-FORTSIM-
SIMULATION USING STRUCTURED FORTRAN PLUS TABLE MANAGEMENT

Richard D. Crumm and Sen-Lang Wang
Computer Sciences Corporation, Falls Church, Virginia

Edward H. Cooper
Defense Communications Engineering Center, Reston, Virginia

INTRODUCTION

The Defense Communications System (DCS) Performance Simulator developed by Computer Sciences Corporation for the Defense Communications Agency (DCA) is composed of four discrete-event and two analytical models which are used in the management of voice and data communication networks under the operational control of DCA. Some of the models have been in existence for more than 10 years and were originally implemented using assembly language. When the DCA computing systems were upgraded to third generation machines, it was necessary to convert the models to a higher order language. Because of its large size, the modeling of the DCS prohibited the use of a simulation language such as Simscript. FORTRAN was selected as the principal language because of its extensive use, wide acceptance, and reasonable standardization among manufacturers.

Several limitations associated with FORTRAN led the DCA to investigate new techniques for simplifying the writing of large programs in this language. The principal limitations for large scale modeling are:

- The general structuring of FORTRAN coding, while suitable for scientific programming, is somewhat limited when used for applications such as modeling.
- The subscripted array concept used by FORTRAN for memory addressing is not readily adaptable to the management of the numerous status tables inherent to large scale discrete event simulation models.
- Since FORTRAN is a word oriented language, it can have massive memory storage requirements. A simple yes or no indicator which can be represented by a 1 or 0 in one bit of memory requires a full word of storage. Although most manufacturers supply instructions to facilitate bit manipulation, these instructions are not standardized.

The primary objectives in implementing the FORTRAN Simulation (FORTSIM) program were to:

- Provide structured programming instructions
- Facilitate the integration of program segments into model systems
- Reduce redundant coding by including a Macro capability
- Permit program development and debugging on a full word basis
- Provide a bit manipulation scheme which would permit source coding to be relatively machine independent
- Provide readability of source code.

STRUCTURED PROGRAMMING INSTRUCTIONS

FORTSIM allows the use of four instructions which facilitate the top-down, structured programming approach when using the FORTRAN language. The instructions are: IF (...) THEN... ELSE, WHILE (...), UNTIL (...), and DO SEGMENT XXXXX.

IF (...) THEN Instructions

The IF instruction is usable in two forms:

```
IF (...) THEN     or     IF (...) THEN
...                      ...
...                      ELSE
ENDIF                    ...
                         ENDIF
```

In the first case the coding between THEN and ENDIF is executed whenever the logical condition is TRUE, and in the second case, the coding between THEN and ELSE is executed for TRUE and that between ELSE and ENDIF for FALSE. These instructions may be nested (up to 50 deep) bearing in mind that the ELSE and ENDIF instructions are not labeled so as to relate them to a particular IF instruction. In processing these instructions, the program considers them related to the unterminated IF (...) THEN instructions immediately preceding them in the sequential flow of coding. An example of nesting would be:

```
┌ IF (...) THEN
│ ...
│┌ IF (...) THEN
││ ...
│├ ELSE
││ ...
│└ ENDIF
│ ...
│ ENDIF
└
```

Control may be transferred to statements within the range of the IF ... ENDIF instructions. Program flow then will be the same as though the logical expression were TRUE (or FALSE, if entered following an ELSE).

WHILE (...) Instructions

The form of this instruction is:

```
WHILE (...)
...
...
...
ENDWHILE
```

The execution of this set of instructions is similar to a set of IF (...) THEN ... ENDIF instructions except that upon reaching the ENDWHILE instruction, control returns to the WHILE instruction, where the logical expression is again evaluated. The program will remain in this loop until the condition is FALSE, at which time control is transferred to the statement following the ENDWHILE instruction. WHILE (...) instructions may be nested (up to 50) in the same manner as IF (...) THEN.

Control may be transferred into the middle of a WHILE ... ENDWHILE loop; however, control will not fall through the

ENDWHILE instruction but will be passed to the WHILE (...) instruction.

