Application Program

GH20-0367-3

System/360 Continuous System Modeling Program User's Manual

Program Number 360A-CX-16X

This is an IBM System/360 program for the simulation of continuous systems. It provides an application-oriented input language that accepts problems expressed in the form of either an analog block diagram or a system of ordinary differential equations. Data input and output are facilitated by means of application-oriented control statements.

This manual contains a general description of the program, detailed programming information, and a description of the inputs and outputs.

Fourth Edition (October 1969)

This edition, H20-0367-3, is a reprint of H20-0367-2 incorporating TNL's N20-1938 and N20-2039. It does not obsolete H20-0367-2 as updated by those TNL's.

This edition applies to version 1, Modification Level 2, of System/360 Continuous System Modeling Program (360A-CX-16X) and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters.

Changes are continually made to the specifications herein. Therefore, before using this publication, consult the latest System/360 SRL Newsletter (N20-0360) for the editions that are applicable and current.

Copies of this and other IBM publications can be obtained through IBM branch offices.

A form has been provided at the back of this publication for readers' comments. If this form has been removed, address comments to: IBM Corporation, Technical Publications Department, 112 East Post Road, White Plains, N. Y. 10601.

CONTENTS

INTRODUCTION

The S/360 Continuous System Modeling Program (S/360 CSMP) is a problem-oriented program designed to facilitate the digital simulation of continuous processes on large-scale digital machines. The program provides an application-oriented language that allows these problems to be prepared directly and simply from either a block-diagram representation or a set of ordinary differential equations. The program includes a basic set of functional blocks with which the components of a continuous system may be represented, and accepts application-oriented statements for defining the connections between these functional blocks. S/360 CSMP also accepts FORTRAN statements, thereby allowing the user to readily handle nonlinear and time-variant problems of considerable complexity. Input and output are simplified by means of user-oriented control statements. A fixed format is provided for printing (tabular format) and print-plotting (graphic format) at selected increments of the independent variable. Through these features S/360 CSMP permits the user to concentrate upon the phenomenon being simulated, rather than the mechanism for implementing the simulation.

Typical applications might be a control engineer's study of the effectiveness of various control system designs, a physiologist's simulation of the cardiovascular system, and a mechanical engineer's investigation of the effects of damping and backlash in a proposed mechanical device. This program is based on the Digital Simulation Language (DSL/90).

The publication System/360 Continuous System Modeling Program, Application Description (H20-0240-1) provides an introduction to the material in this manual and is strongly recommended as preliminary reading.

GENERAL DESCRIPTION OF THE PROGRAM

S/360 CSMP is a "continuous system simulator" that combines the functional block modeling feature of "digital analog simulators", such as 1130 CSMP, with a powerful algebraic and logical modeling capability. Designed for use specifically by the engineer or scientist, it requires only a minimum knowledge of computer programming and operation. The input language enables a user to prepare structure statements describing a physical system, starting from either a block diagram or a differential equation representation of that system. Simplicity and flexibility are salient characteristics of this language. A knowledge of basic FORTRAN is helpful but not necessary.

The program provides a basic set of 34 functional blocks (also called functions), plus means for the user to define functions specially suited to his particular simulation requirements. Included in the basic set are such conventional analog computer components as integrators and relays plus many special purpose functions like delay time, zero-order hold, dead space, and limiter functions. This complement is augmented by the FORTRAN library functions, including, for example, cosine, and absolute value. Special functions can be defined either through FORTRAN programming or, more simply, through a macro capability that permits individual existing functions to be combined into a larger functional block. The user is thereby given a high degree of flexibility for different problem areas. For example, by properly preparing a set of special blocks, he can restructure S/360 CSMP into a problem-oriented language for chemical kinetics, control system analysis, or biochemistry. In effect, S/360 CSMP does not have to operate within the framework of a digital analog simulator language, but can take on the characteristics of a language oriented to any particular special purpose field in continuous system simulation.

Application-oriented input statements are used to describe the connections between the functional blocks. S/360 CSMP also accepts FORTRAN statements, thereby allowing the user to readily handle complex nonlinear and time-variant problems. A translator converts these structure statements into a FORTRAN subroutine, which is then compiled and executed alternately with a selected integration routine to accomplish the simulation.

Figure 1 shows the general form and application of the S/360 CSMP functions and structure statements. A specific example of a structure statement is

```
Y=INTGRL(IC, X)
```

which states that the output Y is obtained by integrating X, with the initial condition that Y at time zero is equal to IC. The name INTGRL defines the particular DEVICE, that is, the particular function to be performed on the variable X.

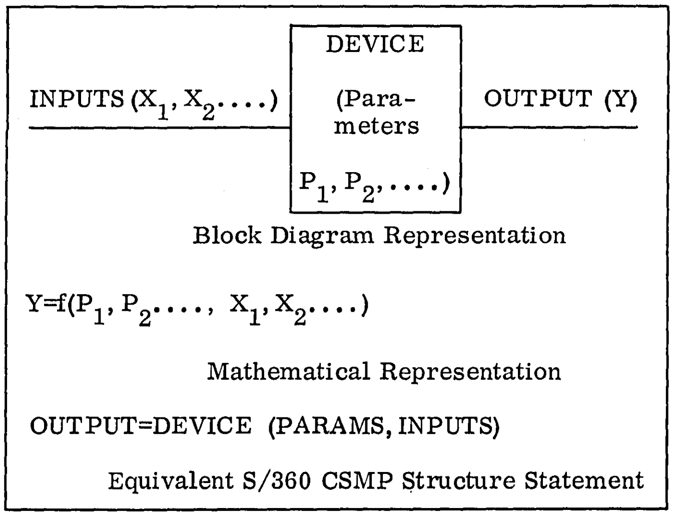

DEVICE
(Parameters $P_1, P_2, \ldots$)
INPUTS $(X_1, X_2 \ldots)$
OUTPUT (Y)
Block Diagram Representation
$Y=f(P_1, P_2 \ldots, X_1, X_2 \ldots)$
Mathematical Representation
OUTPUT=DEVICE (PARAMS, INPUTS)
Equivalent S/360 CSMP Structure Statement

Figure 1. Illustration of S/360 CSMP functional blocks

Input and output are simplified by means of a free format for data entry and user-oriented input and output control statements. With few exceptions, data and control statements may be entered in any order and may be intermixed with structure statements. Output options include printing of variables in standard tabular format, print-plotting in graphic form, and preparation of a data set for user-prepared plotting programs.

Data statements are used to assign numeric values to variables that are to remain fixed during a run -- for example, IC in the structure statement above. A representative data statement is

```
INCON IC=10.0
```

where INCON is the label identifying the card as an initial condition card, IC is the variable to be assigned a numeric value, and 10.0 is the value assigned.

Control statements are used to control certain operations associated with the running of the program, such as run time, printout, and stacking of jobs. An example is the output control statement

```
PRINT Y
```

where PRINT is a card label specifying that a list of the variable Y is to be printed.

Two important features of S/360 CSMP are statement sequencing and a choice of integration methods. With few exceptions, structure statements may be written in any order and, at the user's option, may be sorted automatically (or not sorted) by the system to establish the correct information flow. Centralized integration is used to ensure that all integrator outputs are computed simultaneously at the end of the iteration cycle. A choice may be made between the fifth-order Milne predictor-corrector, fourth-order Runge-Kutta, Simpson's, second-order Adams, trapezoidal, and rectangular integration methods. The first two methods allow the integration interval to be adjusted by the system to meet a specified error criterion.

An entire S/360 CSMP simulation can be conveniently controlled by a sequence of FORTRAN statements performed only at the termination of a simulation run. A user-written FORTRAN program can test run responses, define run control conditions, and supervise both input and output information. This provides a simple and efficient method for handling the type of iterative computation involved, for example, in automatic search procedures for parameter optimization.

Another feature is the ability to initialize variables or parameters -- that is, to indicate a group of structure statements to be executed only once at the start of the simulation. This feature provides efficiency in executing the problem, since the computations will be made only once and not for each iteration.

FORTRAN IV (Level E) is used as the source language for approximately 95% of this application package; those operations not readily performed in FORTRAN IV (Level E) are coded as subroutines in System/360 Assembler Language. All routines operate under Operating System/360. All calculations are done in single-precision, floating-point arithmetic.

The program requires a minimum of 102K bytes of storage (excluding that required by OS/360), the Standard Instruction Set, and the Floating-Point Option. In addition to the I/O units needed by the Operating System/360 for FORTRAN IV (Level E) compiling, the program will require three logical utility units. One of these must be a direct access storage device (DASD), while the other two may be portions of the required DASD, or may be portions of other DASD's or magnetic tape drives.

The problem deck must be preceded and followed by appropriate OS/360 control cards. Each installation will have its own procedures for making these available to the user. A sample set is shown in the Operator's Manual.

To the user and operator of the system, the entire run will appear as a single job, even though it is a multiple-step program. The general systems chart is shown in Figure 2.

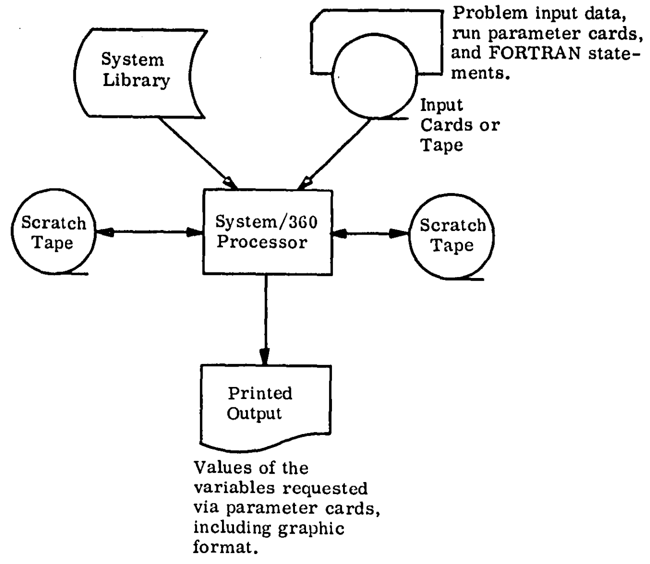

System Library
Problem input data, run parameter cards, and FORTRAN statements.
Input Cards or Tape
Scratch Tape
System/360 Processor
Scratch Tape
Printed Output
Values of the variables requested via parameter cards, including graphic format.

Figure 2. General systems chart

A simple illustration of the program is the set of structure statements that can be prepared from a block diagram representation of the system shown in Figure 3. This system consists of a spring, a mass, and a damper suspended from a fixed reference position. If the mass is displaced from its rest position and then released, it will oscillate until the energy is dissipated by the damper. The purpose of a simulation might be to analyze the effect of different spring constants on the motion of the mass. One possible simulation diagram for representing this simple mechanical system is shown in Figure 4. The corresponding structure statements are shown in Figure 5. Note that operators are used to indicate the basic arithmetic relationships and that the INTGRL function block is used to obtain the variables X and XDOT. The order of statements is of no consequence, since S/360 CSMP will automatically sort them to achieve a proper representation of the parallel physical system. A similar set of statements could, of course, have been developed directly from the system differential equation.

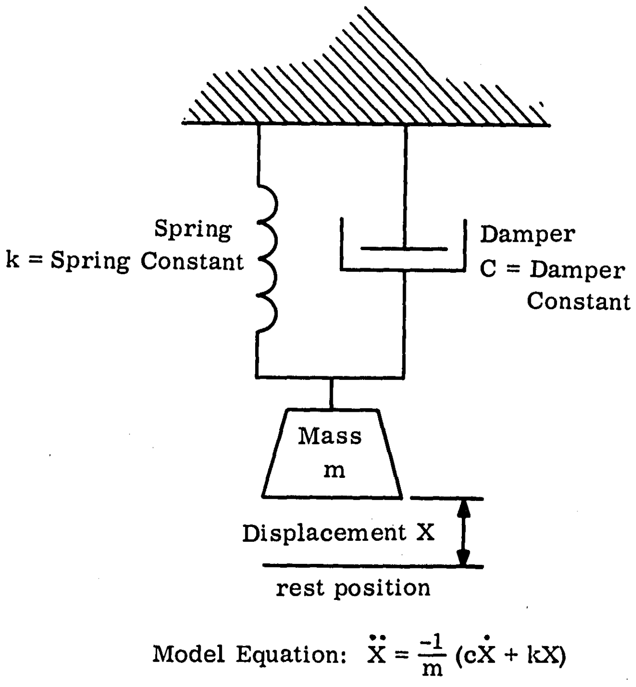


Figure 3. Model of a spring, mass damper system

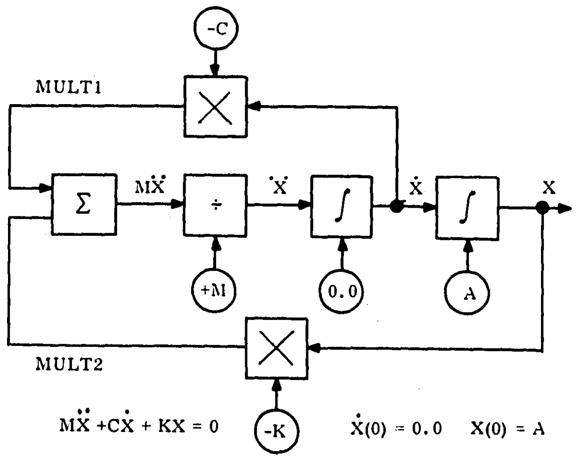


Figure 4. Block diagram for solution of spring, mass, damper problem

```
MX2DOT=MULT1+MULT2
X2DOT=MX2DOT/M
XDOT=INTGRL(0.0,X2DOT)
MULT1=-C*XDOT
X=INTGRL(A,XDOT)
MULT2=-K*X
```

Figure 5. Structure statement development from the block diagram

ELEMENTS OF THE SYSTEM/360 CSMP LANGUAGE

A system to be simulated is described to the program by a series of structure, data, and control statements. Structure statements describe the functional relationships between the variables of the model, and, taken together, define the network to be simulated. Data statements assign numeric values to the parameters, constants, initial conditions, and table entries associated with the problem. Control statements specify options relating to the translation, execution, and output phases of the S/360 CSMP program, such as run time, integration interval, and type of output. The basic elements in the preparation of these three types of statements are numeric constants, symbolic names, operators, functions, and labels.

NUMERIC CONSTANTS

Constants are unchanging quantities used in their numeric form in the input statements. Note that the word "constant" is used in two different ways in this manual. When describing a language element, it refers to a series of characters (digits) written in numeric form. The word is also used as a label in a Data statement -- that is, to describe a type of variable. Context will adequately differentiate between these meanings.

There are two types of constants: integer and real. An integer constant is a whole number written with from one to ten decimal digits without a decimal point, and cannot contain embedded commas or blanks. An integer constant may be positive, zero, or negative; if unsigned, it is assumed to be positive. Its magnitude must not be greater than 2147483647 (2^{31} -1). The following are valid integer constants:

```
0
+91
-3468
24691
```

A real (floating-point) constant is a number written with from one to seven significant decimal digits with a decimal point. It may be positive or negative; if unsigned, it is assumed to be positive. A real constant optionally may be followed by a decimal exponent written as the letter E, followed by a signed or unsigned one- or two-digit integer constant. The decimal exponent E format forms a real constant that is the product of the real constant portion times ten raised to the desired power. It may not contain embedded commas or blanks. The magnitude of a real number, if followed by an E decimal exponent, can be 0 or any value from 16^{-63} through 16^{63} (that is, approximately 10^{75}); otherwise, it may consist of one through seven decimal digits. The following are valid real constants:

```
0.
-97.345
5934.75
+24.951E9
-83.625E-05
```

Real constants are restricted to twelve characters total.

SYMBOLIC NAMES

Symbolic names represent quantities that may either change during a run or be changed, by the program, between successive runs of the same model structure.

A symbolic variable name contains from one to six alphameric characters -- that is, numeric 0 through 9, or alphabetic A through Z. The first character must be alphabetic. A symbolic name must not contain embedded blanks or any of the special characters

```
+   /   (   $
-   =   )   ,
*   .   '
```

and may not be a word reserved either by S/360 CSMP or FORTRAN IV. These reserved words are listed under "Methods" in this manual. The following are valid variable names:

```
RATE
SQ915
A41B
```

All variables and functions that are represented by symbolic names are normally treated within the simulation as being real -- that is, as having floating-point values. Integer variables and functions must be specified with a FIXED translation control statement. Note that this convention differs from FORTRAN, which automatically treats as an integer any variable that has as the first character of its name the letter I, J, K, L, M, or N.

Subject to certain restrictions, symbolic names may be subscripted to reference arrays and matrices. Note, however, that the symbolic name for an array comprised of real variables should not start with the letter I, J, K, L, M, or N. Except for this, the normal rules of FORTRAN apply: the subscript must be an integer, or an integer variable, enclosed in parentheses following the symbolic name. No special restrictions are imposed by S/360 CSMP when subscripted quantities are used within procedural structure -- that is, within NOSORT sections or PROCEDURE functions. Within parallel, or sorted sections, however, subscripted quantities may appear only on the right-hand side of equations; the sorting algorithm processes them as data rather than as output variables. Single-dimensional arrays are best declared by use of the STORAGE label. The associated TABLE feature permits simple data entry for symbols declared in this way. Matrices must be declared by use of a FORTRAN DIMENSION statement with a virgule (/) in cc 1.

OPERATORS

Operators are used, instead of functional blocks, to indicate the basic arithmetical functions or relationships. As in FORTRAN, these operators are:

Symbol	Function	Symbol	Function
+	addition	**	exponentiation
-	subtraction	=	replacement
*	multiplication	( )	grouping of variables and/ or constants
/	division		

Note that, as in FORTRAN, the equal sign means "is to be replaced by", rather than "is equal to". This means that the usual arithmetic usage of the equal sign is valid, but so too is a FORTRAN statement of the type A=A+1.0, when used in an unsorted sequence of statements. If A initially had a value of 2.0, this statement would give it a value of 3.0.

As in algebra and FORTRAN, parentheses may be used in arithmetic expressions to specify the order in which the arithmetic operations are to be performed. Expressions within parentheses are always evaluated first. When parentheses are omitted, or when an entire arithmetic or functional expression is enclosed within a single pair of parentheses, the order in which the operations are performed is as follows:

Operation	Hierarchy
Evaluation of functions	1st (highest)
Exponentiation (**)	2nd
Multiplication and division (* and/)	3rd
Addition and subtraction (+ and -)	4th (lowest)

For operators of the same hierarchy, except exponentiation, the component operations of the expression are performed from left to right. Thus, the arithmetic expression A/B*C is evaluated as (A/B)*C. For exponentiation, the evaluation is from right to left. Thus, the expression A**B**C is evaluated as $A^{(B^C)}$.

