ********************************************
        *   STATISTICS AT TIME 5000 OF SIMULATION.   *
        *              IS DISPLAYED                  *
        *                   HERE                     *
        ********************************************
 ENTER TIME VALUE
100000
```

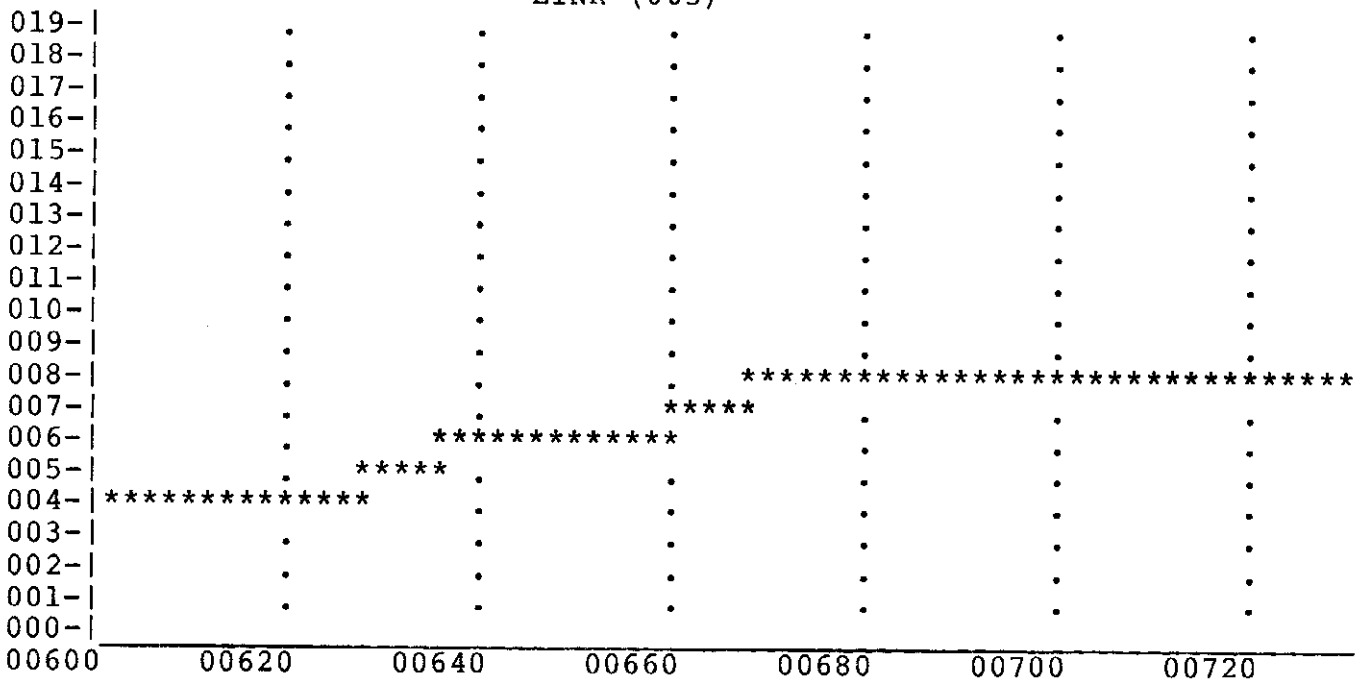
```
 SNAPSHOT TIME IS    100000
                ************************************
                *  STATISTICS FOR LAST TIME ENTRY    *
                *            IS DISPLAYED            *
                *                HERE                *
                ************************************
READY


*************************  SOLPLOT  *********************************

SOLPLOT PACKNET

************************************************************
*               SOL PLOT ROUTINE INVOKED                   *
*           PRINT OUTPUT WILL BE IN DATASET                *
*                    PLOTOUT.DATA                          *
*   TO PRINT THIS DATASET USE THE FOLLOWING COMMAND        *
* WPRT PLOTOUT.DATA <YOUR NAME> FRM(17) NOPAGENO SINGLE    *
************************************************************
 ENTER START TIME FOR PLOT

