AD NUMBER: AD0915548

LIMITATION CHANGES

TO:

Approved for public release; distribution is unlimited.

FROM:

Distribution limited to U.S. Government Agencies only; Test and Evaluation; 1 Dec 1973. Other requests for this document must be referred to the Defense Communications Agency, Reston, VA 22070

AUTHORITY

DCEC ltr dtd 9 May 1974

FORTRAN SIMULATION (FORTSIM) LANGUAGE PROGRAM DESCRIPTION

Prepared for

DEFENSE COMMUNICATIONS AGENCY

Report 4195950-3-1

DCA100-73-C-0033

DECEMBER 1973

CSC

COMPUTER SCIENCES CORPORATION

FORTRAN SIMULATION (FORTSIM) LANGUAGE PROGRAM DESCRIPTION

Prepared for

DEFENSE COMMUNICATIONS AGENCY

Report 4195950-3-1

DCA100-73-C-0033

DECEMBER 1973

COMPUTER SCIENCES CORPORATION

6565 Arlington Boulevard

Falls Church, Virginia 22046

Major Offices and Facilities Throughout the World

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

ABSTRACT

This report contains a detailed description of the concept and use of the FORTRAN Simulation (FORTSIM) Language program which extends the FORTRAN language to accommodate large scale simulation models. It permits greater utilization of available memory, introduces a Macro facility similar to those available in machine language compilers, and provides programming ease with the use of indirect addressing and a method of assuring commonality between programs and subroutines. The program is executable on the IBM 360/370 and Honeywell 6000 Series computer system.

SECTION 1 - INTRODUCTION

1.1 GENERAL

The FORTRAN Simulation (FORTSIM) Language program enhances and extends the FORTRAN language for the programming and maintenance of large simulation models. It permits greater utilization of available memory by simplifying the bit manipulation process, it reduces programming time and effort by introducing a Macro facility similar to those available in many machine language compilers, and it provides programming ease with the use of indirect addressing and a method assuring commonality between programs and subroutines.

In the evolutionary process of converting large scale communication models from machine languages to higher order languages, the FORTRAN language was selected over specialized simulation languages since it provides the required versatility for complex models and is supported by most computer manufacturers. The deficiencies noted in Paragraph 1.2 were recognized and, although some specialized instructions are available which help overcome these deficiencies, they are not universally supported and frequently differ greatly from one manufacturer to another. The FORTSIM program was therefore developed to maintain machine independence while providing the required enhancements.

The FORTSIM program is currently used with all of the models which comprise the Defense Communications System Performance Simulator (Reference 1). Since the program performs a considerable number of input-output functions in processing source language statements, it is written in COBOL to take advantage of that language's efficient I-O operations. It is presently executable on the IBM 360/370 and Honeywell 6000 Series computer systems and may be adapted to any system which provides specialized FORTRAN instructions for bit manipulation.

1.2 FORTRAN DEFICIENCIES FOR SIMULATOR MODELS

The FORTRAN language can be effective in simulating large networks; however, there are several problems frequently encountered.

1.2.1 Inefficient Use of Memory

When coding in generalized FORTRAN IV to reduce machine dependence, a full memory word is required for each data item regardless of its actual length. FORTSIM provides a relatively simple means of using bit manipulation of memory without using a given manufacturer's specialized instructions in the source coding.

1.2.2 Table Addressing and Model Commonality

Each entity in a network simulation requires a table to maintain the status of its components (sizes, connectivity, queues, activity pointers, etc). FORTRAN's storage addressing technique (subscripted arrays) is not easily adaptable to managing multiple large tables. In addition, the structure of a given table must be the same in all programs and subroutines. The modification of one item in one table would require review of all programs associated with a model for required program changes. Using FORTSIM, data and table items may be referenced with indirect addressing. Each item is assigned a mnemonic reference which is further defined in an input file with an actual address and, if desired, a given bit structure within a memory word. This not only provides programming ease in manipulating network tables but also provides commonality since all programs and subroutines of a given simulation model would access the same input file.

1.2.3 Lack of Programming Aids

While FORTRAN is a powerful language in which a few statements will generate many machine language instructions, few aids are available to produce generated coding such as the Macro facility available in most machine language compilers. It is frequently feasible to develop subroutines and functions which reduce the amount of coding required; however, they are expensive in terms of running time due to linkage requirements. The FORTSIM program permits a sequence of statements to be called from an input table and inserted into source coding. It allows parameters to be passed from the calling statement and introduced into the inserted statements.

1.3 CONCEPT

The FORTSIM program, as illustrated in Figure 1-1, effectively adds an additional pass to the FORTRAN compiler in which FORTSIM's commands or pseudoinstructions in the source coding are converted to acceptable FORTRAN instructions. An additional input to the program is the Definition Table file which contains the Macro type instructions and which defines mnemonic references in terms of memory word addresses and the bits of the word to be used. Normal FORTRAN instructions in the source coding are unaffected by the program.

To facilitate the use of mnemonic references, and to simplify some forms of source coding, 10 pseudoinstructions were devised which are recognized by the program. Eight of these are used to manipulate or test data items which have been assigned mnemonic references. These instructions are:

- ADD
- SUBTRACT
- INCREMENT
- DECREMENT
- SET
- CLEAR
- MOVE
- TEST

Two instructions are used for adding statements to the source coding. The INSERT instruction is used for nonexecutable coding such as COMMON, DIMENSION, and FORMAT statements. The EXECUTE instruction is used for executable coding. Parameters may be passed only by the latter instruction.