Some illustrations of the use of operators to build structure statements are:

```
RATE=DIST/TIME

Y=A*X**2+B

A=(B*C)+(D*E)
```

FUNCTIONS

Finally, there are the functional blocks (functions), which perform the more complex mathematical operations such as integration, time delay, quantization, and limiting. Figure 1 has already illustrated the basic form and nature of these functions.

The basic S/360 CSMP library includes the standard functions found in analog computers plus a number of additional special-purpose functions often encountered in simulation problems. Table 1 illustrates this basic library. The general, canonical form of each function is shown with inputs indicated by the letter X, outputs by Y, initial conditions by IC, and parameters by P. For the basic S/360 CSMP functions, all of these are real variables or constants. If computed rather than specified by data statements, initial conditions and parameters should be computed in the INITIAL segment of the model. The DELAY function deserves special mention since its first argument, N, must be a literal integer constant; this tells the translator how many storage locations should be allocated for its "memory" and cannot be varied as a parameter. (Details of the internal coding of these functions are provided in the System Manual, Y20-0111.)

The functions available in the standard FORTRAN IV library in the user's system can also be treated as functional blocks, supplementing the basic S/360 CSMP library. Illustrations of the most useful FORTRAN functions are shown in Table 2. Note that, contrary to normal CSMP usage, several of these FORTRAN functions use or produce integer variables.

Note also that the user may add any desired function to the S/360 CSMP library -- temporarily for his personal use, or permanently so as to be conveniently shared by other users at the installation. Functions supplied by the user must be named according to the specifications described under "Symbolic Names"; additionally, if the output of the function is an integer variable, the name of the function must be placed on a FIXED translation control statement.

LABELS

The first word of S/360 CSMP data and control statements is a label that identifies to the program the purpose of the statement. Some statements contain only the label -- for example, INITIAL, NOSORT, and ENDMAC. Others contain a label and appropriate data. For example, to specify the integration interval and the "finish time" for a run, one would use the TIMER statement as follows:

```
TIMER      DELT = 0.025,      FINTIM = 450.
```

The format of each statement type is described in detail later in this manual. With the exception of COMMON and ENDJOB, the program examines only the first four characters of a label; hence, INITIAL and INIT, or PARAMETER and PARAM, or PROCEDURE and PROCED are equally acceptable. Note that for all those statements in which additional data follows the label, the label must be separated from that data by at least one blank column.

MATHEMATICAL FUNCTIONS

GENERAL FORM	FUNCTION
$Y = INTGRL\ (IC,\ X)$ $Y\ (0) = IC$ INTEGRATOR	$Y = \int_0^t X\,dt + IC$ EQUIVALENT LAPLACE TRANSFORM: $\frac{1}{S}$
$Y = DERIV\ (IC,\ X)$ $\dot{X}\ (t = 0) = IC$ DERIVATIVE	$Y = \frac{dX}{dt}$ EQUIVALENT LAPLACE TRANSFORM: S

$Y = DELAY\ (N,\ P,\ X)$
P = DELAY TIME
N = NUMBER OF POINTS SAMPLED IN INTERVAL P (INTEGER CONSTANT)
 	
DEAD TIME (DELAY)
	$Y\ (t) = X\ (t - P) \quad t \geq P$
	$Y = 0 \quad t < P$
	
	EQUIVALENT LAPLACE TRANSFORM: e^{-PS}

$Y = ZHOLD\ (X_1,\ X_2)$ ZERO-ORDER HOLD	$Y = X_2 \quad X_1 > 0$ $Y =$ LAST OUTPUT $\quad X_1 \leq 0$ $Y\ (0) = 0$ EQUIVALENT LAPLACE TRANSFORM: $\frac{1}{S}\ (1 - e^{St})$

Y = IMPL (IC, P, FOFY)
IC = FIRST GUESS
P = ERROR BOUND
FOFY = OUTPUT NAME OF LAST STATEMENT IN ALGEBRAIC LOOP DEFINITION
 	
IMPLICIT FUNCTION
	Y = FUNCT (Y)
	$\lvert Y - FUNCT\ (Y) \rvert \leq P \lvert Y \rvert$

Table 1. Library of S/360 CSMP functional blocks

SYSTEM MACROS

GENERAL FORM	FUNCTION
$Y = \text{MODINT}(IC, X_1, X_2, X_3)$ MODE-CONTROLLED INTEGRATOR	$Y = \int_0^t X_3\,dt + IC \qquad X_1 > 0, \text{ any } X_2$ $Y = IC \qquad X_1 \leq 0,\ X_2 > 0$ $Y = \text{LAST OUTPUT} \qquad X_1 \leq 0,\ X_2 \leq 0$
$Y = \text{REALPL}(IC, P, X)$ $Y(0) = IC$ 1ST ORDER LAG (REAL POLE)	$P\dot{Y} + Y = X$ EQUIVALENT LAPLACE TRANSFORM: $\frac{1}{PS + 1}$
$Y = \text{LEDLAG}(P_1, P_2, X)$ LEAD-LAG	$P_2\dot{Y} + Y = P_1\dot{X} + X$ EQUIVALENT LAPLACE TRANSFORM: $\frac{P_1 S + 1}{P_2 S + 1}$
$Y = \text{CMPXPL}(IC_1, IC_2, P_1, P_2, X)$ $Y(0) = IC_1$ $\dot{Y}(0) = IC_2$ 2ND ORDER LAG (COMPLEX POLE)	$\ddot{Y} + 2P_1P_2\dot{Y} + P_2^2 Y = X$ EQUIVALENT LAPLACE TRANSFORM: $\frac{1}{S^2 + 2P_1P_2S + P_2^2}$

Table 1. (Continued)

SWITCHING FUNCTIONS

GENERAL FORM	FUNCTION	
$Y = FCNSW(X_1, X_2, X_3, X_4)$ FUNCTION SWITCH	$Y = X_2$	$X_1 < 0$
	$Y = X_3$	$X_1 = 0$
	$Y = X_4$	$X_1 > 0$
$Y = INSW(X_1, X_2, X_3)$ INPUT SWITCH (RELAY)	$Y = X_2$	$X_1 < 0$
	$Y = X_3$	$X_1 \geq 0$
$Y_1, Y_2 = OUTSW(X_1, X_2)$ OUTPUT SWITCH	$Y_1 = X_2, Y_2 = 0$	$X_1 < 0$
	$Y_1 = 0, Y_2 = X_2$	$X_1 \geq 0$
$Y = COMPAR(X_1, X_2)$ COMPARATOR	$Y = 0$	$X_1 < X_2$
	$Y = 1$	$X_1 \geq X_2$
$Y = RST(X_1, X_2, X_3)$ RESETTABLE FLIP-FLOP	$Y = 0$	$X_1 > 0$
	$Y = 1$	$X_2 > 0, X_1 \leq 0$
	$Y = 0$	$X_1 \leq 0, X_2 \leq 0,$ $X_3 > 0, Y_{n-1} = 1$
	$Y = 1$	$X_1 \leq 0, X_2 \leq 0,$ $X_3 > 0, Y_{n-1} = 0$
	$Y = 0$	$X_1 \leq 0, X_2 \leq 0,$ $X_3 \leq 0, Y_{n-1} = 0$
	$Y = 1$	$X_1 \leq 0, X_2 \leq 0,$ $X_3 \leq 0, Y_{n-1} = 1$

Table 1. (Continued)

FUNCTION GENERATORS

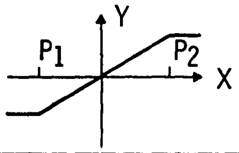
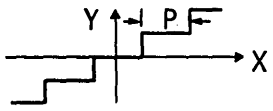
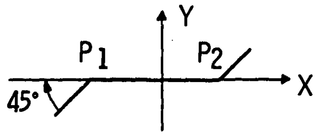
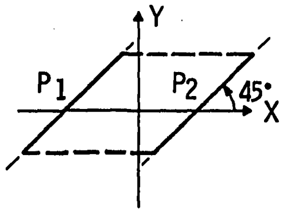
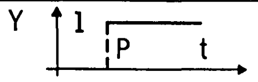
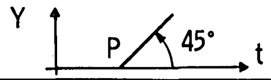
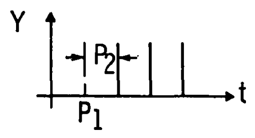
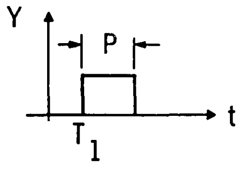
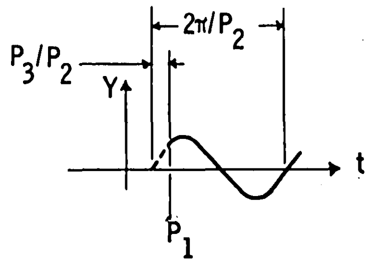
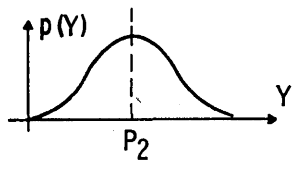
GENERAL FORM	FUNCTION
$Y = \text{AFGEN (FUNCT, X)}$ ARBITRARY FUNCTION GENERATOR (LINEAR INTERPOLATION)	$Y = \text{FUNCT (X)}$
$Y = \text{NLFGEN (FUNCT, X)}$ ARBITRARY FUNCTION GENERATOR (QUADRATIC INTERPOLATION)	$Y = \text{FUNCT (X)} \quad X_0 \leq X \leq X_n$
$Y = \text{LIMIT} (P_1, P_2, X)$ LIMITER	$Y = P_1 \quad X<P_1$ $Y = P_2 \quad X>P_2$ $Y = X \quad P_1 \leq X \leq P_2$ 
$Y = \text{QNTZR (P, X)}$ QUANTIZER	$Y = kP \quad (k - 1/2)P < X \leq (k + 1/2)P$ $k = 0, \pm 1, \pm 2, \pm 3 \ldots$ 
$Y = \text{DEADSP} (P_1, P_2, X)$ DEAD SPACE	$Y = 0 \quad P_1 \leq X \leq P_2$ $Y = X - P_2 \quad X>P_2$ $Y = X - P_1 \quad X<P_1$ 
$Y = \text{HSTRSS (IC}, P_1, P_2, X)$ $Y(0) = IC$ HYSTERESIS LOOP	$Y = X - P_2 \quad (X - X_{n-1})>0$ AND $Y_{n-1} \leq (X - P_2)$ $Y = X - P_1 \quad (X - X_{n-1})<0$ AND $Y_{n-1} \geq (X - P_1)$ OTHERWISE $\quad$ Y = LAST OUTPUT 

Table 1. (Continued)

SIGNAL SOURCES

GENERAL FORM	FUNCTION
Y = STEP (P) STEP FUNCTION	$Y = 0$ $t < P$ $Y = 1$ $t \geq P$ 
Y = RAMP (P) RAMP FUNCTION	$Y = 0$ $t < P$ $Y = t - P$ $t \geq P$ 
Y = IMPULS (P_1, P_2) IMPULSE GENERATOR	$Y = 0$ $t < P_1$ $Y = 1$ $(t - P_1) = kP_2$ $Y = 0$ $(t - P_1) \neq kP_2$ $k = 0, 1, 2, 3....$ 

Y = PULSE (P, X)
P = MINIMUM PULSE WIDTH
PULSE GENERATOR (WITH X>0 AS TRIGGER)
	$Y = 1$ $T_k \leq t < (T_k + P)$ or $X > 0$
	$Y = 0$ OTHERWISE
	T_k = TIME OF TRIGGER
	

Y = SINE (P_1, P_2, P_3) P_1 = DELAY P_2 = FREQUENCY (RADIANS PER UNIT TIME) P_3 = PHASE SHIFT IN RADIANS TRIGONOMETRIC SINE WAVE WITH DELAY, FREQUENCY AND PHASE PARAMETERS	$Y = 0$ $t < P_1$ $Y = \mathrm{SIN}(P_2 (t - P_1) + P_3)$ $t \geq P_1$ 
Y = GAUSS (P_1, P_2, P_3) P_1 = ANY ODD INTEGER P_2 = MEAN P_3 = STANDARD DEVIATION NOISE (RANDOM NUMBER) GENERATOR WITH NORMAL DISTRIBUTION	NORMAL DISTRIBUTION OF VARIABLE Y p(Y) = PROBABILITY DENSITY FUNCTION 

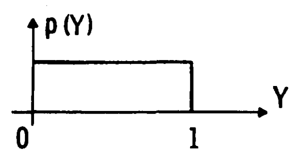
Y = RNDGEN (P)
P = ANY ODD INTEGER
NOISE (RANDOM NUMBER) GENERATOR WITH UNIFORM DISTRIBUTION
	UNIFORM DISTRIBUTION OF VARIABLE Y
	p(Y) = PROBABILITY DENSITY FUNCTION
	

Table 1. (Continued)

LOGIC FUNCTIONS

GENERAL FORM	FUNCTION	
$Y = AND (X_1, X_2)$	$Y = 1$	$X_1>0, X_2>0$
AND	$Y = 0$	OTHERWISE
$Y = NAND (X_1, X_2)$	$Y = 0$	$X_1>0, X_2>0$
NOT AND	$Y = 1$	OTHERWISE
$Y = IOR (X_1, X_2)$	$Y = 0$	$X_1\leq 0, X_2\leq 0$
INCLUSIVE OR	$Y = 1$	OTHERWISE
$Y = NOR (X_1, X_2)$	$Y = 1$	$X_1\leq 0, X_2\leq 0$
NOT OR	$Y = 0$	OTHERWISE
$Y = EOR (X_1, X_2)$	$Y = 1$	$X_1\leq 0, X_2>0$
	$Y = 1$	$X_1>0, X_2\leq 0$
EXCLUSIVE OR	$Y = 0$	OTHERWISE
$Y = NOT (X)$	$Y = 1$	$X\leq 0$
NOT	$Y = 0$	$X>0$
$Y = EQUIV (X_1, X_2)$	$Y = 1$	$X_1\leq 0, X_2\leq 0$
	$Y = 1$	$X_1>0, X_2>0$
EQUIVALENT	$Y = 0$	OTHERWISE

Table 1. (Continued)

GENERAL FORM	FUNCTION
Y = EXP (X) EXPONENTIAL	$Y = e^X$
Y = ALOG (X) NATURAL LOGORITHM	Y = LN (X)
Y = ALOG10 (X) COMMON LOGORITHM	$Y = LOG_{10}(X)$
Y = ATAN (X) ARCTANGENT	Y = ARCTAN (X)
Y = SIN (X) TRIGONOMETRIC SINE	Y = SIN (X)
Y = COS (X) TRIGONOMETRIC COSINE	Y = COS (X)
Y = SQRT (X) SQUARE ROOT	$Y = X^{1/2}$
Y = TANH (X) HYPERBOLIC TANGENT	Y = TANH (X)
Y = ABS (X) ABSOLUTE VALUE (REAL ARGUMENT AND OUTPUT)	$Y = \|X\|$
Y = IABS (X) ABSOLUTE VALUE (INTEGER ARGUMENT AND OUTPUT)	$Y = \|X\|$

Table 2. FORTRAN functions

GENERAL FORM	FUNCTION
Y = AMAX0 $(X_1, X_2 \ldots X_n)$ LARGEST VALUE (INTEGER ARGUMENTS AND REAL OUTPUT)	Y = MAX $(X_1, X_2 \ldots X_n)$
Y = AMAX1 $(X_1, X_2 \ldots X_n)$ LARGEST VALUE (REAL ARGUMENTS AND OUTPUT)	Y = MAX $(X_1, X_2 \ldots X_n)$
Y = MAX0 $(X_1, X_2 \ldots X_n)$ LARGEST VALUE (INTEGER ARGUMENTS AND OUTPUT)	Y = MAX $(X_1, X_2 \ldots X_n)$
Y = MAX1 $(X_1, X_2 \ldots X_n)$ LARGEST VALUE (REAL ARGUMENTS AND INTEGER OUTPUT)	Y = MAX $(X_1, X_2 \ldots X_n)$
Y = AMIN0 $(X_1, X_2 \ldots X_n)$ SMALLEST VALUE (INTEGER ARGUMENTS AND REAL OUTPUT)	Y = MIN $(X_1, X_2 \ldots X_n)$
Y = AMIN1 $(X_1, X_2 \ldots X_n)$ SMALLEST VALUE (REAL ARGUMENTS AND OUTPUT)	Y = MIN $(X_1, X_2 \ldots X_n)$
Y = MIN0 $(X_1, X_2 \ldots X_n)$ SMALLEST VALUE (INTEGER ARGUMENTS AND OUTPUT)	Y = MIN $(X_1, X_2 \ldots X_n)$
Y = MIN1 $(X_1, X_2 \ldots X_n)$ SMALLEST VALUE (REAL ARGUMENTS AND INTEGER OUTPUT)	Y = MIN $(X_1, X_2 \ldots X_n)$

Table 2. (Continued)

STRUCTURE OF THE MODEL

The nucleus of a "continuous system simulator" is, of course, a computer mechanism for solving the differential equations that represent the dynamics of the model. Usually, however, there are also computations that must be performed before each run, and sometimes, computations after each run. For example, certain parameters of a model might be considered basic; secondary parameters and initial conditions are often expressed as functions of these basic parameters. Evaluation of these functions is desired just once per run. Frequently, too, one needs to perform some terminal evaluation of each run; in a design study, for example, one might compute some "figure-of-merit" for each run of a parameter search.

To satisfy these requirements, the general S/360 CSMP formulation of a model is divided into three segments -- Initial, Dynamic, and Terminal -- that describe the computations to be performed before, during, and after each simulation run. These represent the highest level of the structural hierarchy. Each of the segments may comprise one or more sections; these, in turn, contain the structure statements that specify model dynamics and associated computations. The sections represent rational groupings of structure statements and may be processed as either parallel or procedural entities as appropriate. The overall structural hierarchy is illustrated in Figure 6.

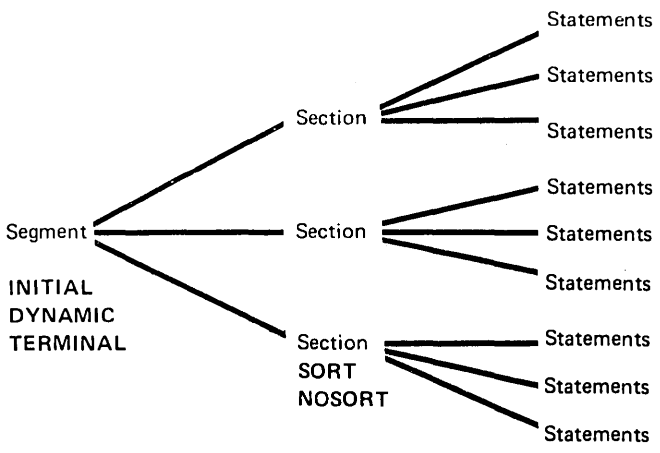


Figure 6. Structural hierarchy of the System/360 CSMP language

INITIAL, DYNAMIC, AND TERMINAL SEGMENTS

The Initial segment is intended exclusively for computation of initial condition values and those parameters that the user prefers to express in terms of more basic parameters. Thus, if a model repeatedly makes use of the cross-sectional area of a cylindrical member, the Initial segment might contain the structure statement:

```
AREA = 3.1416*(R**2)
```

with radius R, which might be considered the more basic parameter, specified on a data card as follows:

```
PARAMETER R=7.5
```

Many simple simulations do not require this feature. The Initial segment is, therefore, optional.