0

 PLOT ANALYSIS STARTING AT TIME =     0
 THE FOLLOWING STORES & TRUNKS WERE MODELLED

 TYPE,ID              STORE NAME  SUBSCR   CAPCT
    1, 1                    CORE       9   10000
    1, 2              DISASSQUEUE      9    1000
    1, 3                 ASSQUEUE      9    1000
    1, 4                SENTQUEUE      9    1000
    1, 5               TASKQUEUE1      9    1000
    1, 6               TASKQUEUE2      9    1000
    1, 7               TASKQUEUE3      9    1000
    1, 8               TASKQUEUE4      9    1000
    1, 9               TASKQUEUE5      9    1000
    1,10                     LINK     20      10

 ENTER IDENTIFIER    ( TYPE,ID )
1,10

 ENTER SUBSCRIPT ( 0 FOR COMPOUND PLOT )
5

 ENTER VERTICAL INCREMENT VALUE FOR PLOT
1

 CLEAR SCREEN AND ENTER START TIME & INCREMENT
0,20
```



TO PRINT THIS PLOT RESPOND WITH "YES" ELSE "NO"
'NO'

TO MODIFY TIME SCALE RESPOND WITH "YES" ELSE "NO"
'YES'

CLEAR SCREEN AND ENTER START TIME & INCREMENT
600,2



```
 TO PRINT THIS PLOT RESPOND WITH "YES" ELSE "NO"
'YES'

 TO MODIFY TIME SCALE RESPOND WITH "YES" ELSE "NO"
'NO'

 TO CREATE OTHER PLOTS RESPOND WITH "YES" ELSE "NO"
'NO'