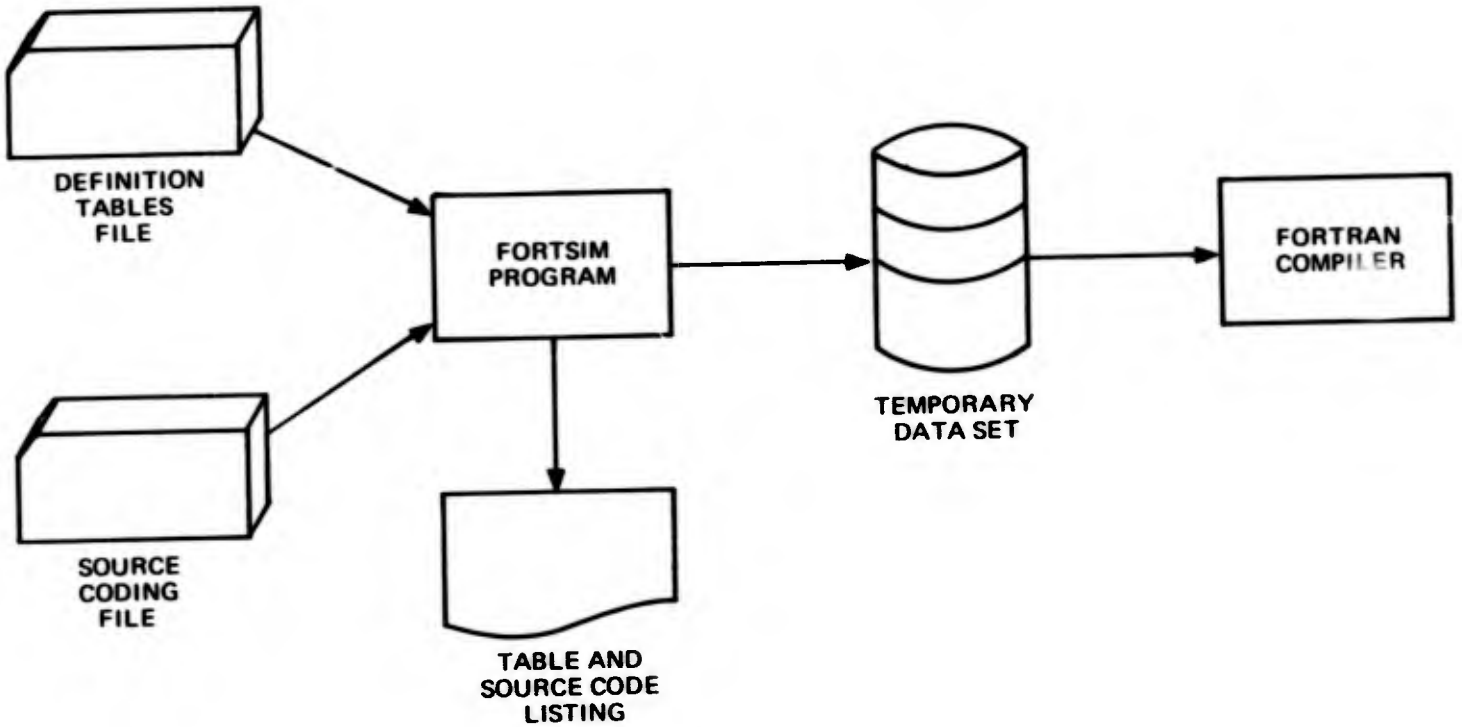


Figure 1-1. Program Data Flow

1.4 REPORT ORGANIZATION

This report contains six sections:

Section 1 - INTRODUCTION, is devoted to a general description of the program

Section 2 - DATA MANIPULATION, describes the data manipulation pseudoinstructions and provides examples of their use

Section 3 - TESTING DATA ITEMS, describes the TEST pseudoinstruction and provides examples of its use

Section 4 - MACRO INSTRUCTIONS, describes the two Macro type pseudoinstructions and provides examples of their use

Section 5 - ASSEMBLING DEFINITION TABLES, provides guidance in structuring network tables and assembling FORTSIM input tables

Section 6 - PROGRAM RESTRICTIONS AND GUIDELINES FOR USE, provides general guidance in program operation and execution.

SECTION 2 - DATA MANIPULATION

2.1 PSEUDOINSTRUCTIONS FOR DATA MANIPULATION

There are seven pseudoinstructions for the processing of data items: ADD, SUBTRACT, INCREMENT, DECREMENT, SET, CLEAR and MOVE. The instruction contains a command and one or two references. Each reference may be a mnemonic reference entered in the definition tables, a variable name, or in some cases an integer constant. The program searches the definition table for each reference, and if no match is found, treats the item as a program variable or constant.

The program changes the pseudoinstruction card to a comment card and adds the necessary coding to accomplish the desired result. This coding varies according to the instruction and the type of reference. If the instruction contains a statement number, the number is transferred to the first line of generated coding.

If the receiving field of an ADD, INCREMENT, or MOVE instruction is defined in the tables and a maximum value for the item is specified, coding is added to make a suitable test for overflow. If the receiving field of a SUBTRACT or DECREMENT instruction is defined in the tables as less than a full word, coding is added to make a test for a negative result. Should either of these conditions exist in program execution, a PRINT statement is executed that shows which mnemonic reference caused the condition and an exit is made to statement 9999. A statement with this number should be included in the source program deck for program wrap-up or error recovery purposes.

2.1.1 ADD Instruction

This instruction contains two references and is interpreted as add Reference 1 to Reference 2. Reference 1 may be a table item, a variable, or an integer constant. Reference 2 may be a table item or a variable. The characters TO must be inserted between References 1 and 2.

2.1.2 SUBTRACT Instruction

This instruction contains two references and is interpreted as subtract Reference 1 from Reference 2. Reference 1 may be a table item, a variable, or an integer constant. Reference 2 may be a table item or a variable. If Reference 2 is a table item and occupies less than a full memory word, a test is made for a negative result which could adversely affect adjacent items in that word. This test is not applied if the item occupies a full word. The characters FROM must be inserted between References 1 and 2.

2.1.3 INCREMENT Instruction

This instruction contains one reference and is used to simplify counting. The integer value of one is added to the reference which may be a variable or a table item.

2.1.4 DECREMENT Instruction

This instruction contains one reference and is also used in counting. The integer value of one is subtracted from the reference which may be a variable or table item. If the reference is a table item and occupies less than a full memory word, a test is made for a negative result.

2.1.5 SET Instruction

This instruction contains one reference and is used to set a one-bit on-off indicator. If the reference is a variable or a table item which occupies a full word, it is set to the integer value of one. If the reference is a table item in a packed word, the rightmost bit of the field is set to one. If the item occupies more than one bit, the remaining bits are cleared to zeros.

2.1.6 CLEAR Instruction

This instruction contains one reference and complements the SET instruction. The reference may be a variable or a table item. If the reference is a variable or a table item which occupies a full word, it is set to zero. All bits of a packed table item are cleared, but the remainder of the word is not disturbed.

2.1.7 MOVE Instruction

This instruction contains two references and is interpreted as move Reference 1 to Reference 2. Reference 1 may be a table item, a variable, or an integer constant. Reference 2 may be a table item or a variable. This instruction causes the contents of Reference 2 to be replaced by those of Reference 1 which remains unchanged. The characters TO must be inserted between References 1 and 2.