The Dynamic segment is normally the most extensive in the model. It includes the complete description of the system dynamics, together with any other computations desired during the run. Functionally, the Dynamic segment is analogous to the block diagram representation or to the ordinary differential equation representation of system dynamics. The structure statements within this segment are generally a mixture of S/360 CSMP and FORTRAN statements; the following might be considered representative:

```
DRAG = 0.5*RHO*S*CD*(V**2)

VX=INTGRL(VZERO, (THRUST-DRAG)/MASS)
```

For most models, the Dynamic segment consists of a single section. For more complicated systems, however, it is often desirable, and sometimes required, that it be divided into several sections. In modeling an industrial process, for example, it might be desirable to separate the various unit process models into separate sections simply because the physical units are distinct.

The Terminal segment is used for those computations desired after completion of each run. This will often be a simple calculation based on the final value of one or more model variables, but more powerful use of this segment is readily possible. For example, one might incorporate an optimization algorithm that will modify the values of critical system parameters. If this section includes the statement

```
CALL RERUN
```

S/360 CSMP will automatically be recycled through the simulation using the newly set parameters. The section "Sample Problem" in this manual illustrates use of this rerun feature for solution of a simple two-point boundary value problem. Many simulations require no terminal computations. The Terminal segment is therefore optional; it is omitted merely by deletion of the TERMINAL label.

Segmentation, the explicit division of the model into computations to be performed before, during, and after each run, is provided by the control statements: INITIAL, DYNAMIC, TERMINAL, and END. In the general case, as illustrated in Figure 7, INITIAL is the first statement of the specification of model structure. (It is not normally the first statement of the composite S/360 CSMP data deck, since certain other translation control statements and all MACRO definitions must be placed before any structure statements. This is discussed more fully under "Translation Control Statements".) The DYNAMIC statement separates the Initial segment from the Dynamic segment. In similar manner, the TERMINAL statement separates the Dynamic segment from the first statement of the Terminal segment. The first occurrence of the END (or CONTINUE) statement completes the specification of model structure.

```
        . . . . . . . . . . .
        . . . . . . . . . . .
INITIAL
        . . . . . . . . . . .  }
        . . . . . . . . . . .  }    Initial Segment
DYNAMIC
        . . . . . . . . . . .  }
        . . . . . . . . . . .  }    Dynamic Segment
        . . . . . . . . . . .  }
TERMINAL
        . . . . . . . . . . .  }
        . . . . . . . . . . .  }    Terminal Segment
END
```

Figure 7. Structural segmentation

The Dynamic segment is required; the Initial and Terminal segments are optional. If an Initial segment is used, it must precede the Dynamic segment. The Terminal segment, if used, must follow the Dynamic segment. An Initial segment should always be preceded by an INITIAL statement; the DYNAMIC statement must be used to separate the Initial and Dynamic segments. The Terminal segment, when used, must be separated from the Dynamic segment by a TERMINAL statement. The model may consist of the Dynamic segment alone, and in this case none of these labels is required.

Most simulation studies require only a simple superstructure such as illustrated in Figure 7, which shows use of all three segments, but not further divided into sections. Sectioning, the grouping of structure statements within a segment, is usually required only for rather sophisticated simulations. Since the model illustrated in Figure 7 does not include SORT or NOSORT labels, the standard options concerning parallel and procedural structure are automatically obtained. S/360 CSMP, in this case, assumes that the Initial and Dynamic segments represent parallel structure while the Terminal segment represents procedural structure. Thus, all the structure statements within the Initial segment will be automatically sorted by the program. Those within the Dynamic segment will also be sorted, but independently, of course, from those in the Initial segment. In contrast, all structure statements within the Terminal segment are assumed to be procedural and already in the desired computational order; the program thereby does not rearrange the order of these statements.

These standard options may be easily overridden by use of the labels SORT and NOSORT. For example, one might want the entire Dynamic segment to represent procedural structure, thus permitting unrestricted use of FORTRAN conditional logic and branching. This option is achieved merely by using the label NOSORT immediately after the label DYNAMIC. The same reasoning applies to the Initial segment. The entire Terminal segment could similarly be made parallel structure by inserting the label SORT immediately after the TERMINAL label.

SECTIONS -- PARALLEL OR PROCEDURAL

A section is a group of structural statements. If a segment contains only a single section, then sectioning sometimes is not explicitly indicated. When desired, sectioning is obtained through use of the translation control statements SORT and NOSORT. The former declares that the structure within the section is to be considered parallel; that is, the translator should arrange the statements in proper computational sequence to achieve the same effect as would be obtained by a parallel computer. Thus, the user need not be concerned with the computational sequence when modeling ordinary dynamic effects, since, in nature, most phenomena do exist and operate "in parallel". The sections are terminated by the occurrence of the next section or segment, that is, by the next SORT or NOSORT within the segment or by the DYNAMIC or TERMINAL translation statement.

The NOSORT statement declares that all the subsequent structure statements within the section are to be considered procedural. Thus, in generating the derivative subroutine UPDATE, the translator inserts the structure statements of a NOSORT section without changing their sequence. Full use of FORTRAN, including conditional branching and input/output, is thereby permissible within a NOSORT section. The sorting algorithm ignores the statements in such a section and responsibility for the computational sequence rests with the user.

It should be noted that the translator does not change the order of the sections. For example, if the DYNAMIC segment of a model contains three sections -- SORT, NOSORT, SORT, in that order (see Figure 8) -- the corresponding portion of the derivative subroutine UPDATE will have three sections. In the first and third, the computational sequence will have been rearranged to satisfy the sorting algorithm; the sequence of those statements corresponding to the NOSORT section will be unchanged. The rationale for sectioning a model rests with the user. This feature of S/360 CSMP facilitates sophisticated modeling, but the inexperienced user should defer its use until needed. (See "Modeling Techniques" for a more thorough discussion of this topic.)

```
  .......
INITIAL
  NOSORT
  .......  }
           }  Unsorted, procedural section
  .......  }
DYNAMIC
  .......  }
           }  Sorted, parallel section
  .......  }
  NOSORT
  .......  }
           }  Unsorted, procedural section
  .......  }
  SORT
  .......  }
           }  Sorted, parallel section
  .......  }
TERMINAL
  .......  }
           }  Unsorted, procedural section
  .......  }
END
```

Figure 8. Illustration of complex sectioning

STRUCTURE STATEMENTS

Structure statements define the network or model to be simulated by describing the functional relationships between the variables of the model. They also can be viewed as connectors between the S/360 CSMP functional blocks, which are identified by inputs, outputs, and function type. Various uses of the same function are made unique by assigning different output variable names for each use.

The general form of a structure statement using a S/360 CSMP function has been shown in Figure 1; the S/360 CSMP library of functions has been listed in Tables 1 and 2. Specification of a typical function involves the use of output-variable names, initial conditions, parameters, and inputs. The latter may be arithmetic expressions, previously computed functional block outputs, variables, constants, or references to other functions. Initial conditions and parameters may be variable names or constants. Structure statements are in FORTRAN equation form with the output variables on the left of the equal sign and an expression on the right. The expression may be a single constant or variable; an output from a function or subroutine; or a combination of constants, variables, and functions connected by operators. Examples of structure statements are:

```
   Y=A*X+B

ROOT=SQRT(X**2+Y**2)

   Z=PULSE(PAR,XDOT)

XDOT=INTGRL(2.0,X**2+R/D)
```

In the above examples SQRT, PULSE, and INTGRL are functional block names. The outputs are Y, ROOT, Z, and XDOT.

Note that user-defined MACROs and PROCEDUREs also involve structure statements. MACROs are defined before the first actual model statements -- that is, before the INITIAL statement. The MACRO definition is not considered part of the model but, rather, a preliminary extension of the MACRO library. A MACRO is later used or invoked within the model by a simple structure statement that must agree exactly with the canonical form prescribed in its MACRO definition. A PROCEDURE is conceptually a single functional element, but its definition usually requires several structure statements. (Both of these elements are described more fully under "Modeling Techniques".)

An alternative "specification" form of the INTGRL statement is available for special applications involving arrays of integrators or integrators within subprograms. Its use should be deferred until one is familar with the more commonly used features of the language. The basic form of this usage is as follows:

```
Z1 = INTGRL (ZIC1, DZDT1, nn )
```

where nn represents a literal integer constant corresponding to the number of elements in the integrator array. Note that this "specification" form requires three arguments, whereas the normal use of the INTGRL function requires only two. This special form does not of itself provide integration; instead, it specifies to the S/360 CSMP translator that the total number of integrators indicated in the DYNAMIC segment of the model should be augmented by the quantity nn. The symbols Z1, ZIC1, and DZDT1 are usually dummies, equivalenced to the first elements of corresponding vector arrays Z, ZIC, and DZDT.

Structure statements are translated and placed into a FORTRAN subroutine called UPDATE, which is executed at each iteration cycle. In general, structure statements may be written in any order and intermixed freely with data and control statements. The system establishes the correct sequence of computation, based on the inputs and outputs of each statement. A statement is considered to be in sequence when all of its inputs have been processed previously in the current iteration cycle.

Structure statements are subject to the following general rules:

1. The operators +, -, *, /, and ** may not appear consecutively; for example, A + -B is in error, but A + (-B) will work correctly.
2. Function arguments in an expression must be separated by commas and enclosed in parentheses; for example, Z=PULSE(PAR, XDOT) should not be written as Z=PULSE, PAR, XDOT or as Z=PULSE (PAR) (XDOT).
3. Any expression may be enclosed in parentheses, and expressions may be connected by arithmetic operators -- for example, Y=(A+B)-(C+D).
4. All variable names and constants, including statement numbers, must be separated from each other by blanks, operators, or special characters, as appropriate. For example, Y=AB is not the same as Y=A*B, but rather AB would be a variable in itself.
5. Expressions may be nested, that is, contained one within another. They may also appear in combination on the same level. However, the user should be aware that errors can often be made in the number of parentheses and commas. The only saving with nesting is in the number of cards punched; there is no saving in run time or storage. In brief, nesting should generally be avoided by the beginning user.
6. If an INTGRL function is included in an expression, it must be the rightmost part of that expression; for example, Z=Y + INTGRL (IC, X) is correct, but Z=INTGRL (IC, X) + Y is not correct.
7. The initial condition of an INTGRL function may not be an expression. If a parameter, it should be specified by an INCON data statement. If a variable, it must be computed in the INITIAL segment of the model.
8. The output of the INTGRL function may not be a subscripted variable.
9. The INTGRL function may not be used directly as the argument of any multiple-output function or MACRO.
10. No output of a multiple-output function may be a subscripted variable.
11. Certain names are reserved for system use and may not be used. A list of these is given under "Methods" in this manual.
12. With two exceptions, a statement may be continued on as many as eight cards, for a total of nine cards. Any card concluded with three consecutive decimal points is considered to be followed by a continuation card. Cards should not be continued in the middle of variable names or constants. The first card of a MACRO definition or MACRO use may be continued for a total of only four cards. Structure statement cards within a MACRO definition may not be continued.
13. An asterisk in cc 1 denotes a comments card. Comments cards can be used to insert appropriate reminders or explanations in the sequence of statements. The card will be printed with the input and then ignored.
14. Columns 73 - 80 are not processed by S/360 CSMP and may be used for identification.
15. Cards with a / (slash -- or, properly, virgule) in cc 1 are not processed, but the / is removed and the card is transferred, as is, to the UPDATE subroutine. A maximum of ten such cards can be processed. This additional feature can be used by those familiar with FORTRAN to insert FORTRAN specification statements into the problem definition. Continuation cards for statements with a / in cc 1 must be in the usual FORTRAN sense (that is, with a nonzero cc 6); these cards must also contain a / in cc 1. All / cards appear at the beginning of the UPDATE subroutine. Note that two S/360 CSMP variables cannot be EQUIVALENCEd.
16. **Blank cards can be used freely to space the input listing.**

DATA STATEMENTS

Data statements are used to assign numeric values to the parameters, constants, initial conditions, and table entries associated with the model. They can be used to assign numeric values to those variables that are to be fixed during a given run. The advantage of assigning variable names and using data statements to specify numeric values is that the latter can be changed, automatically, between successive runs of the same model structure. An example of a data statement is:

```
PARAMETER PAR1=2.97,RATE=550.0
```

where PARAMETER is the label identifying the card as a parameter card, PAR1 and RATE are the variables to be assigned values, and 2.97 and 550.0 are, respectively, the values assigned. The format for each assignment is variable name, equal sign, value to be assigned, and a comma if additional assignments follow. At least one blank must follow any card label -- in this case, PARAMETER. Each different type of data statement is identified by a different card label punched into the card. Data statements may appear in any order and may be intermixed with structure statements.

Examples of each type of data statement are shown in Table 3. They are used in the following ways:

1. PARAMETER, INCON, and CONSTANT

PARAMETER, INCON, and CONSTANT cards are used to assign values to variable names used as parameters, initial conditions, and/or constants. The three types may be used interchangeably. The value of the specified variable is set to the corresponding real or integer constant. Any number of variables may appear on a card or on continuation cards. In data statements, the variable name must be on the left of the equal sign. The name must conform to the specifications for variable names, and the numeric value must conform to those for constants. Blanks are not considered. A comma following a numeric value permits another assignment. A sequence of simulation runs may be designated by enclosing several values of the variable inside parentheses. For example, with X=(5.0, 5.5, 6.0, 6.5), four simulation runs will be made. X will be 5.0 in the first run, 5.5 in the second, 6.0 in the third, and 6.5 in the fourth. If increments of the variable are of equal size as in the example, the specification could be X = (5.0, 3*0.5). Only one multiple value parameter may be used for each sequence of runs; the sequence should be terminated by means of an END card. A sequence of simulation runs may also be specified by designating both individual parameter values and increments. For example, if X = (4.9, 5.0, 3*0.5, 7.2, 8.0, 4*0.2), the multiple value parameter, X, will sequence through eleven simulation runs. The maximum number of runs in any sequence is fifty.

PARAMETER	PAR1 = 4.98, X=(5.0, 5.5, 3*2.0)
CONSTANT	CON7 = 17.95, VEL = 8.7E5
INCON	IC = 9.92, AA = -1.2, AB = 1.79E - 3
FUNCTION	F1OFX = (45., 898.0), (48.7, 917.3), ...
OVERLAY	F2OFX = 17.3, 0.3, 17.9, 0.4, ...
TABLE	PRM(3) = 3.91, PAR1 (1-7) = 4.98,6*5.9

Table 3. Data statements

2. FUNCTION

This statement is used for specification of pairs of x, y coordinates for use by the function generator elements, AFGEN and NLFGEN. AFGEN (arbitrary function generator) and NLFGEN (nonlinear function generator) are used for simulating those portions of a model wherein some characteristic, Y = f(X), is available in tabular or graphic form versus the argument X. The values are converted and reserved in a table to be referenced by use of the name given it. The first value, and alternating values thereafter, are those of the independent variable and must be presented in order of algebraically increasing values. Increments may be of unequal size. Each of these must be followed by its corresponding coordinate value. The x, y pairs may, if desired, be enclosed within parentheses. A comma is required between each value but the parentheses are optional. This list of values may extend to continuation cards. There is no specific restriction on the number of function generators in a simulation or the number of points per function, since they are placed in simulator data storage (see "Program Restrictions").

3. OVERLAY

This statement permits modification of a previously specified table of x, y coordinates used with either an AFGEN or NLFGEN function generator element. The particular table is identified by using the same function name as first used with the FUNCTION statement. This feature can be used only if the overlaying table does not contain more x, y pairs than did the original specification.

4. TABLE

This feature allows blocks of data to be handled and transmitted more conveniently. Table values are converted and substituted for the current values of the corresponding dimensioned variables according to the specified index or consecutive indices. The form K*n causes K consecutive entries of the value n. A symbolic name for the table must appear on a STORAGE translation control statement. A complete description of the use of this feature is given under "Tabular Data" in this manual.

Data statements must be prepared in the following format:

1. Each data type must be identified by its specific label, such as PARAMETER, INCON, or FUNCTION. Labels do not have to start in cc 1, but must be followed by at least one blank.

2. Any card concluded with three consecutive decimal points is considered to be followed by a continuation card. A data statement may be continued on as many cards as necessary. Cards may not be continued in the middle of variable names or constants.

3. Alphabetic and numeric data may appear anywhere on the data card (or on a continuation card) following the label that specified the type of data; that is, data statements are free form, like the structure statements.

4. Columns 73-80 are not processed by S/360 CSMP and may be used for sequencing or identification.

5. The required format for PARAMETER, INCON, and CONSTANT cards is a variable name, followed by an equal sign, followed by a numeric. The numeric will be converted to a real or integer constant and treated as the current value of the preceding variable. Numerics may be integers or real numbers; the latter are identified by a decimal point and follow the rules for real constants. A minus sign must precede a negative number.

CONTROL STATEMENTS

These statements are used to specify certain operations associated with the translation, execution, and output segments of the program. Examples are to specify a certain variable as an integer instead of a real (floating-point) number, to specify the finish time for the run, or to specify the names of the variables to be printed. The control statements may be changed as readily as the data statements. Most of the control statements may appear in any order and may be intermixed with structure and data statements.

Control statements are, in general, "free form" and follow the same format rules as data statements. The only exceptions are COMMON, COMMON MEM, ENDDATA, ENDJOB, and ENDJOB STACK; these must begin in cc 1.

TRANSLATION CONTROL STATEMENTS

Translation control statements specify how the structure and data statements are to be translated. Each of the different types of translation control statements is identified by a different label. Examples of each type are shown in Table 4. Recommended practice is to order the statements such that RENAME, FIXED, MEMORY, HISTORY, and STORAGE occur, when appropriate, before MACRO definitions. MACRO definitions must be entered before any structure statements.

The various statements are used as follows:

RENAME	TIME = DISP, DELT = DELTX
FIXED	K, COUNT, NUMBER
MEMORY	RHO(9), PHI(3), GADGET
HISTORY	PAR1(4), PAR7(13)
STORAGE	IC(6), PARAMS(30)
DECK	
MACRO	X1 , X2 = FCN(IN1, IN2, IN3)
:	
ENDMAC	
INITIAL	
DYNAMIC	
TERMINAL	
END	
CONTINUE	
SORT	
NOSORT	
PROCEDURE	X,Y = FUNCT(A, B, X)
:	
ENDPRO	
STOP	
ENDJOB	
ENDJOB STACK	
COMMON	
COMMON MEM	
DATA	
ENDDATA	

Table 4. Translation control statements

1. RENAME

By means of this card, certain reserved names can be altered if they do not adequately describe the variables for a specific problem. The substitute names appear on all output and should be used in the user's structure statements. The six reserved names that may be changed are TIME, DELT, DELMIN, FINTIM, PRDEL, and OUTDEL. Successive renamings on the same card must be separated by a comma. At least one blank must follow the card label, RENAME. No continuation cards (...) may be used with the RENAME card; however, multiple RENAME cards may be used if needed. If there is more than one substitute name for a reserved name, the last substitute name given the reserved name will govern. RENAME cards must appear before the TIMER card.