 READY
```

V. BIBLIOGRAPHY AND REFERENCES

1. D. E. Knuth and J. L. McNeley, "A Formal Definition of SOL," IEEE Transactions on Electronic Computers, EC-13 No. 5 (Aug 1964) pp 409-414

2. D. E. Knuth and J. L. McNeley, "SOL - A Symbolic Language for General Purpose Systems Simulation ," IEEE Transactions on Electronic Computers, IC-13, No. 5 (Aug 1964) pp 401-408

3. R&D Technical Report ECOM-3085 (AD-850159L), "MALLARD Trafiic Simulation, Results and Analysis, Final Report," James A. Armstrong and Horst E. Ulfers, Feb 1969

4. J. Armstrong, H. Ulfers, D. Miller, H. Page, "SOLPASS - A Simulation Oriented Language Programming an Simulation System," Proceedings of the Third Conference on Applications of Simulation, Dec 1969

5. R&D Technical Report ECOM-0043-F, "SOL Compiler Design," H.C. Page, D.J. Miller (Patterson & Smith Inc), Feb 1968 Feb 1968

6. Horst E. Ulfers, "PACKNET - A Packet Switch Network Simulator," Proceedings of the 1975 ICC, June 1975

7. C. G. Guffee and H. E. Ulfers , "SOL-370," Proceedings of the 1975 Summer Computer Simulation Conference, July 1975, pp 1-11

TECHNICAL NOTE NO. 23-76

SOL-370

INSTALLATION AND ERROR TRACING GUIDE

AUGUST 1976

Prepared by:

o Horst Ulfers

Approved for Publication:

ROBERT E. LYONS
Chief, Computer Systems Division

FOREWORD

The Defense Communications Engineering Center (DCEC) Technical Notes (TN's) are published to inform interested members of the defense community regarding technical activities of the Center, completed and in progress. They are intended to stimulate thinking and encourage information exchange; but they do not represent an approved position or policy of DCEC, and should not be used as authoritative guidance for related planning and/or further action.

Comments or technical inquiries concerning this document are welcome, and should be directed to:

Director
Defense Communications Engineering Center
1860 Wiehle Avenue
Reston, Virginia, 22090

TABLE OF CONTENTS

I. INTRODUCTION

This Technical Note contains instructions for detection and tracing of errors which occur during translation, compilation, or execution of simulation models implemented with the SOL-370 Simulation System. It also gives instructions for the installation and initial testing of the system. It is to be used as a supplement to Technical Note No. 25-75, "SOL-370 Language Reference Manual and Users' Guide."

Each step of the SOL-370 system has its own error detection scheme in addition to the standard IBM error indications.

The SOL-370 translator checks each SOL source statement for propper coding and will generate an error message, which is either displayed at the users TSO terminal or listed in the printer output. If no error has been found the following message is produced:

NUMBER OF INPUT CARDS: <n> NUMBER OF ERRORS: 0

The translator does not check the PL/1 statements which are allowed to be intermixed with the SOL stements. The entire PL/1 code is checked at compile time.

The PL/1-F, PL/1-Checkout, or PL/1-Optimizing compiler will produce the regular IBM error messages which refer to particular statement or line numbers of the PL/1 code. The 3rd to 6th digits of the line number, referred to, represent the line number of the original SOL code, which contains the error. For a more detailed description how to invoke this numbering option refer to section III-2.

When executing the model the SOL-370 System monitors the proper functioning of the model and terminates execution whenever an error is detected. At this time an error message including an completion code followed by a dump of the important variables is produced. When operating in the TSO mode the user may inquire about the completion code by entering the command

SOLERROR.

A short list of all completion codes will then be displayed (figure 1). A more detailed description of a particular error condition is displayed by entering this command:

SOLERR O(<abend code>)

The listings produced are shown in section II.

II. COMPLETION CODES

One of the following completion codes is produced at the completion of each simulation run. Any code other than 0000 indicates a run-time error. The error may be caused by a user mistake or systems malfunctioning. In the case of a user abend code, the error is contained in the SOL source code and the user should carefully check the code. In the case of a system abend code, the user should obtain a listing of the SOL source code, a complete compile listing (compile the model with OPTION(OFFSET)), and the error dump produced by the execution of the model, and submit those listings to SOL Systems Maintainance for analysis. A summary table of the completion codes is provided in Table I.

SYSTEM ABEND CODE 0001

RELATED SOL STATEMENT - RELEASE

ERROR CONDITION	- A transaction releases a facility when other transactions are queued up to seize the same facility. The facility queue does not contain a transaction waiting on this facility, however.
USER ACTION	- Save SOL error dump and notify Systems of error condition.
ERROR FLAGGED IN	Procedure RELEAS locations 50010970 / 50011000

SYSTEM ABEND CODE 0010

RELATED SOL STATEMENT - DEMAND

ERROR CONDITION	- This error condition is raised when a transaction holding a trunk is preempted and the system cannot find the preempted transaction in the trunk queue.
USER ACTION	- Save SOL error dump and notify Systems of condition.
ERROR FLAGGED IN	Procedure $CONTROL location 50001960

USER ABEND CODE 0011

RELATED SOL STATEMENT - ENTER

ERROR CONDITION - The amount of storage requested in an enter statement exceeds the declared capacity of this store.

USER ACTION - Check store name and capacity value in the enter statement. If ok., increase the value in the declaration for this store or change the amount requested in the crresponding enter statement.

ERROR FLAGGED IN Procedure ENTER, location 50020110

USER ABEND CODE 0100

RELATED SOL Procedure - YIELD

ERROR CONDITION - The transaction executing a yield statement has not previously demanded the trunk it is trying to yield now or the amount to be returned to the trunk is larger than the transaction had taken from the trunk.

USER ACTION - Check for bad trunk name or capacity value in yield statement.

ERROR FLAGGED IN Procedure YIELD
Locations - 50005870 / 50005910

USER ABEND CODE 0101

RELATED SOL STATEMENT - RELEASE

ERROR CONDITION - The transaction executing the release statement has not previously seized the facility it is trying to release now.

USER ACTION - Check for bad facility name used in the release statement.

ERROR FLAGGED IN Procedure RELEAS
Location 50010650

SYSTEM ABEND CODE 1000

RELATED SOL STATEMENTS - DEMAND, YIELD

ERROR CONDITION - A transaction has been preempted on a trunk. However, an entry for the transaction cannot be found in the $WU- and $WUINFO- arrays.

USER ACTION - Save error dump and notify Systems of error condition.

ERROR FLAGGED IN Procedure DEMAND
Locations 50005400 / 50005480
Procedure YIELD
Locations 50004630 / 50004710

USER ABEND CODE 1001

RELATED SOL STATEMENT - LEAVE

ERROR CONDITION - The transaction executing the leave statement tries to return capacity to a store it has not previously used, or it tries to return more capacity units than it has taken.

USER ACTION - Check for bad store name or capacity in the leave statement.

ERROR FLAGGED IN Procedure LEAVE
Location 50020700

USER ABEND CODE 1010

RELATED SOL STATEMENT - NEW TRANSACTION TO

ERROR CONDITION - The execution of the NEW TRANSACTION TO statement causes the creation of a new transaction. This error condition is raised when active transactions in the model exceed the number of transactions declared in the process declaration (T=?)

USER ACTION - Increase the value for T in the process declaration.

ERROR FLAGGED IN Procedure $NEWTRN
Location 50005690

SYSTEM ABEND CODE 1011

RELATED SOL STATEMENTS - SEIZE, ENTER

ERROR CONDITION - On requesting a resource (facility or store) the $ACTIVE array is checked for a prevous use of this resource. This error condition is raised when a matching resource has been found, but the type had been set to zero (0) erroneously.

USER ACTION - Save SOL error dump and notify Systems of error condition

ERROR FLAGGED IN Procedure $UPDATE
Location 50004890

USER ABEND CODE 1101

RELATED SOL STATEMENTS - SEIZE, ENTER, DEMAND

ERROR CONDITION - A request for a resource (store, facility, or trunk) is made by a transaction. However, the space reserved for storage per transaction has been exhausted.

USER ACTION - Increase the storage availability by a higher value for R in the process declaration.

ERROR FLAGGED IN Procedure $UPDATE
Location 50005070

USER ABEND CODE 1110

RELATED SOL STATEMENT - DISTRIBUTION

ERROR CONDITION - This error condition is raised when the distribution function contains more than 100 arguments.

USER ACTION - Correct distribution statement

ERROR FLAGGED IN Procedure $DISTR
Location 50007410

SYSTEM ABEND CODE 1111

RELATED SOL STATEMENT - ANYPLACE

ERROR CONDITION - This error condition is raised whenever one of the IBM run-time error conditions is detected. An additional message with the corresponding IBM Abend code is printed.

USER ACTION - If error is related to users' SOL source code, correct the error. Refer to IBM the error.

Table I. SUMMARY TABLE OF SOL COMPLETION CODES.

COMPL.CODE	TYPE	LOCATION	ERROR CONDITION
0000	SYSTEM	$CONTROL	NO ERRORS DETECTED
0001	SYSTEM	RELEAS	FACILITY NOT IN $ACTIVE
0010	SYSTEM	DEMAND	TRUNK NOT IN $ACTIVE
0011	USER	ENTER	REQUESTD STORAGE TOO LARGE
0100	USER	YIELD	BAD TRUNK NAME, OR NOT PREVIOUSLY DEMANDED OR AMOUNT YIELDED TOO BIG
0101	USER	RELEASE	BAD FACILITY NAME, OR NOT PREVIOUSLY SEIZED
0110	SYSTEM	$CONTROL	NO TRANSACTION IN QUEUE
0111	USER	FIRST	END OF FILE ON INPUT
1000	SYSTEM	DEMAND	NO RECORD OF PREEMPTED TRANSACTION IN $ACTIVE
1001	SYSTEM	LEAVE	BAD STORE NAME, OR NOT PREVIOUSLY ENTERED, OR AMOUNT RETURNED TOO BIG
1010	USER	NEW TRANSAC-TION TO	TRANSACTION OVERFLOW, R IN PROCESS DECLARATION SELECTED TOO SMALL
1011	SYSTEM	$UPDATE	ILLEGAL TYPE = 0
1100	---------	NOT USED	----------
1101	USER	SEIZE ENTER DEMAND	RESOURCE OVERFLOW, R IN PROCESS DECLARATION SELECTED TOO SMALL
1110	USER	DISTRI-BUTION	DISTRIBUTION FUNCTION HAS TOO MANY ARGUMENTS
1111	SYSTEM	PART1	SYSTEM ABEND

III. TRACING FACILITIES

To verify the logic of a model or to associate error messages with line numbers in the SOL or PL/1 source code the SOL-370 System provides the following tracing facilities.

1. TRANSACTION TRACING FACILITIES

In debugging the model, it may become neccessary to trace the flow of a number of transactions through the model. The SOL-370 system provides a special tracing facility to accomplish this. This tracing facility provides for a trace of up to 15 transactions and can be activated at a specific time during simulation. Similarly it can be deactivated.

The tracing facility must be invoked at translation time by specifying the following comment card ahead of the program:

```
/*          $TRACE          */
```

Then the SOL source code must be renumbered starting with 10 and subsequently incremented by 10. The following actions will then take place:

a. During translation, the translator will generate a trace call at all labels and at all transfer points to the control module. It also generates the preprocessor statement to include the librarian module SOLINC(TRACE) and the associated GET statement to read the trace parameters.

b. At compile time the trace module is patched into the program. The trace module must be contained in the SOL library dataset allocated to file SOLINC with the member name of TRACE.

c. During execution of the model, the program will read first the 17 parameter values for the trace routine. These parameters must be the first 17 values in the dataset allocated to file CARD (default allocation is 'SOL.DATA(GOIN)'). The significance of the values read is as follows:

First value:	Simulation clock time at which the trace is to be invoked.
Second value:	Simulation clock time at which the trace is to be deactivated.

3rd -17th value: Transaction identification numbers of the 15 transactions to be traced. There must be 15 values or commas. The transactions are numbered in order of arrival, starting with one transaction each at each process. Process one will be activated first. When a transaction has been canceled, its number will be reassigned to the next transaction.

The trace printout is placed into the print dataset allocated to print file PRINTER. A sample of the resulting trace follows:

	********** TRANSACTION IDENTIFICATION NO *********									
TIME	1	2	3	4	5	6	7	8	9	10
0	50									
0	60									
0	70	90								
0	70	120								
0	80	120								
67	60	120								
67	70	120	90							
67	70	120	120							
101	80	120	120							
101	60	120	120	90						
101	70	120	120	120						
117	70	130	120	120						
117	70	150	120	120						
129	70		120	120						
137	70		120	130						
145	70		120	150						
156	70	90	120	150						
171	70	120	120							

The numbers listed in the columns under the transaction identfication numbers represent the last statement number the transaction was identified with at that clock time. If numbers repeat while the clock is advancing, the transaction has entered a queue state. If the number is replaced by blanks, the transaction has been canceled.

For better efficiency, the model should be recompiled without the trace facility when the model has been debugged.

2. THE $NUMOC OPTION.

The $NUMOC option of the SOL translator, when enabled at translation time numbers the lines of the generated PL/1 code in a fashion that allows easy cross-reference between the generated PLI code and the original SOL source code the model was coded in. This feature becomes important when the user wants to relate the IBM run-time error messages which are annotated with the line number of the PL/1 code to the corresponding SOL source statement.

To invoke the numbering option, the SOL source code of the model must be preceded by a comment card of the following format:

```
/*          $NUMOC          */
```

The source deck must then be renumbered starting with 10 for the comment card and in subsequent increments of 10. Each line of the generated PLI code is then numbered occording to the following code:

ABBBBBCCC

The first digit 'A' identifies the program module. Only modules 1 or 3 are generated from the SOL model code. The others correspond to the SOL library modules. The second through sixth digits 'BBBBB' correspond directly to the line number of the SOL source code.

3. THE PL/1 CHECK OPTION.

The IBM-provided CHECK condition offers a tool for a very detailed trace of variables and labels. To enable this option the user may specify the translator option $CHECK in the comment card preceding the model source code. The translator will then generate ahead of the PL/1 code the following preprocessor statement:

```
%INCLUDE SOLINC(CHECK);
```

Alternatively, the user may choose to insert this statement into the PLI deck directly. A load module compiled with this option will produce a trace of all important SOL variables and labels. Every time a variable assumes a new value, the new value and the associated statement number are printed. These listings may become very long and therefore this feature should only be used in extreme cases. In conjunction with the regular SOL error dump, which is automatically produced when an error occurs, this feature gives the experienced SOL systems programmer a very effective tool to trace a systems error.

IV. INSTALLATATION PROCEDURES

1. SYSTEMS DATASETS.

The systems tape contains two partitioned datasets :

SYS2.SOLLIB
SYS2.SOLCMDP

The first file "SYS2.SOLLIB" contains all PL/1 source library modules as follows:

SOLTRAN - The Translator Module
SOLSTAT - The Statistical Post-Analysis Module
SOLPLOT - The Plotting Routine Module
CHECK - Check Statement for Error Tracing
PART1 - Common Declarations
PART2 - Control Module and Common Procedures
Part3 - Facility Procedures
PART4 - Store Procedures
PART5 - Trunk Procedures
TRACE - Trace Procedure

The second file contains the command procedures, which allow the user to invoke the steps of the SOL-370 System individually or in combination. These procedures will either generate the proper JCL for background submission through the RJE command or invoke the system for foreground execution.

The systems tape can be read with the following JCL code:

```
//GETTAPE  JOB (MYID,R820),'MYJOB,U,MYNAME',MSGLEVEL=(1,1),
//  NOTIFY=R1111,MSGCLASS=Q,CLASS=A
/*SETUP     JOB REQUIRES TAPE SER NO MYTAPE FOR INPUT      ********
//COPYPDS  EXEC  PGM=IEHMOVE
//SYSPRINT  DD  SYSOUT=(A,U)
//SYSUT1   DD  UNIT=3330,DISP=OLD,VOL=SER=DISK01
//DD1      DD  UNIT=3330,DISP=OLD,VOL=SER=DISK02
//T1 DD DISP=(OLD,KEEP),UNIT=TAPE9,
//  DCB=(LRECL=80,BLKSIZE=800,RECFM=FB),VOL=SER=MYTAPE,
//   LABEL=(,NL)
//SYSIN  DD  *
  COPY PDS=SYS2.SOLLIB,FROM=TAPE9=(MYTAPE,1),TO=3330=DISK01,        C
               FROMDD=T1,CATLG
  COPY PDS=SYS2.SOLCMDP,FROM=TAPE9=(MYTAPE,2),TO=3330=DISK01,       C
               FROMDD=T1,CATLG
/*
```

To implement the SOL TSO commands, all authorized users should be allocated to the 'SYS2.SOLCMD' dataset at LOGON time when a special procedure P(SOLLOG) is specified. Some Command procedures contained in the dataset require CLIST commands that are not IBM standard and may not be available at your installation. In this case,the respective procedures must be modified. The commands used are as follows:

ALLOC, ATTR, CALL, CHANGE, FREE, GO TO, IF,
LABEL, PROC, TPRINT, and RJE.

Before the SOL-370 System can be operated, three load modules must be compiled and placed into the data set named 'SYS2.SOLLOAD'. The PL/1 source code of the programs to be compiled is contained in the data set 'SYS2.SOLLIB' as members:

SOLTRAN
SOLSTAT
SOLPLOT

The source code of these programs is compatible with all versions of the PL/1-F or PL/1-OPT compilers. For best run-time efficiency, the Optimizing compiler should be used with compile option OPT(2).

2. INITIAL TESTS.

To test the installed system, a test model has been provided in dataset 'SYS2.SOLCMDP(TESTMOD)' and is copied into dataset 'SOL.DATA(TESTMOD)' during initialization. To test the system, this model must be translated, compiled, executed, and analyzed. To invoke these steps from a TSO terminal follow the following procedures:

1. Testing in TSO Mode.

a. To initialize the SOL-370 system for foreground execution, enter the following command:

SOL INITIALIZE

b. To execute the SOL Translator in the foreground, enter the command:

SOLTRAN TESTMOD

c. To compile the model with the PL/1-Optimizing compiler, to execute the model, and to analyze the results, enter the following command:

SOLOCGS <YOURNAME> <YOURID> <YOURTASK> TESTMOD

d. To invoke the interactive plotting routine, enter the following command:

```
SOLPLOT TESTMOD
```

2. Testing in the Batch Mode.

The test program may also be executed in the batch mode. To do so, use the folloing JCL setup:

```
//R1591TS0 JOB (1591,T222),'R2MK0,U,ULFERS',CLASS=C,MSGLEVEL=(1,1),
//         NOTIFY=R1591,MSGCLASS=Q
//TRAN  EXEC PGM=SOLTRAN,REGION=150K
//TRAN.STEPLIB DD DSN=SYS2.SOLLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=(A,U)
//TSOOUT    DD SYSOUT=(A,U),DCB=(LRECL=131,BLKSIZE=131,RECFM=F)
//OUTF1  DD UNIT=SYSDA,SPACE=(TRK,(5,5)),
//       DCB=(LRECL=80,BLKSIZE=880,RECFM=FB)
//SOLTRAN DD DSN=&&SOLTRAN,DISP=(MOD,PASS),UNIT=SYSDA,
//       SPACE=(400,(60,60)),DCB=(BLKSIZE=400,LRECL=80,RECFM=FB)
//SYSIN DD *
INTEGER SIMTIME,INTTIME;
STORE 10 LINK(20);
FACILITY TERMINAL(100);
TRUNK 20 CPU(10);
PROCESS CONTROL, T=1, R=0;
SIMTIME=10000;
WAIT SIMTIME;
STOP;
END;
PROCESS NETWORK, T=100, R=5;
INTEGER ORIG,DEST,ROUTE,ORNODE,DENODE;
INTTIME=100;
START:
NEW TRANSACTION TO LOAD;
WAIT INTTIME;
GO TO START;
LOAD:
ORIG=1:50;
DEST=51:100;
ROUTE=1:20;
ORNODE=1:5;
DENODE=6:10;
SEIZE TERMINAL(ORIG);
DEMAND CPU(ORNODE);
WAIT 10;
ENTER LINK(ROUTE);
DEMAND CPU(DENODE);
WAIT 10;
SEIZE TERMINAL(DEST);
WAIT 1000;
CANCEL;
END;
```