2.2 EXAMPLES OF DATA MANIPULATION INSTRUCTIONS

The format of these instructions is essentially free form. To be recognized as a pseudoinstruction the letter P must appear in column 1. A statement number is optional and may contain up to five digits. It may follow directly after the letter P or be separated by one or more blanks. The command must be separated from the letter P or statement number by at least one blank. In addition to the seven listed commands, the abbreviations SUB, INC, and DEC may be used. The first reference must be separated from the command by at least one blank. When the command is MOVE, ADD or SUBTRACT, a delimiter word (TO or FROM) and a second reference must also be used, each separated by at least one blank. Reference names are limited to six characters. The remainder of the card may be freely used for comments. Some examples are:

```
P      ADD ITEM1 TO ITEM2
P      SUB     ITEM3   FROM ITEM4
P100   INC  ITEM5
P 1350 DEC ITEM7
P SET ITEM8
P      CLEAR      ITEM9
P      MOVE ITEM10 TO ITEM11        THIS PART OF THE CARD MAY BE USED FOR COMMENTS
```

2.3 OTHER USES OF DATA MANIPULATION INSTRUCTIONS

The use of these instructions is not limited to those items defined in input tables. Any program item which is defined in no more than six characters (including subscripts) may be used as a reference. It should be born in mind that the INCREMENT, DECREMENT, SET, and CLEAR instructions use integer values of 1 and 0. If the reference is a real number, inefficient coding may be generated. Real number representations (1.0, 0.0) may be used with the ADD, SUBTRACT, and MOVE instructions with no loss of efficiency.

SECTION 3 - TESTING DATA ITEMS

3.1 TEST PSEUDOINSTRUCTION

There is one pseudoinstruction for the testing of data items: TEST. This instruction has two forms: simple and complex. In either case, the format of the basic statement is the same. It contains two references separated by a relational operator. The references may be variables, table items, or integer constants.

3.1.1 Simple TEST Instructions

The simple TEST instruction duplicates the logical IF statement in that one statement may be executed based upon the results of the logical comparison. The executable statement may be a data manipulation type pseudoinstruction or any acceptable FORTRAN statement.

3.1.2 Complex TEST Instructions

The complex TEST instruction is similar to the IF statements found in COBOL or PL/I languages in that a series of instructions may be executed if the results of the logical comparison is a TRUE condition. Optionally, a different series of conditions may be executed if the comparison results in a FALSE condition. The executable statements may be acceptable FORTRAN statements or pseudoinstructions (except another complex TEST instruction).

3.2 EXAMPLES OF THE TEST INSTRUCTION

The format of the TEST instruction is essentially free form. The first part of the instruction is the same for either the simple or the complex test. To be recognized as a pseudoinstruction the letter P must appear in column 1. A statement number is optional and may contain up to five digits. It may follow directly after the letter P or be separated by one or more blanks. The command, references, and relational operator must be separated by at least one blank. If the test to be made is the simple form, the one executable statement must be included in the same card. If this statement is a pseudoinstruction, it must be of the data manipulation type and completely contained within

that card. The statement is free form and follows the same rules as a normal statement of this type. If the statement is an acceptable FORTRAN statement, it may be continued on a normal FORTRAN continuation card. Some examples are:

```
P      TEST ITEM1 NE ITEM2        GO TO 2050

P      TEST ITEM3 EQ 0                        WRITE (10,100) ITEM1, ITEM2,
      X (ARRAY(J),J=1,10)
 100  FORMAT (216,10F6.2)

P2050 TEST ITEM4 GT 100         P MOVE 100 TO ITEM4
```

If the test to be made is the complex form, only the TEST statement may be used on the first card. All FORTRAN or pseudoinstruction statements which follow this card are considered to be part of the test until a card specifying ENDTEST is encountered. This card is free form but must begin with the letter P in column 1. An example is:

```
P      TEST ITEM1 EQ ITEM2
P      MOVE ITEM3 TO ITEM4
       ALPHA=BRAVO**2
P      ENDTEST
```

If the test is to include different instructions for TRUE and FALSE conditions, the statements are separated by a card containing ELSE. This card is free form but must begin with the letter P in column 1. An example is:

```
P      TEST ITEM1 LT ITEM2
P      MOVE ITEM2 TO ITEM9
P      ELSE
P      MOVE ITEM1 TO ITEM9
P      ENDTEST
```

3.2.1 Compound Tests

There is no provision for a compound test using AND or OR operators within the TEST instruction. Compound tests can be made with little loss of program efficiency by placing data items in temporary variables using the MOVE instruction and executing FORTRAN logical IF statements.

3.2.2 Other Uses for the Test Instruction

The use of the simple or complex TEST instruction is not limited to those items defined in input tables. Any program item which is defined in no more than six characters (including subscripts) may be used as a reference. The complex TEST instruction in particular may be used to simplify coding.

SECTION 4 - MACRO INSTRUCTIONS

4.1 MACRO TYPE PSEUDOINSTRUCTIONS

There are two Macro type pseudoinstructions: INSERT and EXECUTE. Both call coding statements from definition tables and insert them into the source coding.

4.2 INSERT INSTRUCTION

The purpose of the INSERT instruction is to introduce nonexecutable statements into source coding. It is particularly useful in programs which use COMMON storage and have many subroutines. It is also useful in introducing identical DATA statements into several programs where data items must have the same definition, or for introducing identical FORMAT statements into programs which read or write the same files. The INSERT instruction may not have a statement number; however, any statement numbers included in the statements placed in the definition tables will be transferred to the source coding.