2. FIXED

This allows the user to declare the listed variables as being fixed-point numbers (integers), instead of real (floating-point) numbers, within the translated program. The variables can then, and only then, be used as integers in S/360 CSMP structure and/or FORTRAN statements. FORTRAN rules for fixed-point operations apply to computations using these variables. In addition, names of integer FORTRAN functions, beginning with I, J, K, L, M, and N, used in structure statements must appear on a FIXED translation control card.

3. MEMORY

This card is required when the user defines his own MEMORY functions. It must be used to notify the translator that the user-defined functional blocks named on the card are memory functions and, therefore, require special handling. A memory function is one in which the output depends only on past values of the input and output. The number within the parentheses specifies the number of storage locations needed to save the total number of past inputs and outputs that influence the current output. This number of locations is required each time the function is used in a simulation. The sorting algorithm also requires the identity of all memory functions. Since the output of a memory functional block at any time is independent of the input variable at that instant, such functions can initiate a computational sequence. A MEMORY translation control card must appear before the first reference to the function.

A memory element is sometimes implemented by a PROCEDURE which itself may be within a MACRO definition. For proper sorting the translator must be informed that it is a memory element, but the PROCEDURE does not require assignment of additional storage. For such cases, the name of the PROCEDURE must be entered on a MEMORY statement without the set of parentheses.

4. HISTORY

This card is required when the user defines his own HISTORY functions. It must be used to notify the translator that the user-defined functional blocks named on the card are history functions and, therefore, require special handling. A history function is one in which the output depends on both past values of the input and output and the present value of the input. As in the MEMORY functions, the number within the parentheses specifies the number of storage locations required by the function each time it is used within the structure statements. A HISTORY translation control card must appear before the first reference to the function.

5. STORAGE

This feature allows the user to specify that certain variable names, which appear on the card, are subscripted. The number within parentheses must be the maximum number of storage locations necessary to contain data for the corresponding variable. Data is entered into these areas by use of the TABLE data statement.

6. DECK

This option permits the user to request that a translated deck be punched for the simulation run; included are the UPDATE and user-supplied subprograms, literals of the symbol table used in the execution phase, and all data, execution, and output control statements. The translated deck can be compiled and executed using a FORTRAN compile-load-and-go procedure, bypassing the translation phase of S/360 CSMP for subsequent runs. The expert user may even wish to introduce minor modifications directly in the translated deck.

The cards containing the literals of the symbol table and the various data and control statements must be prepared as a separate data set to be read at execution time. An object deck can be obtained by compiling the translated deck, and can be used with the aforementioned data set. An illustration of the use of this object deck is given in the System Manual (Y20-0111).

If a model is known to be in final form, so that no changes will be made in the translated FORTRAN program, an even more efficient method is available. If the label DECK SYMBOLS is used, and the OS/360 control cards are modified as described in the Operator's Manual (H20-0368-2), the execution phase load module as well as the corresponding symbol table for the model will be stored under unique data set names. Subsequent simulation runs may thereby be executed without again performing translation, compilation or link editing.

7. MACRO

•

•

•

ENDMAC

These labels are used to identify a group of statements that define a MACRO. This feature allows the user to build larger functional blocks from the basic S/360 CSMP and FORTRAN functions. Once defined, a MACRO can be used any number of times in the model, just like any other function. A full discussion of this feature is given under "Modeling Techniques". MACRO block definitions must be placed in the deck before any structure statements, including any in an initialization section.

8. INITIAL

This label identifies the beginning of the INITIAL segment of the model; the segment is terminated by the DYNAMIC statement. The structure statements within the INITIAL segment are executed only at TIME equals zero. This option provides a convenient means for an initializing computation of those initial conditions and parameters which are themselves functions of other, perhaps more basic, parameters of the model. If sectioning of the segment is not explicitly indicated, it is assumed that the segment comprises a single <u>sorted</u> section. Note that MACRO definitions must precede the INITIAL statement.

9. DYNAMIC

This label identifies the beginning of the DYNAMIC segment of the model; the segment is terminated by the TERMINAL statement, or, if there is no TERMINAL segment, by the first END or CONTINUE statement. The structure statements within the DYNAMIC segment describe the dynamics of the model, and the corresponding computations are performed repeatedly, under control of the selected integration routine, during each run. If sectioning of the segment is not explicitly indicated, it is assumed that the segment comprises a single <u>sorted</u> section.

10. TERMINAL

This label identifies the beginning of the TERMINAL segment of the model; the segment is terminated by the first END or CONTINUE statement. The structure statements within the TERMINAL segment prescribe the computations to be performed upon completion of each run. The segment is automatically entered when the TIME equals FINTIM or when any FINISH condition is satisfied. Unless sectioning is explicitly indicated, all structure statements within the segment are assumed to comprise a single unsorted section.

11. END

The first occurrence of this statement defines the completion of the structural description of the model, as well as the data and control specifications for the first run. Subsequent uses of this statement terminate the specifications for successive runs. As explained later under "Run Control", an END card permits the simulation to accept new data and control statements for another run which is automatically initiated at the conclusion of the preceding run. The END card resets the independent variable (TIME) to zero and resets initial conditions. The last use of the END statement must be followed by STOP.

12. CONTINUE

If the run is to continue from the point at which the previous run ended, the CONTINUE card is used in place of the END card. When restarted after data or control statement input, the program does not reset initial conditions or the independent variable (TIME), but continues from the point at which the previous run ended. The major advantage of the CONTINUE feature is that it allows a control statement to be changed during a simulation. Run control by means of either a multiple-valued parameter on a PARAMETER, INCON, or CONSTANT card, or a TERMINAL section, should not be used in conjunction with the CONTINUE card. If CONTINUE follows a FINISH condition, PRDEL and OUTDEL will be incremented from the TIME at which the FINISH condition was encountered, even if this TIME was not a multiple of PRDEL and OUTDEL as originally specified.

Note also that the FORTRAN statement CONTINUE, which is valid within procedural portions of S/360 CSMP, must be used with a statement number to avoid confusion with this label.

13. SORT

The SORT statement defines the beginning of a section and declares that all structural statements within the section are to be considered parallel. Thus, the structure statements within a SORT section are sorted by the translator in accordance with the sorting algorithm. This requires that current values for all variables appearing on the right-hand side of a structure statement must be available before the statement can be processed. The section is terminated by definition of the next section (indicated by SORT or NOSORT) or by the next segment (indicated by DYNAMIC or TERMINAL).

14. NOSORT

The NOSORT statement defines the beginning of a section and declares that all structure statements within the section are to be considered procedural. Thus, the structure statements within a NOSORT section are not sorted but, instead, are kept in their logical order as supplied. Full use of FORTRAN conditional branching is permissible within such a NOSORT section. The section is terminated by definition of the next section (indicated by SORT or NOSORT) or by the next segment (indicated by DYNAMIC or TERMINAL).

15. PROCEDURE

•

•

•

ENDPRO

These labels provide a convenient means for using the logic capabilities of FORTRAN in defining new functions. Statements included between the cards labeled PROCEDURE and ENDPRO are not sorted internally but are treated as a single functional entity by the sorting algorithm. A full discussion of the use of this feature is contained under "Modeling Techniques".

16. STOP

This card must follow the last END card in a sequence of simulation runs of the same model.

17. ENDJOB

This card must be used to signify the end of a job. If there are user-supplied FORTRAN subroutines, this card must follow them; if not, it must follow the STOP card. The label ENDJOB must be in cc 1-6.

18. ENDJOB STACK

This card can be used instead of the ENDJOB card when another S/360 CSMP job follows. For a series of short jobs, its use will increase efficiency and throughput. A blank card must follow this card. The label ENDJOB must be in cc 1-6 and STACK in cc 9-13. Note that this feature may be used only if the operating system uses the card reader for SYSIN. An alternate method for stacking simulation jobs is described in the Operator's Manual (H20-0368) for any other configurations.

19. COMMON

This feature allows user-supplied routines access to data in the previously established S/360 CSMP COMMON. The label is placed at the beginning of the user's routine, to indicate to the translator that access to the COMMON used for UPDATE is necessary. The translator will replace this card with the COMMON statements needed. Cards entered into UPDATE by the / option are not included. COMMON must appear in cc 1-6.

20. COMMON MEM

This label makes that portion of COMMON used for history, memory, and implicit functional blocks available to user-defined programs of those types. The translator will replace this card with the necessary statements. COMMON must appear in cc 1-6 and MEM in cc 9-11.

21. DATA

●

●

●

ENDDATA

These labels identify a set of cards as input data that will be entered by means of a FORTRAN statement READ(5,xxx). The labels advise the S/360 CSMP translator to skip over these cards without checking their syntax, since they are not S/360 CSMP statements. The READ statement is usually placed within a NOSORT section of the INITIAL segment. The user is responsible to check that the number of such data cards and their format is consistent with the READ and FORMAT statements. Note especially that the label ENDDATA must be in cc 1-7. The label DATA should immediately follow the END statement for the particular run.

EXECUTION CONTROL STATEMENTS

Execution control statements are used to specify certain items relating to the actual simulation run -- for example, run time, integration interval, and relative error. An example of an execution control statement is:

```
TIMER  FINTIM=8.0,DELT=.02
```

where TIMER is the label identifying the card as a timer card, FINTIM and DELT (integration interval) are the system variables to be assigned values and 8.0 and .02 are the values assigned. Successive assignments on the same card must be separated by a comma. At least one blank must follow the card label. Each of the different types of execution control statements is identified by a different card label. Examples of each type are shown in Table 5. They are used as follows:

1. TIMER

This feature allows the user to specify the values of system variables. The current value of the specified system variable is replaced by the corresponding integer or real number. If there is more than one value for a TIMER reserved word for a simulation run, the last value given will be used. User specifications are automatically adjusted by the program, as necessary, to ensure a consistent relationship between integration interval, run time, and output intervals. The system variables that may be set are:

Name	Description
PRDEL	Print increment for output printing. If both PRDEL and OUTDEL are required, the smaller is adjusted, as necessary, to be a submultiple of the larger. If a PRDEL is required but has not been specified, it is set equal either to OUTDEL if OUTDEL is required and has been specified, or to FINTIM/100.
OUTDEL	Print increment for the print-plot output, and preparation of a data tape for user-prepared plotting programs. If both OUTDEL and PRDEL are required, the smaller is adjusted, as necessary, to be a submultiple of the larger. If an OUTDEL is required but has not been specified, it is set equal either to PRDEL if PRDEL is required and has been specified, or to FINTIM/100.
FINTIM	Maximum simulation value for the independent variable. This must be specified for each simulation. FINTIM is adjusted, as necessary, to be the highest multiple of the most frequent output that would occur within the originally specified FINTIM.
DELT	Integration interval or step size for the independent variable. If DELT is specified, the program adjusts it, as necessary, to be a submultiple of PRDEL or OUTDEL, whichever is smaller. If neither a PRDEL nor OUTDEL has been specified, DELT is adjusted, as necessary, to be a submultiple of FINTIM/100. If DELT is not specified, the program first attempts to assign it a value equal to 1/16 of PRDEL or OUTDEL, whichever is smaller.
DELMIN	Minimum allowable integration interval or step size for variable-step integration methods. It is taken as FINTIM x 10^{-7} if unspecified.

```
TIMER    DELT=.02,FINTIM=10.0

FINISH   ALT=0.0,X=5000.0,X=Y

RELERR   XDOT=5.0E-5,X=1.5E-4

ABSERR   X2DOT=4.0E-3

METHOD   MILNE
```

Table 5. Execution control statements

2. FINISH

This label allows the user to specify terminating conditions in addition to FINTIM. A run can be ended when any dependent variable reaches or first crosses some specific bound. In the example in Table 5, the problem terminates if ALT reaches 0. or X reaches 5000., before specified FINTIM has elapsed. X=Y can also be specified on a FINISH card, so that the run will terminate when X is equal to Y or the difference between X and Y changes sign, where both X and Y are variable names. Up to ten equivalences can be placed on the FINISH card(s). Each use of the FINISH label overrides any previous FINISH specifications. A RESET card must be used to nullify FINISH specifications between successive runs. FINISH conditions are checked at each integration interval. It is, therefore, possible that the run will terminate at a time which is not a multiple of PRDEL or OUTDEL. In this case, printing will occur at the FINISH time.

3. RELERR

This feature allows the user to specify a relative error for each integrator output. It is used only for the RKS and MILNE integration routines, which are allowed to vary the integration interval to satisfy error bounds. If relative error is specified for any integrator, the last error specified before an END or CONTINUE card is applied to all integrators that are unspecified. If none is specified, the error is set at 0.0001. RELERR specifications are additive; that is, use of the RELERR label does not override all previous RELERR specifications. A change in the relative error for a particular integrator can be made by specifying the desired error and integrator output name on a new RELERR card. The RESET feature can be used to nullify all previous RELERR specifications. For further information see "Modeling Techniques" and "Integration Techniques".

4. ABSERR

This is similar to RELERR but is used to control absolute error in the RKS method only. If none is specified, the error is set at 0.001. As shown under "Integration Techniques", the maximum permissible error for each integrator is the sum of the permissible absolute and relative errors. Thus as the integrator output approaches zero, the permissible absolute error dominates. As the integrator output grows in magnitude, the permissible relative error dominates.

5. METHOD

This label specifies the particular centralized integration routine to be used for the simulation. If none is specified, the RKS method is used. Names must be exactly as shown below:

Name	Method
ADAMS	Second-order Adams integration with fixed interval
CENTRL	A dummy routine that may be replaced by a user-supplied centralized integration subroutine, if desired
MILNE	Variable-step, fifth-order, predictor-corrector Milne integration method
RECT	Rectangular integration
RKS	Fourth-order Runge-Kutta with variable integration interval; Simpson's Rule used for error estimation
RKSFX	Fourth-order Runge-Kutta with fixed interval
SIMP	Simpson's Rule integration with fixed integration interval
TRAPZ	Trapezoidal integration

OUTPUT CONTROL STATEMENTS

Output control statements are used to specify such items as the variables to be printed and/or print-plotted. An example of an output control statement is:

```
PRINT  X, XDOT, VELOC
```

where PRINT is the label identifying the card as a print card, and X, XDOT and VELOC are the variables to be printed. A comma must be inserted before each successive variable name. A card without continuation marks signals the end of the list. Each of the different types of output control statements is identified by a different label. Illustrations of the output capabilities are given later under "Data Output". Table 6 shows each type of output control statement:

1. PRINT

The PRINT card is used to specify variables whose values will be printed at each PRDEL interval during the simulation. If any variables are specified, printout of the independent variable (TIME) is forced. Variable names are printed in printout headings. The PRINT control card is intended for use with real variables. If output of integer variables is desired, they should first be equated to real variables; printing of the real variable should then be requested. For example, if printing of the fixed variable I is desired, the user must provide the following statement:

```
X=I
PRINT X
```

The label PRINT should appear only on the first PRINT card if more than one card is needed to specify all the outputs. As in other cards, three periods (...) identify continuations. If more than one card or group of cards is identified by the label PRINT, the last in sequence will be used. Up to 49 variables may be specified for one simulation run. PRINT specifications can be nullified by use of the RESET.

PRINT	X, XDOT, ALT
TITLE	PROBLEM DESCRIPTION
PREPARE	DIST, VELOC
PRTPLOT	X(Y1, Y2), Z(3.0, 4.0, Y3), W
LABEL	PRINT PLOT PAGE HEADING
RANGE	ALT, DIST
RESET	PRINT, PRTPLOT, FINISH

Table 6. Output control statements

2. TITLE

The TITLE statement allows the user to specify a heading that appears at the top of each page of printed output. The heading consists of those symbols between the label TITLE and cc 72 of that statement. The first nonblank character following the label TITLE must be either alphabetic or numeric; special characters may, however, be used within the remainder of the heading.

The S/360 CSMP continuation device (...) is not permitted with TITLE statements. However, a maximum of five TITLE statements may be used per run. Multiple statements result in successive headings on each page; thus, several lines of heading are easily obtained.

TITLE specifications cannot be nullified by RESET. However, all previous TITLE specifications are nullified by the first use of the TITLE label after an END or CONTINUE card. Subsequent TITLE cards within this same run are then additive.

3. PREPARE

The PREPARE card allows the user to specify variables to be written on OS/360 I/O device 15 at the specified OUTDEL interval, during the simulation run, for possible later plotting. If any variables are specified, output of the independent variable (TIME) is forced. The label PREPARE should appear only on the first PREPARE card if more than one is required. Continuation cards are allowed. If the label PREPARE identifies more than one card or set of cards, the last in sequence will be used. Information specified on TITLE cards will also be written on the PREPARE data set. PREPARE specifications can be nullified by use of RESET. Up to 49 variables may be specified.

4. PRTPLOT

The PRTPLOT card(s) allows the user to specify which variables are to be print-plotted. Up to ten PRTPLOT cards can be used for each simulation run, but continuation cards are not allowed. A number of options are available within the framework of this feature, as illustrated in Table 6. Variables not enclosed within parentheses will be print-plotted as well as printed, and each such variable will cause a separate print plot. Variables inside the parentheses (up to a maximum of three) will be printed adjacent to the corresponding print plot. Print plotting of integer variables requires the same procedure described for the PRINT control card. Constants inside the parentheses specify lower and upper bounds for the corresponding print plot. If only a lower or an upper bound is desired, commas must be used to indicate which. For example, Z(3.0,,Y3) indicates only a lower bound of 3, and Z(,4.0,Y3) indicates only an upper bound of 4. The example in Table 6 shows both lower and upper bounds. Note that commas are not needed if neither bound is specified. Successive print-plot designations are separated by commas. PRTPLOT cards are additive, up to the maximum of ten; a RESET card must, therefore, be used to nullify previous specifications. All print plots requested on the same card will use the same LABEL card, as explained below.

5. LABEL

The LABEL statement allows the user to specify a heading for each page of print-plot output. The first nonblank character following the label LABEL must be either alphabetic or numeric; special characters may, however, be used within the remainder of the heading.

The S/360 CSMP continuation device (...) is not permitted with LABEL statements. However, a maximum of ten LABEL statements may be used per run. The first LABEL statement is associated with all print-plots requested by the first PRTPLOT statement; the second with the second, and so on. If PRTPLOT statements outnumber LABEL statements, the excess print-plots will not have labels. LABEL statements are additive, as are PRTPLOT statements, and so must be RESET to nullify previous designations. The name of the print-plot variable will also appear automatically in the print-plot page heading. A parameter name and its value will be printed if the multiple-value parameter feature is used.