If the condition is FALSE the first time tested, the loop is not executed.

UNTIL (...) Instructions

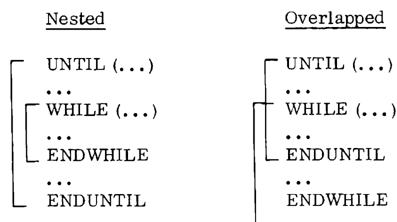
The form of this instruction is:

```
UNTIL (...)
...
...
...
ENDUNTIL
```

The execution of this set of instructions is similar to the WHILE (...) set except that the loop is not executed when the condition is TRUE but passes to the instruction following the ENDUNTIL. Nesting and passing control into the loop are the same. If the condition is TRUE the first time it is tested, the loop is not executed.

Combinations of IF (...) THEN, WHILE (...), and UNTIL (...) Instructions

IF (...) THEN, WHILE (...), and UNTIL (...) instructions may be nested in the same manner as DO loops. Care must be taken to be certain that loops do not overlap.



The overlapped coding in the preceding example could produce unpredictable results depending upon the logical expressions.

When several nested loops have the same common end, an ENDALL statement may be used in place of multiple ENDIF, ENDWHILE, or ENDUNTIL statements. This statement will terminate all open loops regardless of type in the following order: all IF (...) THENs, all UNTILs (...), and all WHILEs (...). If the nested loops are not all the same type, overlapped coding could result.

Segmented Coding

The use of segmented coding can often eliminate the use of subroutines which consume execution time for linkage. It can also be used to reduce redundant code. Sections of coding may be designated as segments in the following manner:

```
...
SEGMENT XXXXX
...
...
...
END SEGMENT XXXXX
...
```

The code so designed may then be executed by "falling through" the SEGMENT statement or from other parts of the program with the statement DO SEGMENT XXXXX.

In any main program or subroutine, up to 50 sections of code may be designed as segments and up to 45 DO SEGMENT instructions may be used for each segment. Areas of code designated as a segment should be reasonably straight line code; they may be nested but may not overlap. Related SEGMENT and DO SEGMENT instructions must be within the same main program or subroutine. The segment name may contain up to 31 alphanumeric characters, including special characters (except semicolons, vertical bars, or question marks).

MACRO INSTRUCTIONS

The use of Macros can relieve the programmer of the tedious task of repetitive coding. FORTSIM permits a sequence of previously defined statements to be introduced into the source coding. Since the use of a given sequence of statements may not be identical in every instance, up to 50 parameters may be passed from the calling statement into the stored statements, tailoring the Macro to a precise application.

For an example of how to construct and use a Macro, consider where several three-dimensional arrays are to be cleared. Assume that IARRAY is dimensioned (10, 20, 30); then, the coding to clear the array might be:

```
      DO 10 I = 1,10
      DO 10 J = 1,20
      DO 10 K = 1,30
   10 IARRAY (I,J,K) = 0
```

Similar coding would be required for the clearing of each array. This repetitive coding could be eliminated by constructing a Macro such as:

```
   MACRO CLEAR-ARRAY

         DO  $*1$ I = 1,$2$
         DO  $*1$ J = 1,$3$
         DO  $*1$ K = 1,$4$
   $*1$  $1$ (I,J,K) = 0
```

The first line names the Macro CLEAR-ARRAY. A dollar sign-number-dollar sign (1) combination indicates that parameters are to be passed from the source coding. For this Macro to be executed more than once, the statement label used at the foot of the DO loop must be different for each execution. The dollar sign-asterisk-number-dollar sign combination ($*1$) is a symbolic statement number which signals the FORTSIM program to supply a unique generated statement number at each execution. Up to 30 of these symbolic statement numbers may be used in each Macro.

The instruction used in the source coding to execute the preceding example would be:

```
   MACRO   CLEAR-ARRAY   $IARRAY $ 10 $ 20 $ 30 $
```

The format of the card is free form. At least one blank column must appear between the word MACRO and the Macro name, and between the Macro name and the first parameter. The Macro name may be up to 31 characters in length and may contain special characters.