4.2.1 Examples of the INSERT Instruction

The format of the instruction is essentially free form. To be recognized as a pseudoinstruction, the letter P must appear in column 1. The command INSERT must be separated from the letter P by at least one blank. The reference name can be up to 31 characters and must be separated from the command by at least one blank. As an example of use, consider a large program with multiple subroutines which are entered frequently. To decrease program execution time, it is possible to place all subroutine arguments in COMMON storage as opposed to passing the arguments with each CALL statement. To accomplish this, appropriate COMMON statements must be used in the main program and all subroutines. An example might be:

```
      COMMON /COM1/ IRAY1(100,20), J1, K1, L1
      COMMON /COM2/ IRAY2(10), J2, K2, L2, IRAY3(50,3),
     X J3, K3, L3
```

To simplify this process using the FORTSIM program, the COMMON statements would be entered into the definition tables with an appropriate mnemonic reference, such as:

```
INSERT CMSTAT
      COMMON /COM1/ IRAY1(100,20), J1, K1, L1
      COMMON /COM2/ IRAY2(10), J2, K2, L2, IRAY3(50,3),
     X J3, K3, L3
END
```

The main program and each subroutine would contain a single pseudoinstruction:

```
P     INSERT   CMSTAT
```

the output from the program would then appear as:

```
C     INSERT   CMSTAT
      COMMON /COM1/ IRAY1(100,20), J1, K1, L1
      COMMON /COM2/ IRAY2(10), J2, K2, L2, IRAY3(50,3),
     X J3, K3, L3
```

4.3 EXECUTE INSTRUCTION

This instruction relieves the programmer of the tedious task of repetitive coding. It permits a sequence of statements (defined as a Macro) which have been placed in the definition table to be introduced into the source coding. Since the use of a given sequence of statements may not be identical in every application, the instruction allows up to 16 parameters to be passed from the EXECUTE statement into the coding of the Macro. This feature allows a generalized Macro to be tailored to a different precise application at each execution. Another feature of this function is the use of generated statement numbers. Consider the case where it would be desirable to branch from one part of the Macro to another line of coding also within the Macro. This could be accomplished by passing a new statement number into the Macro as a parameter in the EXECUTE card upon each execution; however, the Macro can be written with a symbolic statement number. The program will then assign a different unused statement number each time the Macro is executed. A Macro may have up to 30 symbolic statement numbers.

4.3.1 Examples of Macros and EXECUTE Instructions

For an example of how to construct and use a Macro, consider the case where a program has a series of 10 variables which must be tested to determine if the value of the variables are within prescribed limits after some iteration. If a variable is out of its range, it is desired to print a message stating the variable is not within its limits and then branch to some other part of the program which would apply correctional measures. The coding to accomplish one of these tests might be:

```
      IF (ITEM1.GE.10.AND.ITEM1.LE.100) GO TO 50
      PRINT 40
40    FORMAT (31H VARIABLE ITEM1 IS OUT OF RANGE)
      GO TO 2802
50    CONTINUE
```

In this coding the variable ITEM1 is tested for a value between 10 and 100. Similar coding would be required for all ten variables. This repetitive coding could be eliminated by constructing a Macro such as the following:

```
MACRO TEST-VARIABLE
      IF ($1$.GE.$2$.AND.$1$.LE.$3$) GO TO $*1$
      PRINT $*2$
$*2$  FORMAT (31H VARIABLE $1$ IS OUT OF RANGE)
      GO TO $4$
$*1$  CONTINUE
```

The first card names the Macro as TEST-VARIABLE. Parameters which are to be passed from the EXECUTE instruction card are indicated by a dollar sign-number-dollar sign (1) combination. In this case, the word ITEM1 should be the first parameter on the EXECUTE card and it would be inserted in the coding wherever 1 appears. Symbolic statement numbers are indicated by a dollar sign-asterisk-number-dollar sign ($*1$) combination. In this example two statement numbers would be inserted into the generated coding.

The format of the EXECUTE instruction card is similar to the other pseudo-instruction cards. The first character must be the letter P in column 1. If a statement number is used on this card, a CONTINUE statement bearing that number will be inserted before the first line of generated coding. The reference name for the Macro may be up to 31 characters in length and may contain special characters except a dollar

sign. It may not contain imbedded blanks. The parameters to be passed begin with a dollar sign, are separated by a single dollar sign and end with a dollar sign. As with other FORTRAN cards, this card ends at column 72, but may be continued with a special continuation card. This card must have the letter Q in column 1. It is scanned from columns 2 to 72 for additional parameters. The EXECUTE card required for the above example would appear as:

```
P     EXECUTE TEST-VARIABLE  $ITEM1$10$100$2802$
```

In this card the first parameter is the name of the variable to be tested, the second is the low value, the third is the high value.. and the fourth is the statement number to which a branch is made if the variable is out of range. The generated coding would be:

```
C     EXECUTE TEST-VARIABLE  $ITEM1$10$100$2802$
      IF (ITEM1.GE.10.AND.ITEM1.LE.100)  GO TO 10001
      PRINT 10002
10002 FORMAT (31H VARIABLE ITEM1 IS OUT OF RANGE)
      GO TO 2802
10001 CONTINUE
```

The use of a Macro for testing ten variables would require 16 cards (6 in the input tables and 10 EXECUTE cards) as opposed to 40 cards in straight line coding.

The length of any one parameter may be up to 72 characters and may contain imbedded blanks. Bear in mind in the construction of Macro statements that when a parameter is inserted, any coding to the right of the place where the parameter is to be inserted will be pushed further to the right. The program does not generate continuation cards, therefore any coding moved beyond column 72 is lost.

The statements used in Macros may be any acceptable FORTRAN statements or pseudoinstructions except the complex TEST instruction. A Macro may execute up to 50 other Macros but may not execute itself directly or indirectly. When the input table of Macros is read into the FORTSIM program, a recursive test is applied to each Macro. If it is found that a Macro will call itself, a warning message is printed and a flag is set. After checking all Macros, the program will terminate if this warning flag is set since a recursive Macro would cause looping within this program.

SECTION 5 - ASSEMBLING DEFINITION TABLES

5.1 TABLE STRUCTURE

There may be up to three types of inputs in the definition tables used by the FORTSIM program: Macro statements, INSERT statements and mnemonic reference definitions. The inputs may be in any order and, with the exceptions noted below concerning mnemonic references, may be intermixed. An example is given in Figure 5-1.

5.2 MNEMONIC REFERENCE TABLES

The format of mnemonic reference input cards is given in Table 5-1. In the definition tables, these cards must be grouped together and be headed by a card stating TABLES in columns 1 to 6 and be followed by a card with END in columns 1 to 3. More than one group may appear in the tables but each group must have the header and end cards. The names used as references may be any combination of six characters acceptable to the system, except the words TABLES, END, MACRO and INSERT. The names may not contain imbedded blanks. The program will accept up to 1024 references.

5.3 INSERT TABLES

Each group of cards to be inserted into a program must be headed by a card specifying INSERT in columns 1 to 6 and containing an identifying name of up to 31 characters. The name must be separated from the word INSERT by at least one blank, and may be any combination of acceptable characters without imbedded blanks. The cards to be inserted into the program may contain any acceptable FORTRAN coding and will be inserted exactly as entered without changes. A statement number appearing on an INSERT pseudoinstruction is ignored. The FORTSIM program will accept a maximum of 100 groups of cards and a total of 300 cards.