6. RANGE

The RANGE card allows a user to obtain the minimum and maximum values reached, during the simulation, for specified variables. Ranges are automatically taken for all PREPARE and PRTPLOT variables. The total number of different variables that may appear on RANGE, PREPARE, and PRTPLOT cards is 100. The names of the specified variables, their minimum and maximum values, and the time of occurrence of each are printed at the integration interval. Only the first RANGE card in a sequence should be so labeled. Continuations are allowed. If there is more than one card or set of cards labeled RANGE, the last in sequence will be used. RANGE specifications can be nullified by use of RESET. The values provided in the case of a multiple parameter or TERMINAL-controlled sequence of runs are those for the entire sequence. If a CONTINUE card is used, the values provided are those obtained in the time period prior to termination by the FINISH or FINTIM condition.

7. RESET

A RESET card allows the user to conveniently nullify specifications previously given on certain control statement cards and used in a previous run of a sequence of runs. It can be used with PRINT, PREPARE, RANGE, LABEL, PRTPLOT, RELERR, ABSERR, and FINISH. Its use with RELERR, ABSERR, and FINISH has already been explained under these respective headings in the section entitled "Execution Control Statements". RESET PRTPLOT or RESET PREPARE nullifies all previous PRTPLOT, PREPARE, and LABEL specifications. RESET LABEL nullifies only previous LABEL specifications. A RESET card with only RESET on it is interpreted as nullifying all previous PRINT, PREPARE, PRTPLOT, RANGE, and LABEL statements. Note again that TITLE cannot be RESET. To ensure proper RESETing, the RESET card should be placed immediately after the END or CONTINUE card.

USER-DEFINED FUNCTIONS

In some simulation problems, the mix of functional blocks available from the S/360 CSMP library may not be sufficient to describe the problem adequately. The user, therefore, has been provided with the facility for building his own special purpose functional blocks. These functions may range from a few nonlinear statements to an extremely complex model of a complete plant in a process control problem. To define special purpose functions, either S/360 CSMP functional block statements or FORTRAN, or a combination of both, may be used. Three different types of functions may be prepared by the user: MACROs, PROCEDUREs, and subprograms. They differ somewhat in their use and the way in which they are handled in the S/360 CSMP program.

These several methods for building special functions give the user a high degree of flexibility for different problem areas. For example, by properly preparing a set of special functions and adding them either to the S/360 CSMP library or to the data deck at run time, the user is able to restructure S/360 CSMP into a problem-oriented language geared to any special purpose field in continuous system simulation.

MACRO FUNCTIONS

The MACRO type of function-defining capability in S/360 CSMP is a particularly powerful feature of the language. It allows the user to build larger functional blocks from the basic functions available in S/360 CSMP, and, thereby, to identify, as a parallel functional entity, a subsection of a simulation block diagram or the corresponding subset of structure statements. Once defined, a MACRO function may be used any number of times within the simulation structure statements.

As an illustration of this feature, consider a control system simulation study that involves several transfer functions (s is the Laplace operator) with differing parameter values, but all having the general form:

$$\frac{Z(s)}{X(s)} = \frac{s^2 + as + b}{s^2 + cs + d}$$

A simple technique for modeling such transfer functions is to define a new variable Y such that

$$\frac{Y(s)}{X(s)} = \frac{1}{s^2 + cs + d}$$

where the numerator is unity but the same denominator is used. Then, solving for the highest-order derivative of Y, one obtains

$$s^2 Y = X - c\,sY - d\,Y$$

Note that this is equivalent to the time domain statement

$$\ddot{Y}(t) = X(t) - c\,\dot{Y}(t) - d\,Y(t)$$

which is readily modeled by the S/360 CSMP statement

```
S2Y = X - C * SY - D * Y
```

Then by two successive integrations we obtain Y(t). Since

$$Z(s) = s^2 Y + a\,s\,Y + b\,Y$$

we finally obtain Z(t) by combining terms with the following S/360 CSMP statement

```
Z = S2Y + A * SY + B * Y
```

The user may define a MACRO to represent this general functional relationship, assigning it some unique name--for example, FILTER. The S/360 CSMP statements to define this new MACRO might be as follows:

```
MACRO      Z = FILTER( A, B, C, D, X )
         S2Y = X  - C*SY - D*Y
          SY = INTGRL( 0.0, S2Y )
           Y = INTGRL( 0.0,  SY )
           Z = S2Y + A*SY + B*Y
ENDMAC
```

Such MACRO definition cards must be placed at the beginning of the S/360 CSMP deck before any structure statements in the initialization or dynamic segments. Several rules must be observed in defining MACRO functions:

1. The first card of a MACRO definition is the translation control card containing the word MACRO and the canonical form that the user assigns for the new function. The names used to represent variables in the definition statements are merely "dummy" variables; they are automatically replaced by the corresponding variable names assigned when the MACRO is used. The required format is as follows:
   a. If the MACRO function has a single output variable, a dummy name for this variable is placed to the left of the equal sign. If the MACRO function has several output variables, these must all appear to the left of the equal sign, separated by commas.
   b. The unique name assigned by the user to the function must appear to the right of the equal sign. This is the name by which the function will subsequently be used in structure statements.
   c. Dummy names to represent initial conditions, parameters, and input variables for the MACRO must appear within a pair of parentheses following the function name. Commas must be used to separate names within the parentheses.
2. The last card of a MACRO definition contains merely the word ENDMAC.

3. The statements between the MACRO and ENDMAC translation control cards specify structure. Thus, S/360 CSMP data or control statements may not appear. Any S/360 CSMP or FORTRAN structural statement may be used except as follows:
   a. A structure statement must be completed on a single punched card; the S/360 CSMP continuation device (...) is not applicable within MACRO definitions.
   b. FORTRAN control and input/output statements may not be used unless embedded within a PROCEDURE function. An input/output statement may reference a FORMAT statement, but the FORMAT statement itself must not appear within the MACRO. Good practice is to place all FORMAT statements in a TERMINAL statement.

In general, the statements within a MACRO definition may appear in any order, since the program assumes that they represent parallel structure and will automatically sort them to determine the computational sequence. However, a MACRO may be used within a PROCEDURE function or within a NOSORT section. In both of these cases, the sorting algorithm would not be applied to the MACRO. For such use, therefore, the MACRO must be defined in appropriate procedural order and cannot invoke a MACRO function.

Each use of a MACRO requires a structure statement that has exactly the same format as the canonical statement in the MACRO definition. Thus, the number and order of the inputs, outputs, and parameters must exactly agree with the definition. A MACRO function cannot be used as part of an expression in a structure statement. This restriction pertains not only to user-defined MACROs but also to the system MACROs: REALPL, CMPXPL, MODINT, and LEDLAG. As example, the canonical form for REALPL as illustrated in Table 1 is as follows:

```
    X1 = REALPL( IC, P, X )
```

It would therefore be invalid to attempt its use in the following ways:

```
  X1 = GAIN * REALPL( 0.0, 5.0, Z )
```

or

```
  X2 = REALPL(X20,TC,IN) + INTGRL (IC2,YIN)
```

in each of these examples an expression involving REALPL is used rather than the primitive canonical form.

As an example of the proper use of a MACRO, consider the following structure statement, which uses the FILTER function defined previously:

```
SIGOUT = FILTER( 0.1, PAR23, N2, 34.6, SIGIN )
```

This statement declares that the input to the function is the variable SIGIN, that its output is named SIGOUT, that two of its parameters are literals, and that two are named parameters. The translator portion of S/360 CSMP would automatically expand this structure statement, in accordance with the MACRO definition, as follows:

```
ZZ0027 = SIGIN - N2*ZZ0028 - 34.6*ZZ0029
ZZ0028 = INTGRL( 0.0, ZZ0027 )
ZZ0029 = INTGRL( 0.0, ZZ0028 )
SIGOUT= ZZ0027 + 0.1*ZZ0028 + PAR23*ZZ0029
```

Note that the dummy variables of the definition have been replaced by the corresponding literals and named variables of the structure statement. The translator has assigned unique names ZZ0027, ZZ0028, and ZZ0029 to the intermediate variables S2Y, SY, and Y of the definition; the next time the program is required to generate a unique name for a variable, it will use ZZ0030. As noted in the section entitled "Reserved Words", the names ZZ0000--ZZ9999 are reserved to the program and may not be assigned by the user.

Significant economies of effort as well as conceptual advantages can sometimes be realized by invoking MACROs within the definitions of other MACROs, first defining "primitive" MACROs for the most basic phenomena, then combining these in MACRO definitions for progressively more complex portions of model structure. For example, a physiologist concerned with the distribution, binding, and metabolism of hormones in blood plasma might first develop the following MACRO to represent a simple reversible reaction:

```
MACRO   COUT, RATE = COMP( IC, KA, KR, CIN )
              RATE = KA * CIN  - KR * COUT
              COUT=  INTGRL( IC, RATE )
ENDMAC
```

The outputs of this MACRO represent the concentration and time derivative of the reaction product. To represent a bi-molecular reaction, the MACRO may be invoked using as the fourth argument the product of concentrations of the two reactants. He might then develop the following MACRO definition, invoking "COMP" twice since the particular hormone is bound in reversible reactions with two different proteins, and including terms to represent distribution and metabolism:

```
MACRO  CCP, CAP, CTP, CCAP, CCTP, TOTCCP = DBM( QCORT, INFUS )
              MASSFL = QCORT + INFUS  - DIFFUS - 0.63 * CCP
                 CCP = INTGRL( 1.77, MASSFL/V1 -RATECA -RATECT  )
                 CAP = LIMIT( 0.0, A, A - CCAP )
      CCAP, RATECA = COMP( 2.83  , KAA, KAR, CCP * CAP )
                 CTP = LIMIT( 0.0, T, T -  CCTP )
      CCTP, RATECT  = COMP( 9.27  , KTA, KTR, CCP * CTP  )
           DIFFUS   = K12 * CCP  - K21 * CC2
                 CC2 = INTGRL( 1.77, DIFFUS / V2 )
           TOTCCP = CCP  + CCAP  + CCTP
ENDMAC
```

PROCEDURE FUNCTIONS

The PROCEDURE type of user-defined function is used for procedural coding and allows simple application of the capabilities of FORTRAN. Any S/360 CSMP structure statement or executable FORTRAN statement except another PROCEDURE may be used in a PROCEDURE. Considerable power can be realized fairly readily through the use of FORTRAN's logical, branching, and subscripted variable features.

An example of a PROCEDURE function is:

```
PROCEDURE  VALUE = BLOCKA( PAR1, PAR2, TRIG, IN
           IF( TRIG ) 1, 1, 2
        1  VALUE = LIMIT( PAR1, PAR2, IN )
           GO TO 3
        2  VALUE = IN
        3  CONTINUE
ENDPRO
```

This example defines a new functional element, BLOCKA. This particular function has only a single output, here named VALUE; PROCEDUREs are permitted to have multiple outputs (see example INVERT below). The inputs and parameters that make up the arguments of function BLOCKA are listed within parentheses on the right side of the first card; this permits the sorting algorithm to properly process the new function.

The input variables TRIG and IN are listed as function arguments within parentheses on the right side of the first card of the PROCEDURE. While not required, it is good practice to also include as arguments all parameters used within the PROCEDURE. Thus PAR1 and PAR2, which are parameters of the LIMIT function, are included as arguments of function BLOCKA. Unless this is done, the translator cannot check on whether internal parameters have been properly defined by data statements and is thereby unable either to initialize such unspecified parameters to zero value or to generate a diagnostic message regarding their unavailability.

The set of statements making up a PROCEDURE function may be written anywhere in the sequence of S/360 CSMP statements. Several rules must be observed in preparing PROCEDURE functions:

1. The first card of the PROCEDURE is the translation control card containing the word PROCEDURE and the names of the function, input variables, output variables, and parameters. The output names must be placed to the left of the equal sign and be separated by commas. The function name, which is a "dummy" name, must appear to the right of the equal sign. Names of the input variables must appear within a pair of parentheses following the function name. Commas must be used to separate names within the parentheses.

2. The last card of the PROCEDURE function contains merely the word ENDPRO.

3. The statements describing the PROCEDURE are placed between the PROCEDURE and ENDPRO translation control cards.

4. Variables defined within a PROCEDURE function are not available for data output by means of the PRINT, PRTPLT, RANGE, or PREPARE option unless they appear as output names on the PROCEDURE card.

The PROCEDURE is conceptually a single functional element, even though its definition requires a number of statements. During sorting, the statements that describe a PROCEDURE are treated as a group and the entire set is moved around as an entity in order to satisfy the input/output sequencing requirements of the sorting algorithm. There is no internal sorting of statements within a PROCEDURE. If the operation performed by the PROCEDURE is required only once, the PROCEDURE is defined where needed within the appropriate section of the INITIAL or DYNAMIC segment. Note that the PROCEDURE is specifically designed for use in parallel, sorted sections of the INITIAL or DYNAMIC segments. Use of a PROCEDURE within a NOSORT section is inconsistent modeling and will result in an abnormal exit. Since the TERMINAL segment is normally procedural, there is seldom need for a PROCEDURE within that segment.

Although FORTRAN logical, branching, and input/output statements cannot generally be used within a MACRO since parallel structure is the basic premise of the MACRO, it is permissible to use PROCEDUREs within MACROs. This feature of S/360 CSMP permits the modeler to define MACROs involving both parallel and procedure structure and invoke the MACRO as often as required. **A MACRO definition may include multiple PROCEDUREs as well as multiple invocations of other MACROs (which latter may include additional PROCEDUREs or MACROs). The restriction is that a** PROCEDURE within a MACRO definition may not itself invoke another MACRO. Repeated invocation of a MACRO will cause the translator to generate the complete set of statements according to the pattern given in the MACRO definition. The sorting algorithm will ensure that an appropriate statement sequence is obtained; those sets of statements that were specified as PROCEDUREs will, however, be treated as separate entities by the sorting algorithm. Any output name or statement number used within a PROCEDURE will be given a unique name or number, just like any other name or statement within a MACRO.

Often one wishes to define a MACRO that is entirely procedural, and for this case a simplified form may be used. It is only necessary that the first statement within the MACRO definition contain the label "PROCEDURAL"; the statement ENDPRO must not be used with this form. For example, one might need a delay function with nonzero initial conditions. Note that such an element is a memory element and its name should appear on a MEMORY statement. The following statements illustrate the simplified form for such an element:

```
MEMORY   DLAY

MACRO       OUT = DLAY( IC, N, PAR, IN )
PROCEDURAL
    IF( TIME .EQ. 0.0 )  OUT = IC
                X = DELAY( N, PAR, IN )
    IF( TIME .GT. PAR )  OUT = X
ENDMAC
```

More commonly, one needs a MACRO in which only a portion must be procedural. For example, a circuit designer might develop a MACRO to represent an electrical transformer. The state variables I1 and I2 represent the currents flowing in the two windings of the transformer. Their derivatives are functions of the applied voltages E1 and E2 and the inductance parameters L1, L2, and M. Computation of the derivatives requires inversion of a two-by-two matrix consisting of the parameter values; since these are normally constants, it is efficient to invert the matrix only once, when TIME is zero. One way to model the transformer is illustrated by the following:

```
MACRO     I1, I2 = TRANS( L1, L2, M, E1, E2 )

PROCEDURE  I1DOT, I2DOT = INVERT(  E1, E2 )
   IF( TIME )   1, 1, 2
  1              D = L1 * L2  - M * M
               B11  = L2 / D
               B12  = -M/ D
               B21  = B12
               B22  = L1 / D
  2        I1DOT  = B11 * E1   + B12 * E2
           I2DOT  = B21 * E1   + B22 * E2
ENDPRO
               I1  = INTGRL( 0.0, I1DOT )
               I2  = INTGRL( 0.0, I2DOT )
ENDMAC
```

SUBPROGRAMS

This approach actually adds little to the algebraic and logical modeling capabilities available through use of the procedural MACRO, and is more difficult to program. It does, however, permit the new functional block to be permanently added to the system library, thereby conveniently available to all users at a particular installation. Since the statements within the subprogram need not be processed by the S/360 CSMP translator, it is also sometimes possible to develop larger models than would otherwise be permissible. As a general rule, it is best to carefully test the performance of any new subprogram independently of the complete model for which it is being designed.

There are two forms of subprograms: the FORTRAN function and the FORTRAN subroutine. The function form is used in developing new functional blocks whenever the output of the block is to be a single variable. If the block is to generate multiple outputs, the subroutine form must be used.

In developing a new subprogram, the user has available the features of FORTRAN and, in addition, may use many of the standard S/360 CSMP functional elements. This combination gives the user a very versatile modeling capability. Specifically excluded from use within any subprograms are: INTGRL, IMPL, DELAY, DERIV, HSTRSS, RST, IMPULS, PULSE, ZHOLD, MODINT, REALPL, CMPXPL, LEDLAG, and any user-defined MACRO's.

To use a function, once defined, it is necessary merely to conform to the ordinary S/360 CSMP structure statement format in calling it. The arguments used must, however, agree in number, order, type, and length with the dummy arguments used in the subprogram definition. Note that literal constants are not permitted to be used directly as subprogram arguments in S/360 CSMP models.

At run time, all user-defined subprograms not cataloged in the S/360 CSMP library must be in FORTRAN and be placed directly behind the STOP card. They must be terminated by an ENDJOB card. Subprograms may be cataloged or added to the S/360 CSMP library by loading them with the FORTRAN library by means of an Operating System/360 procedure.

As an example of a simple FORTRAN subprogram, consider a function used to model a valve that permits only one-directional flow and includes some internal resistance. The complete FORTRAN function definition might be as follows:

```
FUNCTION VALVE ( RESIS, PRES1, PRES2 )
IF ( PRES1 - PRES2 ) 1, 1, 2
1  VALVE = ( PRES1 - PRES2 ) / RESIS
   GO TO 3
2  VALVE = 0.0
3  RETURN
   END
```

An S/360 CSMP structure statement to call this subprogram might be:

```
OUT = VALVE ( K, PR1, PR2 )
```

where the symbolic names PR1 and PR2 represent pressures across some specific valve of a large simulation model, K represents its internal resistance, and OUT represents the resultant flow. Note that the element VALUE might be used many times within such a model, each instance representing a distinct physical unit. Since this functional block involves only a few statements and very simple logic, the subprogram approach offers little advantage over use of a procedural MACRO. This example is therefore only illustrative; the subprogram approach is generally reserved for much more complex elements.