```
/*
//  EXEC PL1LFCLG,TIME=5,
//   PARM.PL1L='M,NOATR,NS,NS2,EXTDIC,OPT=1',
//       REGION.GO=300K
//PL1L.SYSPRINT DD SYSOUT=(A,U)
//PL1L.SOLINC DD DSN=SYS2.SOLLIB,DISP=SHR
//PL1L.SYSIN DD DSN=&&SOLTRAN,DISP=(OLD,DELETE)
//LKED.SYSLMOD DD DSN=R1591.SOL.LOAD(TESTMOD),
//     DCB=(DSORG=PO,LRECL=13030,BLKSIZE=13030,RECFM=U),
//     SPACE=(CYL,(1,1,10)),UNIT=3330,DISP=(MOD,CATLG)
//LKED.SYSPRINT DD SYSOUT=(A,U)
//GO.SYSUT1 DD DSN=R1591.SOL.LOAD(TESTMOD),DISP=SHR
//GO.SYSPRINT DD SYSOUT=(A,U)
//GO.SYSIN DD SPACE=(TRK,(1,1)),UNIT=SYSDA
//NAMEFIL DD UNIT=3330,DSN=R1591.SOL.NAMES(TESTMOD),
//         DCB=(DSORG=PO,LRECL=13000,BLKSIZE=13000,RECFM=F),
//         DISP=(MOD,CATLG),SPACE=(TRK,(1,2,5))
//$LOGF DD UNIT=3330,DSN=R1591.SOL.LOG(TESTMOD),
//          DCB=(DSORG=PO,LRECL=465,BLKSIZE=465,RECFM=F),
//          DISP=(MOD,CATLG),SPACE=(TRK,(10,10,5))
//CARD DD *
0,10000000,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
/*
//PRINTER DD SYSOUT=(A,U)
//DISK  DD SYSOUT=(A,U)
//STAT EXEC PGM=TSOSTAT,REGION=200K
//STAT.STEPLIB DD DSN=SYS2.SOLLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=(A,U)
//SYSIN DD *
0,
NO,
NO,
10000000,
/*
//LOG DD DSN=R1591.SOL.LOG(TESTMOD),DISP=SHR
//NAMME DD DSN=R1591.SOL.NAMES(TESTMOD),DISP=SHR
/*
```

BIBLIOGRAPHY

1. D. E. Knuth and J. L. McNeley, "A Formal Definition of SOL," IEEE Transactions on Electronic Computers, EC-13 No. 5 (Aug 1964) pp 409-414.

2. D. E. Knuth and J. L. McNeley, "SOL - A Symbolic Language for General Purpose Systems Simulation ," IEEE Transactions on Electronic Computers, IC-13, No. 5 (Aug 1964) pp 401-408.

3. R&D Technical Report ECOM-3085 (AD-850159L), "MALLARD Traffic Simulation, Results and Analysis, Final Report," James A. Armstrong and Horst E. Ulfers, Feb 1969.

4. J. Armstrong, H. Ulfers, D. Miller, H. Page, "SOLPASS - A Simulation Oriented Language Programming and Simulation System," Proceedings of the Third Conference on Applications of Simulation, Dec 1969.

5. R&D Technical Report ECOM-0043-F, "SOL Compiler Design," H.C. Page, D.J. Miller (Patterson & Smith Inc),Feb 1968.

6. Horst E. Ulfers, "PACKNET - A Packet Switch Network Simulator," Proceedings of the 1975 ICC, June 1975.

7. C. G. Guffee and H. E. Ulfers , "SOL-370," Proceedings of the 1975 Summer Computer Simulation Conference, July 1975, pp 1-11.

8. DCEC TN 25-75, " SOL-370 Language Reference Manual and Users' Guide," Horst E. Ulfers, Dec 1975.