To provide adequate flexibility, Macro coding may call other Macros, which in turn may call other Macros, etc. While this feature provides flexibility, it also provides the possibility of endless looping in the FORTSIM program if the Macros are recursive (Macro A calls B, B calls C, C calls A). To eliminate this possibility, the program allows up to 50 imbedded Macro calls for any Macro call appearing in the source coding.

TABLE MANAGEMENT

FORTSIM uses the data dictionary approach to provide an indirect addressing capability. Meaningful names are assigned to all table items. These names, called reference names or references, are used in the source coding in place of array

names, etc. Using a supermarket problem as an example, consider those items which might appear in a status table to describe the checkout counters.

Entry Description	Reference Name	Address
Counter open flag	CCOPEN	ICCRAY (1, CCNUM)
Average time to ring item	CCRING	ICCRAY (2, CCNUM)
Average time to bag each item	CCBAG	ICCRAY (3, CCNUM)
Average time to collect money	CCRIP	ICCRAY (4, CCNUM)
Number of people in queue	CCQUE	ICCRAY (5, CCNUM)

In normal FORTRAN coding, the addresses shown on the right would be used for access to the status table. Using FORTSIM, the reference name is all that is needed in source coding, once the variable CCNUM is set to the desired value. The statement

IF (CCOPEN. EQ. 1) THEN ...

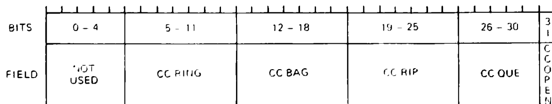
is simpler to write and more meaningful than:

IF (ICCRAY (1, CCNUM). EQ. 1) THEN ...

Also, consider the maximum values to be used in the fields. The following might be representative values:

Field	Maximum Value	Bits Required
Counter open flag	1	1
Average time to ring each item (in tenths of seconds)	100	7
Average time to bag each item (in tenths of seconds)	100	7
Average time to collect money (in seconds)	120	7
Number of people in queue	30	5

The total number of bits required is 27 which will fit into one word of storage on most machines. If the five items were to be packed into one word on an IBM 360 or 370, the word layout might look like:



The dimension statement could now be:

DIMENSION ICCRAY (10)

thus permitting an 80-percent reduction in memory space for this table. The input tables to FORTSIM might appear:

Reference Name	Reference Address	Number of Bits	Starting Bit
CCOPEN	ICCRAY (CCUNM)	1	31
CCQUE	ICCRAY (CCNUM)	5	26
CCRIP	ICCRAY (CCNUM)	7	19
CCBAG	ICCRAY (CCNUM)	7	12
CCRING	ICCRAY (CCNUM)	7	5

Whether the tables are defined on a full word or a packed-bit basis, the source coding remains the same.

Use of Reference Names in Source Coding

Reference names may be used freely in normal arithmetic statements, logical and arithmetic IF statements, and WHILE and UNTIL instructions. Reference names may not be subscripted; however, those which are assigned integer values may be used as subscripts. Reference names may be used in the list of INPUT/OUTPUT statements and as the initial, test, or incremental values of a DO LOOP index. They may also be used in an implied DO LOOP in an I/O statement.

In certain cases, the generated coding created by FORTSIM contains a variable with the same designation as a reference name. For example, when a reference is used as a subscript, the contents of the addressed memory location (which may itself be subscripted and bit-packed) must be moved to a variable to be properly executed. For this reason, the naming of references should follow the same rules as variables in regard to real and integer designations.

If references are to be used in subroutines or functions, it is necessary that the indirect address definitions (array ICCRAY and table pointer CCNUM in the preceding examples) be made available. This can be accomplished by passing them as arguments; however, the recommended procedure is to place all such arrays and pointers in COMMON. (This can be simplified by constructing a Macro which contains the COMMON statement(s) and calling it in the main program and every subroutine or function.) For this reason, no provision has been made to use references as arguments in CALL, SUBROUTINE, ENTRY, or FUNCTION statements.