As noted, if a functional block generates multiple outputs, the FORTRAN subroutine form of subprogram must be used rather than the FORTRAN function form. To properly design such an element, the user must understand how the S/360 CSMP translator processes structural statements with multiple outputs. For example, suppose the user wishes to define a functional block FLIP with two outputs, a single trigger input, and an associated gain parameter. Ultimately he will use this block within the dynamic segment of a model with a structural statement such as:

```
      Y1, Y2 = FLIP ( GAIN, TRIG )
```

where the symbolic variables are appropriate to the specific model. In generating the derivative routine UPDATE, the S/360 CSMP translator will replace the user's structural statement with the following:

```
      CALL FLIP ( GAIN, TRIG, Y1, Y2 )
```

Note that the output variables of the original structural statement are placed by the translator on the right-hand side of the subroutine argument string without rearrangement of their order. In programming the subroutine, the arguments must be arranged in corresponding order. Thus, the initial FORTRAN statement for the subroutine might be:

```
      SUBROUTINE FLIP ( PARAM, XINPUT, OUT1, OUT2 )
```

where the symbolic names PARAM, XINPUT, OUT1, and OUT2 are used merely for definition of the subroutine and have no external meaning.

The S/360 CSMP function generators AFGEN and NLFGEN may be used within a subprogram if the names of all functions upon which they are to operate are used as arguments of the subprogram. As example, consider the design of a functional block VALUE, which includes a requirement for both linear and quadratic interpolation of associated data arrays defining two curves. With a specific model, one might then use the structural statement:

```
      OUT = VALUE ( FUNCT1, FUNCT2, INPUT )
```

and the associated data statements:

```
FUNCTION    FUNCT1 = (   0.0    ,   -10.5   )  ,...
                     (   3.5    ,    34.6   )  ,...
                     .
                     .
FUNCTION    FUNCT2 = ( -234.0   ,   5.34E-4)   ,...
                     (  437.4   ,   34.578)    ,...
                     .
                     .
```

Note that the names of the two curves, as specified on the FUNCTION statements, are used as arguments of the structural statement. The subprogram developed for the block VALUE must be prepared to use these arguments; as example, the following FORTRAN function definition might follow the STOP statement of the model:

```
          .
          .
    STOP
          FUNCTION VALUE ( X, Y, SIGIN )
          REAL NLFGEN
          .
          .
          VALUE = 0.6 * AFGEN(X, SIGIN)
         $         + 0.4* NLFGEN(Y, SIGIN)
          RETURN
          END
    ENDJOB
```

Note that when used within a subprogram, NLFGEN must be identified as real; otherwise, FORTRAN would assume it to be an integer-valued function. The user is reminded that within subprogram definitions all the normal rules of FORTRAN are applicable and all variable names beginning with the letters I, J, K, L, M, N, are assumed to be integer-valued.

If the new function involves past input or output values and thus needs unique storage locations for each use, the S/360 CSMP data deck must include a MEMORY or HISTORY translator control card, as appropriate. These cards tell the translator how many storage locations are required for each use, and must appear before the first reference to the function. The translator assigns unique storage in the COMMON area and inserts the index of the assigned location as the first argument of the subprogram.

The handling of a function that involves both the present and past values of the input is illustrated by another example. Consider a function MAGCOR, which involves one present input, one past input, and one past output value for each use. The S/360 CSMP translation control and structure statements required to use this function are:

```
HISTORY  MAGCOR(2)
                 Y=MAGCOR(Y0,P1,XIN)
```

where Y0 is an initial condition, P1 is a parameter, and XIN is the input.

The HISTORY card tells the S/360 CSMP translator how many history storage locations are required for each use of the specified function. The translator assigns the next available locations in COMMON, and inserts the index of the assigned location as the first argument of the calling statement for the subprogram. For example, if the next available storage were the twenty-fifth location in the particular COMMON area, the translator would generate the following statement to link to the subprogram:

```
Y=MAGCOR(25,Y0,P1,XIN)
```

The translator index that records the location of the next available storage would then be increased to 27, since two locations are required for this single use of the new element.

In developing the FORTRAN subprogram, the user must recognize that the translator will insert this storage location index as the first argument of the calling statement for memory and history blocks regardless of whether the FORTRAN function form or the subroutine form is used. The subprogram must be properly programmed to use this assigned storage. As an example, the FORTRAN definition of MAGCOR might be as follows:

```
      FUNCTION MAGCOR(K, YIC, PARAM, X)
COMMON MEM
      EQUIVALENCE(C(1), TIME)
      I=K+KPT1-1
      IF(TIME)4, 4, 5
    4 SYMB(I+1)=YIC
      GO TO 6
    5 SYMB(I+1)=FUNCT(X, SYMB(I), SYMB(I+1), PARAM, TIME)
    6 SYMB(I)=X
      MAGCOR=SYMB(I+1)
      RETURN
      END
```

In these statements, KPT1 is the system name for the first available COMMON storage location in the execution phase, SYMB is the system name for the storage area that includes HISTORY and MEMORY functions, FUNCT is the name of the user-defined FORTRAN function that operates on the indicated variables. Note that an EQUIVALENCE card with the name TIME equivalenced to COMMON location C(1) is necessary if reference is made to the independent variable, by the name TIME, in the program.

The KPT1, C, and SYMB variable names are automatically placed in COMMON by the system when the COMMON MEM card is used. The cards supplied by the system to replace COMMON MEM are:

```
COMMON DDUM1(64), C(8000), NALARM, DDUM2(12)
COMMON KPT1, DDUM3(405), KEEP, DDUM4(1316), SYMB(1)
```

Note that in programming a history or memory function, the user must explicitly provide for input of initial conditions--for example, the variable YIC in the problem above.

While not permissible to use the INTGRL statement itself within the definition of a subprogram, it is possible to achieve the same effect. This is done by means of the special "specification" form of the INTGRL statement, which advises the S/360 CSMP translator to assign some desired number of integrators for each use of the subprogram within the model. The translator assigns contiguous locations for the group of integrator outputs and also for the group of integrator inputs; thus operations using subscripted variables are readily performed within the subprogram.

As an illustration of this feature, consider a function subprogram to provide the same performance as the MACRO function named FILTER described earlier in this section. This second-order transfer function requires two integrators. The core location of the first of these can be passed to the subprogram by using the "specification" form of INTGRL as one of the arguments of the subprogram. The total number of integrators used for a particular model is available within COMMON as the variable named NOINTG. The core location of the input to the first of the two integrators is thereby offset from the location of the integrator by NOINTG locations.

A FORTRAN function subprogram to represent the second-order filter might be programmed as follows:

```
      FUNCTION FILTER( A, B, C, D, X, YARRAY )
      COMMON DUMMY(9798), NOINTG
      DIMENSION YARRAY(1)
C     NOINTG IS THE TOTAL NUMBER OF INTEGRATORS IN THE MODEL
C     INPUT TO ANY INTEGRATOR IS LOCATED NOINTG WORDS FROM ITS OUTPUT
C        IN S 360 CSMP COMMON
C     YARRAY(1&2) CONTAIN THE TWO INTEGRATOR OUTPUTS
C     YARRAY(1) REPRESENTS THE INTEGRAL OF THE HIGHEST ORDER DERIVATIVE
C     YARRAY(NOINTG + 1) CONTAINS THE CORRESPONDING INTEGRATOR INPUT
      YARRAY(NOINTG-1) = X -C*YARRAY(1) -D*YARRAY(2)
      YARRAY(NOINTG+2) = YARRAY(1)
              FILTER  = YARRAY(NOINTG+1) +A*YARRAY(1) +B*YARRAY(2)
      RETURN
      END
```

Note the correspondence of the executable statements within this program to those used during the discussion of the MACRO function. The subprogram must compute the inputs to the two integrators and store these values in the appropriate core locations, YARRAY (NOINTG+1) and YARRAY (NOINTG + 2). The integration mechanism of S/360 CSMP computes the values of the integrals and stores these in YARRAY (1) and YARRAY (2). The subprogram must compute the result as a weighted sum of the integrator outputs in correspondence with the terms of the numerator of the transfer function.

To use the new functional block within an S/360 CSMP model, a structure statement such as the following might be used:

```
SIGOUT = FILTER( 0.1, PAR23, N2, 34.6, SIGIN, INTGRL(,,2) )
```

Note, in particular, the use of the "specification" form of INTGRL as the final argument. This advises the S/360 CSMP translator to assign two integrators for this use of the FILTER block. If this were used several times within a simulation, additional integrations would be assigned for each instance. The location of the first integrator is thereby passed to the subprogram, which then knows the location of YARRAY (1) and, by using NOINTG, can obtain the corresponding integrator input location.

Note also that, for this application, the first two arguments of INTGRL were left blank. Names for the initial condition variables were not required; leaving the first variable blank is equivalent to assigning a literal constant of zero. Names for the integrator inputs were not required, since these are used only internal to the subprogram and do not communicate with the rest of the model. Used in this way, the INTGRL block creates zero initial conditions for all integrators in the array and assumes that all derivates are computed within the subprogram.

MODELING TECHNIQUES

Simulation is an approximation technique. The user should be as aware as possible of the approximations made in a particular simulation model. It is best to carefully record any conscious approximations to ensure that their effects will be considered during subsequent evaluation and interpretation of the simulation results.

The simulation model is developed by appropriately preparing S/360 CSMP structure, data, and control statements so as to correspond with the user's visualization of the phenomenon. Preparation of a block diagram or differential equation representation is a critical first step in successful use of the technique. Preparation of the S/360 CSMP statements does require modest amounts of time and attention, but the mechanics of this task should not be of primary concern. Rather, the user should continually ask himself whether the basic equations and/or diagrams truly represent his visualization of the phenomenon.

Included in this section are some comments on modeling techniques and pertinent features of S/360 CSMP.

SORTING

S/360 CSMP was designed with a nonprocedural input language to free the user from the task of correctly sequencing structure statements. The program will automatically sort user-supplied structure statements to establish a correct execution sequence -- that is, to yield a properly organized FORTRAN subprogram. If the user does not want automatic sorting, he can use the NOSORT option, PROCEDURE functional blocks, or his own subroutines. All of these features are explained later in this section of the manual.

A correct sequence ensures that each statement's output for time t is computed on the basis of statement input values for time t. An incorrect sequence would update a statement's output for time t with input values for time t $-\Delta t$. This incorrect sequence would introduce a phase lag that could seriously affect stability and accuracy of the solution. A correct sequence, therefore, is one in which the output of each statement in the sorted sequence can be computed using values either provided as initial input or previously computed in the current iteration cycle.

Since integration is performed between iterations, the outputs of all integrals are known values at the start of an iteration. The outputs of MEMORY functional blocks are also known values. Together with initial conditions, these provide a starting point for the sequence of computations.

Within each sorted section of the S/360 CSMP structure statements, the sorting algorithm operates upon the integrator elements in the order in which they occur in the input deck. That is, the algorithm first determines an appropriate computational sequence for all those functional elements that contribute to the input of the first integrator. Next, it determines the sequence for all additional computations necessary to obtain the input to the second integrator. This procedure is repeated until all the integrators in the particular sorted section have been considered. Finally, any elements within the section that have not already been added to the computational sequence are sorted and placed in sequence.

Statement sequencing can be illustrated by a simple feedback control system consisting of a forward control block that manipulates a process control plant. The block diagram for such a system is shown in Figure 9.

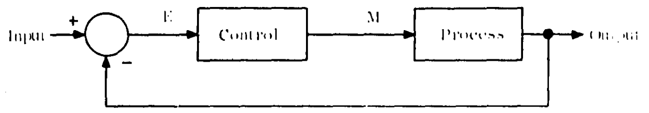


Figure 9. Feedback control system

Suppose that the plant's operation was represented as a single integration and the controller provided a signal equal to the sum of a constant times the error and the integral of the error (proportional plus reset type control). A set of structure statements defining the system could be:

```
MANIP=G*ERROR + INTGRL (IC2, ERROR)

ERROR=INPUT-OUTPUT

OUTPUT=INTGRL (IC1, MANIP)
```

These statements, however, are not in a proper sequence, because the computation of MANIP requires a knowledge of ERROR. Computation based on this sequence would introduce a phase delay. The correct sequence that would be produced by the automatic sorting algorithm is:

```
ERROR =INPUT-OUTPUT

MANIP=G*ERROR+INTGRL(IC2, ERROR)

OUTPUT=INTGRL(IC1, MANIP)
```

NOSORT OPTION

As noted previously, in some simulations it may be desirable not to sort certain sections of the problem configuration. S/360 CSMP provides a NOSORT option that bypasses the sorting phase for sections of S/360 CSMP coding identified by a NOSORT label. Thus, the user may include any type of procedural statement capability, such as branching on conditions and logical tests, within a sequence of either S/360 CSMP or FORTRAN statements.

Note at this point that either the PROCEDURE capability or the NOSORT option may be used to enter procedural coding. The two differ, however, in that the PROCEDURE function is handled by the system as a complete group of coding and is moved around and sorted within the S/360 CSMP structure statements as an entity. The NOSORT option, however, is used to identify a section of coding that divides the other structure statements into groups, each of which is separately sorted.

A NOSORT section may be used to test the run response for the purpose of switching portions of the configuration into or out of the simulation in order to decrease run time or alter the information flow. Problem structure variations that can be anticipated can thereby be included in a single run.

Figure 10 shows a center section of S/360 CSMP configuration coding identified with a NOSORT label. The purpose is to switch a portion of the flight simulation equations, the booster phase, out of the computation sequence after TIME has reached 50. The NOSORT block at the bottom of the statements is to keep the CONTINUE statement at the end.

```
                .....

                .....

                .....

     Statements for Trajectory Phase

                .....

                .....

                .....

NOSORT

             IF(TIME-50.)1,1,2

   1         CONTINUE

SORT

                .....

                .....

                .....

     Statements for Booster Phase

                .....

                .....

                .....

NOSORT

   2         CONTINUE
```

Figure 10. Illustration of NOSORT option

If no NOSORT card is present at the beginning of the structure statements, a SORT card is assumed; that is, sorting is automatic.

INITIALIZATION

Frequently a simulation involves initial conditions or parameters that are themselves functions of other variables or parameters. For example, the equations describing a chemical process might require the volume of a particular reaction vessel. To obtain maximum flexibility, the user might prefer to specify the dimensions of the vessel as parameters, rather than specify the volume. However, for efficiency in the use of the computer, he would prefer that the computation of volume be performed only once, before the simulation run, rather than repeatedly during the run.

The INITIAL and DYNAMIC translator control cards provide this capability. These cards are used to bound a block of nonprocedural statements that are to be performed only at the beginning of the run. The DYNAMIC card signifies that the portions of the problem statement that follow it represent the dynamic portion of the simulation rather than the initialization. If the INITIAL translator control card is not used, the program assumes that all statements describe the dynamics of the simulation; hence, the DYNAMIC card is not required in the absence of the INITIAL card.

Use of this feature is illustrated under "sample problem" both in this manual and in System/360 Continuous System Modeling Program: Application Description (H20-0240).

TERMINATION

It is often desirable to perform a set of computations only at the completion of a simulation run. For example, the user may wish to totalize the results or to process them in accord with his own print or plot routine. Also, in many problems, it is desirable to use the results obtained in a run to control subsequent runs, possibly to decide whether or not these runs should be made. Examples are parameter optimization and two-point boundary value problems that require modification of parameters, between iterative simulations, based on the results obtained in prior runs. A criterion is required in these cases for terminating this iterative sequence when a "sufficiently accurate" solution has been obtained.

The TERMINAL control card allows the user to efficiently and conveniently perform this type of computation or achieve this type of program control. All of the structure statements that follow a TERMINAL card will be performed only at the completion of the run--that is, only when a FINISH condition or FINTIM has been reached. Note that TERMINAL allows both decisions and prescribed parameter modifications to be made after a run, whereas the END control card can be used only for modifications prescribed before the run. Care should be exercised when the TERMINAL feature is used in combination with run or initialization options to ensure that modifications are actually made in the order desired. It must be recognized that until the conditions imposed in the TERMINAL section are satisfied, the program will return from TERMINAL to the structure statements in the initialization section, if one is included. In any case, the specified TERMINAL operations will always be completed before the implementation of run options given by multiple PARAMETER, CONTINUE, or END cards. Computations performed in the TERMINAL section override any specifications on data statements anywhere in the run, but do not override structure statements.

An illustration of the use of TERMINAL is given by the following example:

```
*    INITIAL CONDITIONS
PARAMETER TTOP=2.0,X0=20.0,WTFCT=100.0
PARAMETER TEMP=1000.0
*    PROBLEM EQUATIONS
                .
                .
                .
            TZRO= . . .
            TGPR= . . .
                .
                .
                .
*    THESE STATEMENTS FORM THE CONVERGENCE CRITERION
TERMINAL
          ERROR=TZRO-TGPR
          IF(ABS(ERROR)- 0.5) 50,50,40
    40    TTOP=X0+(TTOP-X0)*WTFCT/(WTFCT-ERROR)
          CALL RERUN
          GO TO 60
    50    H=TZRO**2*TEMP**4
          WRITE(6,51)H,TZRO
    51    FORMAT(10X,F8.2,10X,F8.2)
    60    CONTINUE
END
```

In this example, the objective is to converge on an optimum value of the parameter TTOP, starting with an initial guess of 2.0. After convergence, the program is to calculate an output variable H based on the optimum TTOP, and to output results in accordance with the user's own format and routine.

The statements following the identifying TERMINAL control card form the convergence criterion, which is a comparison between the absolute value of the variable ERROR and 0.5. As long as ABS(ERROR) is greater than 0.5, the iteration continues with a new estimate of TTOP computed as indicated in statement 40. Note that the user must indicate his desire to rerun the simulation with the new value of TTOP, by explicitly using the statement CALL RERUN. This is a specific S/360 CSMP statement that should be used only in connection with TERMINAL.

The GO TO and CONTINUE cards are a convenient method for bypassing the intervening computations. The CONTINUE card may be considered to establish a collection point for the preceding FORTRAN logical statements.

When ABS(ERROR) is either equal to or less than 0.5, iteration is terminated. The program then computes H and prints out H and TZRO in the format specified in statement 51. The user should, of course, always provide an optional termination procedure that can be automatically invoked if his criterion fails to provide convergence after some specified number of iterations.

The control card END signals the completion of this particular sequence of parameter optimization runs. Additional statements could be entered following this card with new parameters--for example, a different value of TEMP to initiate another optimization sequence.

Another illustration of the use of TERMINAL is found later under "Sample Problem".

INTEGRATION

Selection of an integration method and integration interval for a particular simulation study should not be made casually, but only after considering a number of interrelated factors. The objective is to choose the combination of routine and interval that will provide the fastest execution, while maintaining enough accuracy for the purposes of the simulation study and obtaining enough output data for easy interpretation of results.

In general, as the complexity of the integration method increases, the computer time required for a single step also increases. On the other hand, stability in numeric method may also increase, permitting larger time steps. Thus, a number of tradeoffs between accuracy and running time are possible.