5.4 MACRO TABLES

Each group of cards comprising a Macro must be headed by a card specifying MACRO in columns 1 to 5 and containing an identifying name of up to 31 characters. The name must be separated from the word MACRO by at least one blank, and may

be any combination of acceptable characters except a dollar sign. It may not contain imbedded blanks. The types of instructions which may be used and the construction of Macros is detailed in Paragraph 4.3. The FORTSIM program will accept a maximum of 100 groups of cards and a total of 700 cards.

Table 5-1. Mnemonic Definition Card Format

CARD COLUMN	DESCRIPTION	EXPLANATION
1-6	Reference Name	Mnemonic reference of one to six characters, left justified.
7-8	Blank	--
9-36	Reference Address	Address definition of the reference. Must be acceptable to compiler in terms of array names, subscripting, etc.
37-43	Blank	--
44-45	Number of Bits	Contains the number of bits to be used by a data item, right justified. May be a value from 1 to 31 for IBM 360/370 or 1 to 35 for Honeywell 6050. If data item occupies a full word, field should be blank.
46	Blank	--
47-48	Starting Bit	Contains the number of the starting bit (left-most bit used) of a data item. The bits in a computer word are numbered from left to right starting with 0. May be a value from 0 to 31 for IBM 360/370 or 0 to 35 for Honeywell 6050. If data item occupies a full word, field should be blank.
49	Blank	--
50-59	Maximum value	Optional 10-digit entry which may specify a maximum value that a field may contain. Can be used to prevent one field from overflowing into an adjacent field. Right-justified.
60-80	Blank	--

```
MACRO INFO-DESCRIPTOR
P       MOVE   TIME   TO DR(1)
P       MOVE   $1$    TO DR(2)
P       MOVE   $2$    TO DR(3)
P       MOVE   $3$    TO DR(4)
        CALL #RITE (DESFIL,DR)
MACRO THROT-LTC
P       TEST       FELBA      GT    FETT      GO TO $1$
P       SET        FETI
P       EXEC   INFO-DESCRIPTOR     $6$NODNO$LTCNO$
MACRO LCHAN                                                              LCHAN
        CHNADD=LNKADD+LINKSZ*(CHANNO-1)*LINKCH
MACRO TCHANI                                                             TCHANI
        CHNADD=LNKAUD+TRIBSZ*(CHANNO-1)*2
INSERT SIMCOM                                                            SIMCOM
        COMMON /RAYS/ STORAY(80000),NODRAY(31,34)
INSERT SIZES                                                             SIZES
        DATA LINKSZ/6/,TRIBSZ/8/,LINKCH/2/,TRIBCH/1/,FSTLTC/29/,RTSIZE/2/,
       XENDCOR/80000/,NHSIZE/34/,VNTSZ/4/,MAXALT/15/
INSERT COMMON                                                            COMMON
        COMMON /VARS/ VNTADD,TIME,Z(40,2),                               COMMON
       XXPRESS,TRBFLG,DELAY,NOALT,NODUPS,MCB,NOETR,NODES,CARP,           COMMON
       XMSG-LEN,MSGNO,MSGTP,MSGPR,MSGSEC,RICNT,                          COMMON
       XBUGSWT,ALLALT,MAXALT,ALTCTR
TABLES
LTGS     STORAY(LNKADD   )          17 01
LTCA     STORAY(LNKADD   )          08 18
LTSN     STORAY(LNKADD   )          05 26
LTTN     STORAY(LNKADD   )          01 31
LTGE     STORAY(LNKADD+1)           17 01
LTCH     STORAY(LNKADD+1)           08 18
LTLTC    STORAY(LNKADD+1)           02 26
LTSC     STORAY(LNKADD+1)           03 28
LTLO     STORAY(LNKADD+1)           01 31
LTGP     STORAY(LNKADD+2)           17 05
LTOIP    STORAY(LNKADD+2)           01 22
LTNC     STORAY(LNKADD+2)           08 23
LTGF     STORAY(LNKADD+2)           01 31
LNON     STORAY(LNKADD+3)           05 10
LNAA     STORAY(LNKADD+3)           01 15
LNGHI    STORAY(LNKADD+3)           16 16      65535
LNGLO    STORAY(LNKADD+4)           16 16      65535
MPCVNT   NODRAY(NODNO,12)                      65535
NSQL     NODRAY(NODNO,13)           08 07
NSNT     NODRAY(NODNO,14)           17 15
MPORL    NODRAY(NODNO,14)           08 08
NSNL     NODRAY(NODNO,15)           16 16      32767
MPTQC    NODRAY(NODNO,15)           03 08
NSNLT    NODRAY(NODNO,16)           20 11
MPLTCD   NODRAY(NODNO,16)           01 31
NSNO     NODRAY(NODNO,16)           20 11
MPLTCX   NODRAY(NODNO,17)
END
```

Figure 5-1. Example of Definition Tables

SECTION 6 - PROGRAM RESTRICTIONS AND GUIDELINES FOR USE

6.1 PROGRAMMING RESTRICTIONS

In addition to the requirement for a statement numbered 9999 in the source deck as stated in Paragraph 2.1, the program may generate, just prior to the END card in the source deck, FORMAT statements which are numbered 9990 to 9994; therefore, these statement numbers must be reserved. The generated coding may use the integer variables ITMP1 and ITMP2; therefore, these variables should not be used in the source coding, as their contents may be altered by a pseudoinstruction. The program may use the variables MSK1 to MSK999, NM1 to NM999, and NMA1 to NMA999. When used, DATA statements are generated and inserted into the program. The program or subroutine END statement must begin in column 7.

6.2 GUIDELINES AND EXAMPLES OF PSEUDOINSTRUCTION USE

The following paragraphs provide examples of establishing input definition tables and use of pseudoinstructions in program source coding.

6.2.1 Using Bit Manipulation and Indirect Addressing

Indirect addressing can simplify program coding while bit manipulation can materially reduce the memory requirement for program execution.

As an example of how these functions may be used, consider the following processing which might occur in a hypothetical program. It is desired to develop a matrix containing selected data concerning subscribers to the Defense Communications System. The data will be extracted from a data base. The matrix will contain a four-character tributary name, an integer value indicating type of service (1=AUTOVON, 2=AUTODIN, 3=both AUTOVON and AUTODIN), an integer alue from 1 to 8 indicating AUTODIN tributary line and equipment speed, and an integer value from 1 to 2000 indicating the number of AUTOVON access lines.