DATA MANIPULATION INSTRUCTIONS

To provide source coding with some readability not inherent to FORTRAN and to reduce coding, seven data manipulation instructions are available which may be used in place of arithmetic statements. The instructions are: ADD, SUB(tract), MOVE, SET, CLEAR, INC(rement), and DEC(rement).

The format of the ADD, SUB, and MOVE instructions are:

ADD (sending field) TO (receiving field)
SUB (sending field) FROM (receiving field)
MOVE (sending field) TO (receiving field)

The instructions are free form where blanks are significant as field delimiters. The sending fields of these instructions are not limited to a single value, but may be any expression which could appear to the right of an equal sign in an arithmetic statement (no imbedded blanks). The receiving field may be a reference name, variable, or array element. It may also be multiply defined. The instruction

ADD AAAA TO BBBB CCCC DDDD

will produce the following FORTRAN statements

```
BBBB = BBBB + AAAA
CCCC = CCCC + AAAA
DDDD = DDDD + AAAA
```

The INC and DEC instructions add or subtract the integer value of one. The SET and CLEAR instructions equate the receiving field with integer one or zero. The receiving fields of these instructions may also be multiply defined. The formats are:

INC	JJJJ	KKKK	LLLL
DEC	JJJJ	KKKK	LLLL
SET	JJJJ	KKKK	LLLL
CLEAR	JJJJ	KKKK	LLLL

There is no numerical limit placed on the number of receiving fields; however, the complete instruction must be contained on two cards, i.e., not more than one continuation card is permitted.

FORTRAN STATEMENTS

Normal FORTRAN statements are passed by the FORTSIM program to the output data set which is the input to the compiler. Most statements are examined for reference names which must be replaced with the coding which provides true addresses and bit manipulation. The following is a summary of the variations permitted by the FORTSIM program.

Statement Labels

Statement labels may be symbolic names as well as standard numerics. Symbolic labels may be up to 31 characters and may be any combination of letters, numbers, and special characters (except semicolons, vertical bars, or question marks). They may not contain imbedded blanks. The first character should be alphabetic; however, if the first character is a C, it should never be punched in column 1 since that card would be treated as a comment card. When used as a label, the symbolic name must begin between columns 1 and 5.

IF Statements

Statement labels in arithmetic IFs may be symbolic. The imperative expression in a logical IF may use reference names following the rules of the particular statement type. Since the use of references may cause more than one line of generated coding, some logical IFs are converted with GO TO and CONTINUE statements.

The imperative expression of a logical IF statement may not be a MACRO, ELSE, ENDIF, WHILE, ENDWHILE, UNTIL, ENDUNTIL, ENDALL, SEGMENT, or END SEGMENT instruction. The program will also reject another IF statement or the start of a DO loop. A DO SEGMENT instruction is acceptable. All of these instructions may be included within IF (...) THEN... ELSE... ENDIF sets.

GO TO and ASSIGN Statements

Unconditional, Computed, and Assigned GO TO statements may contain symbolic statement labels. The variable in a Computed GO TO may be a reference name. The label in an ASSIGN statement may be symbolic.

Card Format

The average FORTRAN statement uses less than half of a punched card which results in wasted space in source code files. In the source code input to FORTSIM, cards may contain more than one statement or may contain comments following the statement(s). Multiple statements may be punched into a single card by separating the statements with a semicolon or vertical bar; however, any statement which requires a label must begin on a new card using columns 1 to 5 for the label. To place comments in a card, the coding may be terminated with a question mark and the remainder of the card will be ignored by FORTSIM. Since DATA and FORMAT statements could normally contain the separator characters, these cards are not scanned and may not contain comments or be part of a multiple statement card. Reference names appearing in DATA statements are treated as program variables. Also, any card with the letter C in column 1 is considered a comment and is ignored by FORTSIM.

PROGRAM IMPLEMENTATION

The FORTSIM program is written in ANSI COBOL and is implemented on an IBM 370/155. Its memory requirements are adjustable based on the number of references and Macros desired. The current requirement of 224K (needed by the level H FORTRAN compiler) will allow the following:

- 150 Macros with up to 800 cards
- 350 reference names
- 250 symbolic statement labels
- 50 segments.