Basically two types of integration method are available in S/360 CSMP: variable-step and fixed-step. The mathematical formulas for these techniques are given later under "Methods". The former provide automatic adjustment of integration interval consistent with the user's specifications on the ABSERR or RELERR execution control cards. An estimate of error is made in these methods by computing $Y(t+\Delta t)$ by two different formulas having complementary error terms, and applying the results in an estimation equation. The advantage of these methods is that they achieve the specified accuracy with the maximum possible step size.

Two variable-step routines are provided: MILNE and Runge-Kutta (RKS). Both are sophisticated methods involving somewhat lengthy computation. They have the advantage of virtually ensuring a satisfactory solution--but possibly at the cost of excessive running time, if the error criterion has been set too stringently. Note that the error criteria can be set independently for each integrator in the simulation, thereby allowing the severe constraints to be set only as necessary.

Within the set of fixed-step routines, these options, in order of decreasing complexity, are Runge-Kutta (RKSFX), Simpson's (SIMP), trapezoidal (TRAPZ), ADAMS, and rectangular (RECT). For a given step size, speed of solution would be rated in the reverse order. Generally, however, for reasons of stability, the less sophisticated methods require shorter integration intervals to achieve comparable accuracy.

Selection from among these several options is simplified by first considering what output is required from the runs. Are both tabular and plotted output required? At what time interval is tabular output required? What should be the overall duration of a simulation run? Clearly, one should, in any case, first specify FINTIM, PRDEL, and OUTDEL. Note that the program is so designed that the larger of OUTDEL and PRDEL will be satisfied exactly. The lesser will be automatically adjusted by the program to be a commensurable submultiple; FINTIM will likewise be automatically adjusted to be a multiple of the lesser output interval after adjustment.

Note too that the smaller output interval is automatically treated as the maximum possible integration interval in a variable-step method. Accordingly, if either tabular or plotted output requires an interval very small compared with FINTIM, it is possible that no significant advantage will be realized by selection of an integration method using variable step-size adjustment. If the maximum step size is constrained by the relatively small output interval, use of the variable-step integration methods may increase solution time without significantly improving accuracy. In such a case, the less sophisticated integration methods should be considered.

If no integration method is specified, the program automatically utilizes the RKS method. This method is generally a good choice for the initial runs of a simulation study, if the user is unsure of the dynamic response of the simulated phenomenon. It is advisable to make the initial choice of DELT sufficiently small to ensure accurate, stable solution even at the expense of a longer-than-necessary initial computer run. After obtaining greater familiarity with a particular problem, the user may choose a more optimal combination of timing specifications and integration method.

Some additional considerations apply to simulations involving discontinuous functions, such as MODINT, STEP, RAMP, and ZHOLD, or elements involving critical time relationships, such as PULSE and IMPULS. In such problems, the simpler integration methods using short integration steps may give better results. For example, to accurately represent the pulse train generated by the IMPULS element, a fixed-step method should be used with an integration step that is a submultiple of the desired pulse interval.

Above all, the user should be prepared to do some experimentation to achieve an optimal solution in terms of running time and accuracy.

ARBITRARY FUNCTIONS

For handling functions of one variable, S/360 CSMP provides two functional blocks: AFGEN (arbitrary function generator) and NLFGEN (nonlinear function generator).

A particular function is defined and used by means of corresponding data and structure statements. The x, y coordinates of the function points are entered sequentially in a data statement, following the function label and the symbolic name of the function. For example:

```
FUNCTION CURVE1=-10.2,2.3,-5.6,6.4,1.0,5.9,etc.
```

defines a function called CURVE1, where the data points on the curve are given in pairs, x and y. The first point is (-10.2, 2.3); the second, (-5.6, 6.4); the third, (1.0, 5.9); and so on. The corresponding S/360 CSMP structure statement would be:

```
   Y3=AFGEN(CURVE1,XIN)
```

Although the total number of data storage locations is necessarily fixed by machine size, there is no specific restriction on the number of points one may use to define any particular function. The only requirement is that the x coordinates in the sequence x_1, y_1, x_2, y_2 . . . be monotonically increasing. Any number of arbitrary functions may be defined, identified only by the symbolic names assigned by the user.

AFGEN provides linear interpolation between consecutive points, while NLFGEN uses a Lagrange quadratic interpolation. Caution should be exercised in the use of the NLFGEN functional element. In particular, if the desired function contains an abrupt discontinuity, it must be recognized that a quadratic interpolation formula cannot represent such a discontinuity without distortion. For such situations it may be preferable to use the AFGEN element, even though it might, at first glance, appear to offer the less precise representation of nonlinear functions in general.

IMPLICIT FUNCTIONS

The library includes an implicit function for the solution of algebraic loops defined by algebraic equations containing no memory function. This feature, in effect, directs the system to perform a subiteration within the implicit loop, at each instant of time, until the algebraic relationship has been satisfied. A standard convergence formula is provided for which the user can specify the error criterion. However, should this formula not prove satisfactory for a particular problem, the user can program his own convergence routine. A description of how this is done is given in the System Manual. In the standard function, if there is no convergence after 100 iterations, the run terminates and a diagnostic message is provided.

An important point to note in connection with the implicit function is the difference between memory and history functions. A memory function is one in which the output depends only on past values of the input and output. A history function is one in which the output depends on the present value of the input as well as past input and output values. Accordingly, the output of a memory function can be obtained without knowing the current input; the output of a history function cannot. A memory function, therefore, breaks a loop in which it is contained, but a history function does not. It follows that a closed loop containing a history element, but no memory element, is an implicit relationship--and requires an IMPL functional block to break the loop. The standard functions of S/360 CSMP should be treated as follows:

Memory Functions

INTGRL

DELAY

History Functions

DERIV	RST
HSTRSS	IMPULS
ZHOLD	PULSE

Should the user not recognize that the problem involves an algebraic loop, a diagnostic message will be provided which indicates failure of the sorting algorithm and lists the variables in the loop. To use the implicit function, the user starts by writing the following type of S/360 CSMP structure statement:

Y=IMPL(IC, P, FOFY)

where Y is the variable whose convergence is being tested, IMPL is the name of the implicit functional block, IC is the initial guess of the variable Y provided by the user, P is the error criterion that is to be met, and FOFY is the output name of the last statement in the definition of the algebraic loop.

This statement must be followed immediately by a logical set of S/360 CSMP and/or FORTRAN statements evaluating FOFY. S/360 CSMP sets up the necessary iterative loop, which includes the proper sequencing for information flow.

This procedure is illustrated for the implicit equation:

$Z=f(Z)=Ae^{-t}+B\sin(Z)$

where A and B are given parameters. Since the first term does not include Z, it need not be included within the iterative loop.

The S/360 CSMP statements might be as follows:

```
  C1 = A * EXP( -TIME )

   Z = IMPL( Z0, ERROR, FOFZ )

  C2 = B * SIN( Z )

FOFZ = C1 + C2
```

Note that the statements defining a function f(Z) must meet the following requirements.

1. As many statements as necessary may be used to define f(Z), but the output name in the last statement of the definition must be identical to the third argument of the IMPL statement. There is one restriction: this output name cannot be the output of a MACRO or PROCEDURE function.
2. The implicit variable (in this example Z) must appear on the right side of an equal sign at least once in the definition statements.
3. An implicit loop can be defined within a MACRO or PROCEDURE, provided that the entire set of required statements is contained within the MACRO or PROCEDURE.
4. An implicit loop cannot be defined within another implicit loop.

For the example above, the S/360 CSMP translator automatically generates the following statements:

```
30001 Z=IMPL(Z0,ERROR,FOFZ)

      IF(NALARM)30003,30003,30002

30002 CONTINUE

      C2=B*SIN(Z)

      FOFZ=C1+C2

      GO TO 30001

30003 CONTINUE
```

Only four statements are added to those written by the user. The statement numbers are assigned by the program.

The first time the IMPL routine is entered, NALARM is set to one, and Z is given the user's initial guess Z0. After each calculation of f(Z), the program flow returns to the IMPL subroutine where the convergence criterion is tested. If the criterion is satisfied, NALARM is set equal to zero and Z assumes the most recently calculated value of FOFZ. If the convergence criterion is not satisfied, the iteration continues. If convergence fails after 100 iterations, NALARM is set negative and the simulation run is terminated with an appropriate comment. Convergence is indicated if:

$$\left|\frac{Z_{n+1}-Z_n}{Z_{n+1}}\right| \le \text{ERROR}, \qquad |Z_{n+1}| > 1$$

$$|Z_{n+1}-Z_n| \le \text{ERROR}, \qquad |Z_{n+1}| \le 1$$

The implicit function feature of S/360 CSMP is specifically designed to solve relationships of the form Z=f(Z). Thus, the IMPL statement, together with the entire set of statements required to define f(Z), may be viewed as a single "functional block" with variable Z as its output. To avoid possible erroneous results, special precautions must be observed when the feature is used in any other manner.

One such instance would be the use, in some other portion of the model, of an intermediate variable computed within an implicit loop. For the example above, suppose that the quantity B sin(Z) were of use outside the implicit loop -- in addition to Z, the nominal output of the implicit relationship. Program errors may result if the user should attempt to use the variable C2 in portions of the model outside the loop. If a term computed within a loop is required elsewhere, it is simplest to recompute the term where needed. Alternatively, for some models it may be possible to place the entire implicit loop within a multiple-output PROCEDURE thus:

```
PROCEDURE  C2, Z = LOOP( Z0, ERROR, B, C1 )
                Z = IMPL( Z0, ERROR, FOFZ )
               C2 = B * SIN( Z )
             FOFZ = C1 + C2
ENDPRO
```

Note that the PROCEDURE statement must have listed as arguments of the function all variables and parameters used within the PROCEDURE. For this example parameters Z0, ERROR, and B and variable C1 are thereby listed as arguments of function LOOP. Use of the implicit function within a PROCEDURE does require that the user properly sequence the statements within the loop.

TABULAR DATA

This feature of S/360 CSMP allows subscripted variables and blocks of data to be handled conveniently. For example, it permits data stored in a one-dimensional array to be identified in structure statements by the location of the data in that array. Also, in the construction of a special function, the user may have to transfer, to a subroutine, large amounts of data such as input parameters, initial conditions, matrix elements, or history. With this feature the need for a lengthy subroutine argument string is eliminated.

Use of the Tabular Data feature is illustrated by the following set of statements:

```
STORAGE  IC(5), PAR(3), MATRIX(6)

TABLE    IC(1)=25.1, IC(2)=4.0, IC(3-5)=3*41.1, . .

         PAR(1-3)=3*4.1, MATRIX(1)=2.9E4, . . .

         MATRIX(2-6)=5*31.7

Z=PAR(1)+LIMIT(IC(2), IC(3), XIN)

Y=FUNCT(IC, MATRIX, MATRIX(3), X)
```

The first statement, STORAGE, instructs the program to assign a total of 14 locations: five for the array IC, three for PAR, and six for MATRIX. The second statement, TABLE, enters the indicated numeric values into the specified locations. The third statement shows an elementary use of the ability to reference individual items in the table within structure statements. The fourth statement uses the stored variables in a user-defined subprogram that references the entire IC and MATRIX arrays, as well as the individual element MATRIX(3).

An additional illustration of this feature is shown in the following statements:

```
FIXED          L, IBAND
STORAGE        IBAND(12)
TABLE          IBAND(1-12) = 12*0

         G = DEVICE( PAR1, XIN )
       XIN = GAUSS( 3, 0.0, 2.0 )


PROCEDURE      L = BAND( G )
               L = G + 7.0
               L = MAX0( 1, L )
               L = MIN0( 12,L )
     IBAND(L)      IBAND(L) + 1
ENDPRO
```

The purpose of these statements is to prepare a histogram showing the distribution of the variable G within a range of -6.0 to +6.0. G is obtained, according to the problem definition, as the output of a DEVICE with parameter PAR1, and input from the Gaussian noise generator. The STORAGE statement sets up the necessary locations for the bands of the histogram. The TABLE statement simply initializes the contents of those locations. A PROCEDURE function takes the generated values of G and produces the desired histogram. The results can be displayed using FORTRAN output statements in a TERMINAL section.

Note that the following rules must be observed when using the Tabular Data feature:

1. STORAGE card(s) must be placed before any reference to variable names on the card.
2. Dimensioned variables cannot appear to the left of an equal sign, except, as illustrated above, within a PROCEDURE or NOSORT function.
3. MACRO and PROCEDURE arguments cannot be dimensioned.

The S/360 CSMP "TABLE" form of data entry, used in conjunction with a STORAGE declaration statement, is applicable only for one-dimensional arrays. This combination permits convenient modification of tabular data from run to run. Alternate methods using FORTRAN statements are available, however, and must be used for multidimensional arrays. The dimensions of a multidimensional array can be specified using the FORTRAN "DIMENSION" statement.

One means of initializing such an array is by the FORTRAN "DATA" statement. With this method, however, the data stored in the array cannot be conveniently replaced by a new set of data for a subsequent run. Thus, this method is often used where the parameters remain unchanged for the entire sequence of runs. For example, one might specify an array X and initialize its elements as follows:

```
/    DIMENSION  X(2,5)
/    DATA   X/3*0.0,27.4,34.4,0.0,4*5.0/
```

Note that whenever DIMENSION or DATA statements are used with S/360 CSMP, it is necessary that a virgule (/) be used in column 1.

In rare instances it may be desirable to specify an array by means of LABELED COMMON so that the data are available to a user-supplied subprogram. In such cases, the data must be initialized using the BLOCK DATA feature of FORTRAN. This method is not recommended for the novice.

Another method of initializing an array is by means of the FORTRAN "READ (5, xxx)" statement, where xxx is the statement number of the corresponding FORMAT statement. The data cards to be read must be placed between the labels DATA and ENDDATA immediately following the END statement for the corresponding run. As example, the following statements would be equivalent to the preceding example:

```
/     DIMENSION   X(2,5)
      FIXED I,J
INITIAL
 NOSORT
      READ (5, 100 ) ( (X(I,J), J = 1, 5), I = 1,2 )
      100 FORMAT ( 5F8.3)
      .
      .
END
DATA
   0.0    0.0    34.4    5.0    5.0
   0.0   27.4     0.0    5.0    5.0
ENDDATA
      .
      .
TITLE   SECOND RUN WITH NEW DATA FOR X  ARRAY
END
DATA
   1.0    1.0    34.4    7.0    7.8
   0.105  8.37   12.06   7.8    8.0
ENDDATA
STOP
ENDJOB
```

Note that for the second run the second set of data cards would be read assigning new parameter values for the array X. Note also that the second DATA label immediately follows the second END statement. In this manner, a series of runs may be specified and, if desired, new data may be read for each run. By use of logical control statements in conjunction with the READ statement, it is possible to read data for selected runs only. The user must, however, take care that the proper number of data cards are available for each run, and that the format of the data agrees with the FORMAT specification. All continuation cards for FORMAT statements must have a $ in card column 6 and must immediately follow the cards they continue.

RUN CONTROL

SEQUENTIAL RUNS

Several means are available for obtaining an automatic sequence of runs of the same structure statements with different data and control statements. These permit easy variation of parameters, output variables, and output options between runs.

Figure 11 illustrates the use of the CONTINUE and END execution control statements for this purpose. When a run is completed, these operate as follows:

1. A CONTINUE card permits the simulation to accept changed data and control statements, without resetting of TIME, and continues the run. For example, this feature permits such items as integration method, integration interval, and data output interval to be modified during the progress of a simulation. In the illustration, the run halts at TIME = 10.0 and then continues with a new printout interval of 1.0 in place of the previous interval of 0.1. Note that the run will continue to the point at which TIME is equal to 15.0.
2. An END card permits the simulation to accept changed data and control statements, resets TIME, and initiates a new run. In this illustration, the parameter P1 is equal to 5.0 in the first run, and 9.0 in the second run. Note that, unless additional specifications are made, FINTIM will be 15.0 and PRDEL will be 1.0 for the entire second run.
3. A combination of END and STOP cards indicates termination of the sequence.

```
PARAMETER      P1=5.0
   .
   .           Structure, data, and control statements defining the run
   .
TIMER          FINTIM=10.0,PRDEL=0.1

CONTINUE

TIMER          FINTIM=15.0,PRDEL=1.0
   .           Data statements changing certain parameters
   .
   .

END

PARAMETER      P1=9.0
   .           Other data statements changing parameters
   .
   .

END

STOP
```

Figure 11. Illustration of sequential runs

As explained earlier under "Data Statements", changes in either a parameter, initial condition, or constant can also be performed automatically by specifying multiple values on an appropriate data card. For example, the statement:

PARAMETER RATE=(5.0,5.5,6.0)

will initiate a sequence of three runs in which RATE = 5.0 for the first run, RATE = 5.5 for the second run, and RATE = 6.0 for the third run. Only one parameter can use the multiple parameter option at a time. The multiple value parameter option is not intended for use with the CONTINUE option. A multiple value parameter is limited so that the maximum number of runs in any sequence may not exceed fifty. See also section on data statements for use of PARAMETER statement.

If the structural statements are to be changed, multiple jobs must be submitted. Each must be provided either with an ENDJOB STACK card followed by a blank card, as the last cards in the set, or its own OS/360 control cards. This is explained in the Operator's Manual.

Note that if one job in a stack is terminated by the Operating System, the remaining S/360 CSMP jobs in the stack will be bypassed until the next set of OS/360 control cards is read. If a job is terminated by S/360 CSMP, the other jobs will be processed.

MAIN PROGRAM CONTROL

The entire S/360 CSMP simulation can be put under control of a conventional FORTRAN program. This can be done by changing a small program called MAIN, which controls the execution phase of S/360 CSMP. MAIN consists of a series of calls that initialize, scan the input data, and perform the simulation. At the completion of a simulation run, control is returned to the MAIN program and a test is made to see whether there are any more data cards describing parameter changes for another run. Virtually all of the capabilities available by using MAIN are more readily available by using the TERMINAL segment discussed previously. However, MAIN does permit flexible use of the S/360 CSMP system modules, including substitution of user-supplied modules, which TERMINAL does not. Application of this feature, however, requires familiarity with the details of FORTRAN, since the MAIN program is a FORTRAN routine.

The procedure for changing the MAIN program is quite involved; an explanation of it is therefore deferred to the System Manual.

DATA OUTPUT

The output options available are the printing of variables, print-plotting of variables, output of minimum and maximum values of specified variables, and preparation of a data set for user-prepared plotting programs. A title can be specified as a page heading to printed output and a label can be put on the print plots. In addition to these standard options, FORTRAN output capabilities are available to the user through use of a NOSORT section, PROCEDURE function, or TERMINAL segment.