In standard FORTRAN coding, the program might use for the matrix an array named ISUBS dimensioned at 3000 by 4. As each subscriber is identified in the data base, a pointer named NUMSUB will be incremented and used as the first subscript for the array. In filling the matrix, each of the four secondary levels of the array would be identified in the coding as follows:

Subscriber Name = ISUBS (NUMSUB, 1)

Type of Service = ISUBS (NUMSUB, 2)

AUTODIN Speed = ISUBS (NUMSUB, 3)

AUTOVON Lines = ISUBS (NUMSUB, 4)

6.2.1.1 Example of Indirect Addressing

Instead of referring to each of the four secondary levels in the manner previously illustrated, coding may be simplified by using mnemonic references and pseudoinstructions. The direct address shown requires 15 characters to write in the coding, and further, the address is not synonymous with the data it contains. If the data items were identified with the following mnemonic references:

Subscriber Name = SNAME

Type of Service = SERV

AUTODIN Speed = DINSPD

AUTOVON Lines = VONLIN

the entries in the definition table would appear as:

```
TABLES
SNAME   ISUBS(NUMSUB,1)
SERV    ISUBS(NUMSUB,2)
DINSPD  ISUBS(NUMSUB,3)
VONLIN  ISUBS(NUMSUB,4)
END
```

Filling the matrix with the data items is now simplified. If, for example, it was determined that a particular subscriber was served by both AUTODIN and AUTOVON, this would be coded in FORTRAN as:

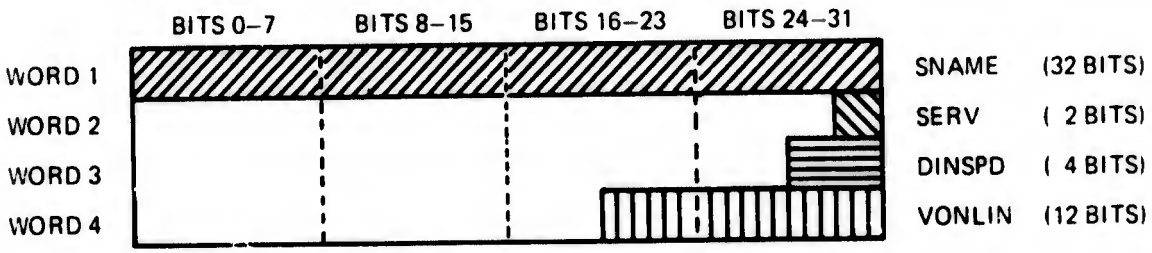
```
ISUBS(NUMSUB,2) = 3
```

Using a pseudoinstruction, the same operation is accomplished with:

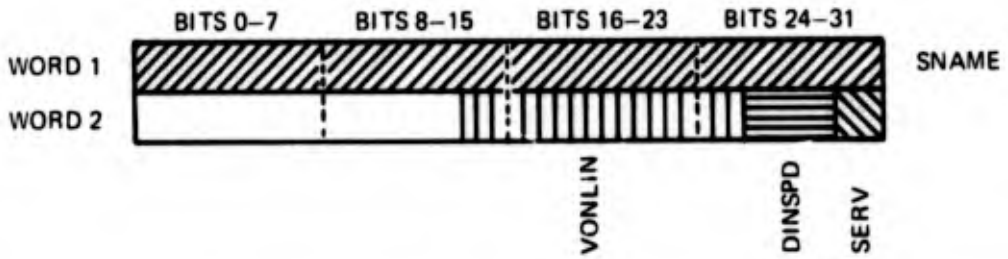
```
P       MOVE      3           TO    SERV
```

6.2.1.2 Example of Bit Manipulation

In the foregoing examples, four words (16 bytes) of IBM 360/370 memory would be required for each subscriber entry. By examining the maximum values of each of the data items, it can be seen that no more than 50 bits would ever be used.



With the use of bit manipulation, the same amount of data could be stored in two words (8 bytes) with 14 bits left over for expansion.



To accomplish the bit packing, the only program source coding change required would be to redimension the array ISUBS to 3000 by 2. The input tables to the FORTSIM program would be changed to reflect the new address and bit structure. The input table would appear as follows:

```
TABLES
SNAME    ISUBS(NUMSUB,1)
SERV     ISUBS(NUMSUB,2)          02 30
DINSPD   ISUBS(NUMSUB,2)          04 26
VONLIN   ISUBS(NUMSUB,2)          12 14
END
```

The pseudoinstructions in the source coding would remain the same in all cases.

6.2.2 Other Considerations

Bit packing may only be used with integer values (including characters). Real number values require a full word for floating point representation. Indirect addressing may be used with real numbers.

While bit packing reduces the amount of memory required, it may increase program execution time due to the additional extract and shift instructions required. The amount of additional execution time may be reduced by proper placing of the data items within the packed word. An item aligned at bit 31 requires only an extract instruction; therefore, the items most frequently referenced should be aligned at that point. Also, the time required to shift an item is determined by the number of bits it must be shifted. A shift of two bits takes less time than a shift of 16 bits. When two or more items are accessed with the same frequency, the smaller items should be aligned to the right of the word.

Bit 0 in a computer word is normally the sign bit and indicates whether the remaining bits represent a positive or negative number. The use of this bit in a packed word can cause problems depending upon a computer system's compiler and arithmetic hardware. It is advisable to avoid the use of this bit if the data item will be acted upon by arithmetic instructions (ADD, SUBTRACT, INCREMENT, DECREMENT).

In processing pseudoinstructions other than INSERT or EXECUTE, the program searches the definition tables for a match with the references in the instruction. If no match is found, the program assumes the reference to be a variable or constant and generates instructions accordingly. In a large program, misspelled references or references omitted from the definition tables may be overlooked. It is recommended that the XREF option in the IBM 360/370 be included when executing the FORTRAN compiler and the variable listing be examined to determine if any items which should be treated as mnemonic references are being treated as variables. This process may be further simplified by choosing mnemonic references which would be unacceptable to the compiler as a variable. This could be accomplished by using references in which the first character is not alphabetic or one which contains a special character.