The program should be executable on any system which supports ANSI COBOL (minor changes may be needed in the Environment Division) with all features usable except the bit manipulation. For this, program modifications are necessary for systems which do not have the SHFTL, SHFTR, and LAND instructions found in the Extended Logic section of the IBM 360/370 level H FORTRAN compiler. Previous versions of FORTSIM (without structured instructions) have been implemented on the UNIVAC 1108 and Honeywell 6050 systems.

The FORTSIM Macro features have been compiled into a separate program, Macro Generator. It is used principally with COBOL programs but may be used with FORTRAN, PL/I, ALC, or any card-oriented language.

COMPARISONS OF FORTRAN AND FORTSIM CODING

A project currently in progress at Computer Sciences Corporation is the use of structured programming and FORTSIM Macro features to convert a Satellite Channel Model from UNIVAC 1108 to IBM 370/155 operation. The following is a comparison of the before and after coding of one main program:

	FORTRAN	FORTSIM
Statement numbers (excluding FORMATS and DO LOOPS)	114	30
GO TO Statements	43	16
COMPUTED GO TO Statements	17	0
IF(...) GO TO Statements	52	24
IF(...) THEN Instructions	0	31
IF(...) THEN ... ELSE Instructions	0	29
Source Coding Cards	1058	936
Subroutines and FUNCTIONS	25	12
DO LOOPS	26	18

The use of IF (...) THEN and IF (...) THEN ... ELSE removed many GO TO and CONTINUE statements which cluttered the program. Consequently, some DO LOOPS were found redundant and could be combined. The use of Macros and Segments permitted the subroutines to be reduced by over half.

As an example of memory reduction made possible by table references and bit manipulation, consider one set of status tables used in another model that simulates the AUTODIN Store-and-Forward Message Network. This network has approximately 1400 Tributary Subscribers. The status table to describe a tributary and its transmission lines to and from the switching center requires 37 items. With unpacked tables, core requirements would be:

1400 x 37 = 51,800 words = 207.2K bytes

Even using 1-, 2-, and 4-byte arrays, the requirements would be:

1400 x 61 bytes = 85.4K bytes

Using bit packing, the 37 items fit into 10 words; thus, the requirements are:

1400 x 10 = 14,000 words = 56K bytes

The use of Macros can materially reduce the time required to produce a running program. Not only does the calling of a Macro require fewer lines of coding to be written, but it also can reduce debug time since error free coding is normally generated. The following is a partial list of checked-out Macros used in the Defense Communications System Performance Simulator models:

- Calendar of Events Management
- Queueing and unqueueing of messages and events
- Writing of Event Descriptors for post-simulation report programs
- Managing dynamic storage in discrete-event models
- Binary table search routines
- Gathering statistics from status tables
- Generating probability distributions
- Determining routing paths.

SUMMARY

Extensive additions to the capabilities of FORTRAN have been made in a simulation language called FORTSIM. This approach provides the following additional capabilities:

- Parametric Macro capability
- DO Segments
- IF (...) THEN ... ELSE, WHILE (...), UNTIL (...) instructions for top-down structured programming
- Bit packing, if needed, for memory reduction
- Table management with indirect addressing
- Free form coding and multiple instructions per card
- Symbolic statement labels
- Extensive error checking of coding
- Data manipulation instructions
- A larger percentage of error free coding by the use of Macros
- Program debugging is greatly simplified.

This approach requires a preprocessor which essentially adds an additional pass to the FORTRAN compiler. For the IBM 360/370 computers, cataloged procedures have been written which are similar to the standard compiler procedures, thus simplifying its use.

Finally, FORTSIM has been used extensively at the DCA in the development of large simulation models and has had enthusiastic reception by programmers. An extensive library of Macros have been developed for discrete simulation applications. Portability of programs is greatly enhanced without recourse to reducing the instruction set. Preprocessors for IBM 370/155, Honeywell 6000 series, and UNIVAC 1108 have been written and used. A version for online use with limited core is under development.