6.3 PROGRAM OPERATION AND OUTPUTS

The first step of the FORTSIM program is to read and verify the Definition Table file. A listing of the file including a card sequence number is produced. An example of this listing is given in Figure 6-1. If an error is detected, such as a recursive Macro or an invalid bit structure in a packed mnemonic reference, an appropriate diagnostic message is printed and the program is terminated upon completion of that step. The second step of the program reads the Source Coding file. As each card is read, it is listed along with its sequence number. An example is given in Figure 6-2. If an error is encountered, such as an undefined pseudoinstruction or an undefined Macro name, an appropriate diagnostic message is printed. An output file is also created in which pseudoinstructions have been converted to acceptable FORTRAN instructions. This file becomes the input to the FORTRAN compiler and would normally be written as a temporary data set. In the IBM 360/370 version of the program, a System Return Code of 8 is issued by the program when serious errors are encountered so that the execution of the compiler may be inhibited. If warning messages are printed, a Return Code of 4 is issued. Warning messages are printed if the program does not encounter an END card in the Source Coding file, if a parameter in an EXECUTE card exceeds 72 characters, or if a blank card is generated as the result of a Macro execution. Similar provisions are made in the Honeywell 6000 version of the program.

6.3.1 Program Options

The first option suppresses the listing of the input tables and source code deck normally provided by the program and may be used when only the listing provided by the compiler is desired. When this option is exercised and an error is detected in the Definition Table or Source Code files, the option is cancelled and a listing begins at that point. The Option card specifies NO LIST.

The second option lists an intermediate source coding in which Macro instructions have been expanded, and INSERT statements have been added. This listing contains sequence numbers. The Option card specifies INTERMEDIATE. An example is given in Figure 6-3.

The third option edits all comment cards out of the source coding output file which is normally passed to the compiler. This option is available to reduce the bulk of the compiler listing and reduce running time. The Option card specifies EDIT.

The fourth option is a listing of table references, variables, and constants which have been used in pseudoinstructions. The names used are cross-referenced to the sequence numbers appearing on the intermediate source code listing. This option automatically generates the intermediate listing. At the end of the main program and each subroutine, two additional sections are printed, one with variables and constants cross-referenced, the other with only table references. These sections are illustrated in Figure 6-4. This option assists in detecting misspelled references. The Option card specifies XREF. The Option card format is given in Table 6-1.

6.4 PROGRAM EXECUTION

Examples of program execution are provided in Reference 2. The FORTSIM program must read the Source Code and Definition Table files twice, once to obtain a list of statement numbers used in the source coding (and thereby know which may be used for generated statement numbers), and once for processing. These files must therefore reside on permanent or temporary data sets for proper program operation. They may not be entered into the program as "instream data."

Table 6-1. Option Card Format

CARD COLUMN	DESCRIPTION	EXPLANATION
1-12	Option Selection	Contains XREF, NO LIST, EDIT, or INTERMEDIATE
13-80	Blank	--

```
                                DEFINITION TABLE LISTING                                    SEQUENCE
                                                                                                    1
INSERT SIMCOM                                                                                       2
      COMMON /RAYS/ STORAY(80000),NODRAY(31,34)                                     SIMCOM          3
INSERT COMMON                                                                                       4
      COMMON /VARS/ VNTADD,TIME,MSADD,HIADD,LTCADD,NODNO,LTCNO,FSTLTC,              COMMON          5
     X DR(4)                                                                        COMMON          6
MACRO INFO-DESCRIPTOR                                                                               7
P     MOVE  TIME  TO DR(1)                                                                          8
P     MOVE  $1$   TO DR(2)                                                                          9
P     MOVE  $2$   TO DR(3)                                                                         10
P     MOVE  $3$   TO DR(4)                                                                         11
      CALL WRITE (DESFIL,DR)                                                                       12
MACRO THROT-LTC                                                                                    13
P     TEST      FELRA     GT   FETT     GO TO $1$                                                  14
P     SET       FETI                                                                             15
P     EXEC  INFO-DESCRIPTOR    $6$NODNO$LTCNO$                                                     16
MACRO SETLTC                                                                        SETLTC         17
      LTCADD=FSTLTC+LTCNO                                                                          18
TABLES                                                                                             19
LTLTC    STORAY(LNKADD+1)                    02 26                                                 20
TROFI    STORAY(LNKADD+6)                    01 16                                                 21
TRINO    STORAY(LNKADD+5)                    17 15                                                 22
TRIO     STORAY(LNKADD+3)                    16 16      65535                                      23
MSLR     STORAY(MSADD+2)                     17 15                                                 24
MSLN     STORAY(MSADD+1)                     09 23                                                 25
FVAL     STORAY(VNTADD   )                                                                         26
FVLK     STORAY(VNTADD+1)                                                                          27
MRIOF    NODRAY(NODNO,5)                                                                           28
MRIOS    NODRAY(NODNO,6)                     01 31                                                 29
MRNLTC   NODRAY(NODNO,17)                                   65535                                  30
NSOIT    NODRAY(NODNO,9)                                                                           31
FETT     NODRAY(NODNO,19)                    20 09                                                 32
FELRA    NODRAY(NODNO,LTCADD)                20 11                                                 33
FETI     NODRAY(NODNO, LTCADD)               01 31                                                 34
END
```

Figure 6-1. Sample Definition Table Listing

```
                                    SOURCE DECK LISTING                    SEQUENCE

      SUBROUTINE NXTMIN                                                        1
      IMPLICIT INTEGER (A-Z)                                                   2
P     INSERT   SIMCOM                                                          3
P     INSERT   COMMON                                                          4
P     TEST     MPNLTC    EQ   1          GO TO 8560                           5
P     MOVE     LTLTC     TO   LTCNO                                           6
P     EXECUTE  SETLTC                                                        7
P     TEST     FFTI      NE   1          GO TO 8560                           8
C     CHANNEL IS THROTTLED. PUT IN INPUT QUEUE UNLESS ALREADY THERE          9
P     TEST     TROFI     EQ   1          RETURN                               10
P     SET      TROFI                                                         11
      CALL MOLES1                                                            12
      VNTADD=H1ADD                                                           13
P     MOVE     LNKADD    TO   EVLK                                          14
P     TEST     MPIOS     EQ   0                                             15
P     MOVE     VNTADD    TO   MPIOS                                         16
P     ELSE                                                                   17
P     MOVE     MPIOE     TO   VNTADD                                        18
P     MOVE     H1ADD     TO   EVBL                                          19
      VNTADD=H1ADD                                                           20
P     ENDTEST                                                                21
P     MOVE     VNTADD    TO   MPIOE                                         22
      RETURN                                                                 23
PH560 MOVE     TRINO     TO   MSADD     SET MSG TRANSMITT                   24
P     MOVE     MSLR      TO   TRINO                                         25
      CALL SNDTMG                                                            26
P     DEC      TRIQ                                                          27
P     DEC      NSQIT                                                         28
C     NOW ADJUST LTC IF NECESSARY                                            29
P     TEST     MPNLTC    EQ   1          RETURN                               30
P     SUB      MSLN      FROM FELBA                                         31
P     EXECUTE THROT-LTC  $8570$                                             32
 8570 RETURN                                                                 33
      END                                                                    34
```

Figure 6-2. Sample Source Code Listing

```
                                INTERMEDIATE SOURCE FILE LISTING                                SEQUENCE
      SUBROUTINE NXTMIN                                                                             1
      IMPLICIT INTEGER (A-Z)                                                                        2
C     INSERT   SIMCOM                                                                               3
      COMMON /RAYS/ STORAY(80000),NODRAY(31,34)                                    SIMCOM           4
C     INSERT   COMMON                                                                               5
      COMMON /VARS/ VNTADD,TIME,MSADD,H1ADD,LTCADD,NODNO,LTCNO,FSTLTC,             COMMON           6
     X DR(4)                                                                       COMMON           7
P     TEST     MPNLTC     EQ   1                                                                   8
     *GO TO 8560                                                                                    9
P     MOVE     LTLTC      TO   LTCNO                                                             10
C     EXECUTE  SETLTC                                                                            11
      LTCADD=FSTLTC+LTCNO                                                          SETLTC           12
P     TEST     FETI       NE   1                                                                  13
     *GO TO 8560                                                                                   14
C     CHANNEL IS THROTTLED. PUT IN INPUT QUEUE UNLESS ALREADY THERE                                 15
P     TEST     TROFI      EQ   1                                                                  16
     *RETURN                                                                                       17
P     SET      TROFI                                                                             18
      CALL HOLES1                                                                                19
      VNTADD=H1ADD                                                                               20
P     MOVE     LNKADD     TO   EVLK                                                              21
P     TEST     MPIOS      NE   0          GO TO 10001                                            22
P     MOVE     VNTADD     TO   MPIOS                                                             23
      GO TO 10002                                                                                24
10001 CONTINUE                                                                                   25
P     MOVE     MPIOF      TO   VNTADD                                                            26
P     MOVE     H1ADD      TO   EVAL                                                              27
      VNTADD=H1ADD                                                                               28
10002 CONTINUE                                                                                   29
P     MOVE     VNTADD     TO   MPIOE                                                             30
      RETURN                                                                                     31
P8560 MOVE     TRINO      TO   MSADD      SET MSG TRANSMITT                                      32
P     MOVE     MSLR       TO   TRINO                                                             33
      CALL SNDTMG                                                                                34
P     DEC      TRIQ                                                                              35
P     DEC      NSQIT                                                                             36
C     NOW ADJUST LTC IF NECESSARY                                                                37
P     TEST     MPNLTC     EQ   1                                                                 38
     *RETURN                                                                                     39
P     SUB      MSLN       FROM FELBA                                                             40
C     EXECUTE THROT-LTC  $8570$                                                                  41
P     TEST     FELBA      GT   FETT      GO TO 8570                                             42
P     SET      FETI                                                                              43
C     EXEC  INFO-DESCRIPTOR    $6$NODNO$LTCNO$                                                   44
P     MOVE     TIME       TO   DR(1)                                                             45
P     MOVE     6          TO   DR(2)                                                             46
P     MOVE     NODNO      TO   DR(3)                                                             47
P     MOVE     LTCNO      TO   DR(4)                                                             48
      CALL WRITE (DESFIL,DR)                                                                     49
 8570 RETURN                                                                                     50
      END                                                                                        51
```

Figure 6-3. Sample Intermediate Source Code Listing

CROSS-REFERENCE LISTING OF TABLE REFFRENCES

FVAL	27	
FVLK	21	
FELBA	41	42
FETI	13	43
FETT	42	
LTLTC	10	
MPIOF	26	30
MPIOS	22	23
MPNLTC	8	38
MSLN	40	
MSLP	33	
MSQIT	36	
TRINO	32	33
TRIO	35	
TROFI	16	18

CROSS-REFERENCE LISTING OF VARIABLES AND CONSTANTS

DR(1)	45			
DR(2)	46			
DR(3)	47			
DR(4)	48			
H1ADD	27			
LNKADD	21			
LTCNO	10	48		
MSADD	32			
NODNO	47			
TIME	45			
VNTADD	23	26	30	
0	22			
1	8	13	16	38
6	46			

Figure 6-4. Sample Cross Reference Listings

REFERENCES

1. Description of the Simulation Models of the Defense Communications System Performance Simulator, Defense Communications Agency System Engineering Facility Technical Note No. 6-73, March 1973.

2. Cataloged Procedures, CSC Report R4195950-2-2, November 1973.

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)
Computer Sciences Corporation
6565 Arlington Blvd.
Falls Church, Virginia 22046

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b GROUP

3 REPORT TITLE

FORTRAN SIMULATION (FORTSIM) LANGUAGE

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)
Program Description

5 AUTHOR(S) (First name, middle initial, last name)

Richard D. Crumm

6 REPORT DATE	7a. TOTAL NO OF PAGES	7b. NO. OF REFS
December 1973	35	2

8a CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
DCA100-73-C-0033	R4195950-3-1
b. PROJECT NO	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
c.	
d.	

10 DISTRIBUTION STATEMENT
Distribution limited to U.S. Government agencies only, Test and Evaluation December 1973. Other requests for this document must be referred to the Defense Communications Agency.

11 SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
NONE	Defense Communications Agency Code R940 Washington, D.C. 20305

13 ABSTRACT

This report contains a detailed description of the concept and use of the FORTRAN Simulation (FORTSIM) Language program which extends FORTRAN language to accommodate large scale simulation models. It permits greater utilization of available memory, introduces a Macro facility similar to those available in machine language compilers, and provides programming ease with the use of indirect addressing and a method of assuring commonality between programs and subroutines. The program is executable on the IBM 360/370 and Honeywell 6000 Series computer system.

DD FORM 1473

Unclassified
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
FORTRAN Simulator Preprocessor						
Bit Manipulation						
Macro facility						
Indirect addressing						
Program commonality						