There are two print formats: one in column form with the variable names at the top of the columns, and one in equation form. The column form is used if eight or fewer dependent variable names are requested; the equation form is used if from 9 to 49 dependent variable names are requested. A maximum of 49 dependent variables can be printed per simulation run. The independent variable (TIME) is automatically printed. Up to five lines in a TITLE will be printed at the top of each page of printed output. In addition, the integration method and the specific value of the parameter (if there is a sequence of multiple parameter runs) will be shown on the first line of the title. Printout occurs at the specified or adjusted PRDEL interval, as explained earlier under "Output Control Statements". As also explained in that section, the PRINT statement and all other data output statements can be used only with real variables. Figure 12 illustrates the column format; Figure 13 illustrates the equation format.

```
                CABLE REEL CONTROL DESIGN                                        RECT      INTEGRATION      GAIN  =  1.0000E 00
 TIME          VACT          VM            ERROR         CONTL         TORQUE        R             I
0.0           0.0           0.0           5.0000E 01    5.0000E 01    0.0           4.0000E 00    4.5150E 03
5.0000E-01    2.1856E 00    5.1229E-01    4.9488E 01    4.9488E 01    1.0009E 04    3.9999E 00    4.5147E 03
1.0000E 00    7.8819E 00    3.4462E 00    4.6554E 01    4.6554E 01    1.5679E 04    3.9995E 00    4.5125E 03
1.5000E 00    1.5435E 01    8.9041E 00    4.1096E 01    4.1096E 01    1.8225E 04    3.9984E 00    4.5072E 03
2.0000E 00    2.3635E 01    1.6009E 01    3.3991E 01    3.3991E 01    1.8475E 04    3.9965E 00    4.4982E 03
2.5000E 00    3.1606E 01    2.3787E 01    2.6213E 01    2.6213E 01    1.7117E 04    3.9937E 00    4.4854E 03
3.0000E 00    3.8755E 01    3.1430E 01    1.8570E 01    1.8570E 01    1.4745E 04    3.9902E 00    4.4690E 03
3.5000E 00    4.4733E 01    3.8357E 01    1.1643E 01    1.1643E 01    1.1854E 04    3.9861E 00    4.4495E 03
4.0000E 00    4.9385E 01    4.4207E 01    5.7933E 00    5.7933E 00    8.8357E 03    3.9814E 00    4.4276E 03
4.5000E 00    5.2709E 01    4.8810E 01    1.1903E 00    1.1903E 00    5.9764E 03    3.9763E 00    4.4038E 03
5.0000E 00    5.4812E 01    5.2146E 01   -2.1458E 00   -2.1458E 00    3.4668E 03    3.9709E 00    4.3788E 03
5.5000E 00    5.5871E 01    5.4304E 01   -4.3037E 00   -4.3037E 00    1.4134E 03    3.9654E 00    4.3531E 03
6.0000E 00    5.6097E 01    5.5443E 01   -5.4433E 00   -5.4433E 00   -1.4530E 02    3.9597E 00    4.3271E 03
6.5000E 00    5.5711E 01    5.5762E 01   -5.7624E 00   -5.7624E 00   -1.2227E 03    3.9541E 00    4.3013E 03
7.0000E 00    5.4921E 01    5.5470E 01   -5.4703E 00   -5.4703E 00   -1.8679E 03    3.9485E 00    4.2759E 03
7.5000E 00    5.3909E 01    5.4767E 01   -4.7673E 00   -4.7673E 00   -2.1518E 03    3.9430E 00    4.2509E 03
8.0000E 00    5.2826E 01    5.3831E 01   -3.8313E 00   -3.8313E 00   -2.1550E 03    3.9376E 00    4.2264E 03
8.5000E 00    5.1785E 01    5.2809E 01   -2.8089E 00   -2.8089E 00   -1.9584E 03    3.9323E 00    4.2025E 03
9.0000E 00    5.0864E 01    5.1813E 01   -1.8127E 00   -1.8127E 00   -1.6365E 03    3.9271E 00    4.1792E 03
9.5000E 00    5.0108E 01    5.0921E 01   -9.2104E-01   -9.2104E-01   -1.2532E 03    3.9220E 00    4.1562E 03
1.0000E 01    4.9535E 01    5.0181E 01   -1.8129E-01   -1.8129E-01   -8.5928E 02    3.9169E 00    4.1337E 03
1.0500E 01    4.9144E 01    4.9614E 01    3.8560E-01    3.8560E-01   -4.9190E 02    3.9119E 00    4.1114E 03
1.1000E 01    4.8917E 01    4.9221E 01    7.7939E-01    7.7939E-01   -1.7543E 02    3.9069E 00    4.0893E 03
1.1500E 01    4.8828E 01    4.8985E 01    1.0148E 00    1.0148E 00    7.7063E 01    3.9019E 00    4.0674E 03
1.2000E 01    4.8845E 01    4.8884E 01    1.1161E 00    1.1161E 00    2.6177E 02    3.8969E 00    4.0455E 03
1.2500E 01    4.8937E 01    4.8888E 01    1.1125E 00    1.1125E 00    3.8182E 02    3.8920E 00    4.0236E 03
1.3000E 01    4.9075E 01    4.8966E 01    1.0338E 00    1.0338E 00    4.4505E 02    3.8869E 00    4.0018E 03
1.3500E 01    4.9233E 01    4.9092E 01    9.0831E-01    9.0831E-01    4.6206E 02    3.8819E 00    3.9800E 03
1.4000E 01    4.9392E 01    4.9240E 01    7.6045E-01    7.6045E-01    4.4448E 02    3.8768E 00    3.9581E 03
1.4500E 01    4.9537E 01    4.9390E 01    6.0970E-01    6.0970E-01    4.0372E 02    3.8718E 00    3.9362E 03
1.5000E 01    4.9659E 01    4.9530E 01    4.7041E-01    4.7041E-01    3.5001E 02    3.8666E 00    3.9143E 03
1.5500E 01    4.9753E 01    4.9648E 01    3.5193E-01    3.5193E-01    2.9184E 02    3.8615E 00    3.8925E 03
1.6000E 01    4.9819E 01    4.9741E 01    2.5908E-01    2.5908E-01    2.3572E 02    3.8564E 00    3.8706E 03
1.6500E 01    4.9859E 01    4.9807E 01    1.9312E-01    1.9312E-01    1.8615E 02    3.8512E 00    3.8488E 03
1.7000E 01    4.9875E 01    4.9847E 01    1.5251E-01    1.5251E-01    1.4579E 02    3.8461E 00    3.8271E 03
1.7500E 01    4.9875E 01    4.9866E 01    1.3382E-01    1.3382E-01    1.1571E 02    3.8409E 00    3.8053E 03
1.8000E 01    4.9861E 01    4.9867E 01    1.3263E-01    1.3263E-01    9.5755E 01    3.8357E 00    3.7837E 03
1.8500E 01    4.9839E 01    4.9856E 01    1.4413E-01    1.4413E-01    8.4903E 01    3.8306E 00    3.7621E 03
1.9000E 01    4.9814E 01    4.9836E 01    1.6368E-01    1.6368E-01    8.1583E 01    3.8254E 00    3.7406E 03
1.9500E 01    4.9789E 01    4.9813E 01    1.8719E-01    1.8719E-01    8.3973E 01    3.8202E 00    3.7191E 03
2.0000E 01    4.9765E 01    4.9789E 01    2.1126E-01    2.1126E-01    9.0224E 01    3.8150E 00    3.6978E 03
```

Figure 12. Illustration of column format

```
   S/360 CONTINUOUS SYSTEM MODELING PROGRAM  --  OUTPUT                  MILNE     INTEGRATION

 TIME  =  0.0              O2CAP =  5.2000E-01     O2TIS =  2.1000E 00      PO4    =  5.6000E-03     CAP    =  1.0000E 00
                           R     =  1.1000E-01     FLOW  =  5.2707E-08      O2CCAP=  1.3000E 02      PO2CAP=  3.5960E 01
                           FLOWAV=  0.0            PO2TAV=  0.0

 TIME  =  2.5000E-01       O2CAP =  5.8335E-01     O2TIS =  2.3911E 00      PO4    = -7.9163E-06     CAP    =  1.0000E 00
                           R     =  4.6620E 00     FLOW  =  1.7006E-01      O2CCAP=  1.4534E 02      PO2CAP=  4.1079E 01
                           FLOWAV=  1.8408E-02     PO2TAV=  8.3480E 00

 TIME  =  5.0000E-01       O2CAP =  6.1200E-01     O2TIS =  2.6458E 00      PO4    =  3.9844E-04     CAP    =  1.0000E 00
                           R     =  4.6618E-01     FLOW  =  1.7003E-05      O2CCAP=  1.5300E 02      PO2CAP=  4.4249E 01
                           FLOWAV=  4.5970E-02     PO2TAV=  1.9618E 01

 TIME  =  7.5000E-01       O2CAP =  6.2057E-01     O2TIS =  2.7244E 00      PO4    =  4.7665E-04     CAP    =  1.0000E 00
                           R     =  4.0916E-01     FLOW  =  1.0089E-05      O2CCAP=  1.5514E 02      PO2CAP=  4.5554E 01
                           FLOWAV=  6.9205E-02     PO2TAV=  3.0883E 01

 TIME  =  1.0000E 00       O2CAP =  6.2887E-01     O2TIS =  2.8040E 00      PO4    =  4.4064E-04     CAP    =  1.0000E 00
                           R     =  9.6447E-01     FLOW  =  3.1150E-04      O2CCAP=  1.5722E 02      PO2CAP=  4.6867E 01
                           FLOWAV=  9.2477E-02     PO2TAV=  4.2119E 01

 TIME  =  1.2500E 00       O2CAP =  6.3644E-01     O2TIS =  2.8484E 00      PO4    =  2.7735E-04     CAP    =  1.0000E 00
                           R     =  3.9941E 00     FLOW  =  9.1613E-02      O2CCAP=  1.5911E 02      PO2CAP=  4.8104E 01
                           FLOWAV=  1.1469E-01     PO2TAV=  5.3313E 01

 TIME  =  1.5000E 00       O2CAP =  6.3608E-01     O2TIS =  2.8321E 00      PO4    =  1.1216E-04     CAP    =  1.0000E 00
                           R     =  4.4142E 00     FLOW  =  1.3668E-01      O2CCAP=  1.5902E 02      PO2CAP=  4.8045E 01
                           FLOWAV=  1.3497E-01     PO2TAV=  6.4513E 01

 TIME  =  1.7500E 00       O2CAP =  6.3063E-01     O2TIS =  2.7694E 00      PO4    =  8.3399E-06     CAP    =  1.0000E 00
                           R     =  4.6508E 00     FLOW  =  1.6842E-01      O2CCAP=  1.5766E 02      PO2CAP=  4.7152E 01
                           FLOWAV=  1.5384E-01     PO2TAV=  7.5725E 01

 TIME  =  2.0000E 00       O2CAP =  6.2324E-01     O2TIS =  2.6949E 00      PO4    = -9.8489E-06     CAP    =  1.0000E 00
                           R     =  4.6620E 00     FLOW  =  1.7006E-01      O2CCAP=  1.5581E 02      PO2CAP=  4.5972E 01
                           FLOWAV=  1.7253E-01     PO2TAV=  8.6972E 01

 TIME  =  2.2500E 00       O2CAP =  6.1381E-01     O2TIS =  2.6046E 00      PO4    = -5.1169E-06     CAP    =  1.0000E 00
                           R     =  4.6620E 00     FLOW  =  1.7006E-01      O2CCAP=  1.5345E 02      PO2CAP=  4.4520E 01
                           FLOWAV=  1.9102E-01     PO2TAV=  9.8256E 01

 TIME  =  2.5000E 00       O2CAP =  6.0328E-01     O2TIS =  2.5151E 00      PO4    = -3.9807E-06     CAP    =  1.0000E 00
                           R     =  4.5688E 00     FLOW  =  1.5685E-01      O2CCAP=  1.5082E 02      PO2CAP=  4.2968E 01
                           FLOWAV=  2.0985E-01     PO2TAV=  1.0962E 02

 TIME  =  2.7500E 00       O2CAP =  6.0142E-01     O2TIS =  2.5534E 00      PO4    =  1.3468E-04     CAP    =  1.0000E 00
                           R     =  1.0905E 00     FLOW  =  5.0905E-04      O2CCAP=  1.5035E 02      PO2CAP=  4.2701E 01
                           FLOWAV=  2.3192E-01     PO2TAV=  1.2096E 02

 TIME  =  3.0000E 00       O2CAP =  6.1016E-01     O2TIS =  2.6297E 00      PO4    =  3.6808E-04     CAP    =  1.0000E 00
                           R     =  5.0034E-01     FLOW  =  2.2561E-05      O2CCAP=  1.5254E 02      PO2CAP=  4.3974E 01
                           FLOWAV=  2.5520E-01     PO2TAV=  1.3232E 02

 TIME  =  3.2500E 00       O2CAP =  6.1890E-01     O2TIS =  2.7150E 00      PO4    =  4.7169E-04     CAP    =  1.0000E 00
                           R     =  4.0914E-01     FLOW  =  1.0088E-05      O2CCAP=  1.5472E 02      PO2CAP=  4.5296E 01
                           FLOWAV=  2.7843E-01     PO2TAV=  1.4350E 02
```

Figure 13. Illustration of equation format

For the print-plot output, the dependent variable is print-plotted across the page and the independent variable (TIME) down the page. In addition, the values of time and the plotted variable are printed to the left side of the print plot. The minimum and maximum values of the dependent variable are listed at the top of the page along with a supplementary title of the form "Name of Variable Plotted" versus "Independent Variable". In addition, the specific value of the parameter, in a sequence of multiple parameter runs, is shown. Up to three other variables can be printed to the right side of the plot. One-line labels can be printed on each print plot. PRTPLT occurs at the specified or adjusted OUTDEL, as explained under "Output Control Statements". Figure 14 illustrates this type of output.

Minimum and maximum values of variables can be obtained by use of the RANGE control card, and are also automatically provided if the variable is specified for the PRTPLT or PREPAR output options. Output from use of the RANGE card is in five columns. The first column gives the variable name; the second gives its minimum value; the third indicates the time when the minimum value was reached; the fourth gives the variable's maximum value; and the fifth gives the time when the maximum value was reached. The TIME interval over which the RANGE values were obtained is shown at the top of the RANGE listing. Figure 15 illustrates this output option.

```
   CABLE REEL CONTROL DESIGN  --  TEST OF SYSTEM STABILITY          PAGE   1

                         MINIMUM              VACT   VERSUS TIME            MAXIMUM
                        0.0                   GAIN  = 1.0000E 00           6.3492E 01
 TIME            VACT          I                                              I       VM            ERROR        R
0.0             0.0            +                                                      0.0           5.0000E 01   4.0000E 00
5.0000E-01      2.1856E 00     -+                                                     5.1229E-01    4.9488E 01   3.9999E 00
1.0000E 00      7.8819E 00     ------+                                                3.4462E 00    4.6554E 01   3.9995E 00
1.5000E 00      1.5435E 01     ------------+                                          8.9041E 00    4.1096E 01   3.9984E 00
2.0000E 00      2.3635E 01     ------------------+                                    1.6009E 01    3.3991E 01   3.9965E 00
2.5000E 00      3.1606E 01     ------------------------+                              2.3787E 01    2.6213E 01   3.9937E 00
3.0000E 00      3.8755E 01     -----------------------------+                         3.1430E 01    1.8570E 01   3.9902E 00
3.5000E 00      4.4733E 01     ----------------------------------+                    3.8357E 01    1.1643E 01   3.9861E 00
4.0000E 00      4.9385E 01     -------------------------------------+                 4.4207E 01    5.7933E 00   3.9814E 00
4.5000E 00      5.2709E 01     ----------------------------------------+              4.8810E 01    1.1903E 00   3.9763E 00
5.0000E 00      5.4812E 01     ------------------------------------------+            5.2146E 01   -2.1458E 00   3.9709E 00
5.5000E 00      5.5871E 01     ------------------------------------------+            5.4304E 01   -4.3037E 00   3.9654E 00
6.0000E 00      5.6097E 01     -------------------------------------------+           5.5443E 01   -5.4433E 00   3.9597E 00
6.5000E 00      5.5711E 01     ------------------------------------------+            5.5762E 01   -5.7624E 00   3.9541E 00
7.0000E 00      5.4921E 01     ------------------------------------------+            5.5470E 01   -5.4703E 00   3.9485E 00
7.5000E 00      5.3909E 01     -----------------------------------------+             5.4767E 01   -4.7673E 00   3.9430E 00
8.0000E 00      5.2826E 01     ----------------------------------------+              5.3831E 01   -3.8313E 00   3.9376E 00
8.5000E 00      5.1785E 01     ---------------------------------------+               5.2809E 01   -2.8089E 00   3.9323E 00
9.0000E 00      5.0864E 01     ---------------------------------------+               5.1813E 01   -1.8127E 00   3.9271E 00
9.5000E 00      5.0108E 01     --------------------------------------+                5.0921E 01   -9.2104E-01   3.9220E 00
1.0000E 01      4.9535E 01     --------------------------------------+                5.0181E 01   -1.8129E-01   3.9169E 00
1.0500E 01      4.9144E 01     -------------------------------------+                 4.9614E 01    3.8560E-01   3.9119E 00
1.1000E 01      4.8917E 01     -------------------------------------+                 4.9221E 01    7.7939E-01   3.9069E 00
1.1500E 01      4.8828E 01     -------------------------------------+                 4.8985E 01    1.0148E 00   3.9019E 00
1.2000E 01      4.8845E 01     -------------------------------------+                 4.8884E 01    1.1161E 00   3.8969E 00
1.2500E 01      4.8937E 01     -------------------------------------+                 4.8888E 01    1.1125E 00   3.8920E 00
1.3000E 01      4.9075E 01     -------------------------------------+                 4.8966E 01    1.0338E 00   3.8869E 00
1.3500E 01      4.9233E 01     -------------------------------------+                 4.9092E 01    9.0831E-01   3.8819E 00
1.4000E 01      4.9392E 01     -------------------------------------+                 4.9240E 01    7.6045E-01   3.8768E 00
1.4500E 01      4.9537E 01     --------------------------------------+                4.9390E 01    6.0970E-01   3.8718E 00
1.5000E 01      4.9659E 01     --------------------------------------+                4.9530E 01    4.7041E-01   3.8666E 00
1.5500E 01      4.9753E 01     --------------------------------------+                4.9648E 01    3.5193E-01   3.8615E 00
1.6000E 01      4.9819E 01     --------------------------------------+                4.9741E 01    2.5908E-01   3.8564E 00
1.6500E 01      4.9859E 01     --------------------------------------+                4.9807E 01    1.9312E-01   3.8512E 00
1.7000E 01      4.9875E 01     --------------------------------------+                4.9847E 01    1.5251E-01   3.8461E 00
1.7500E 01      4.9875E 01     --------------------------------------+                4.9866E 01    1.3382E-01   3.8409E 00
1.8000E 01      4.9861E 01     --------------------------------------+                4.9867E 01    1.3263E-01   3.8357E 00
1.8500E 01      4.9839E 01     --------------------------------------+                4.9856E 01    1.4413E-01   3.8306E 00
1.9000E 01      4.9814E 01     --------------------------------------+                4.9836E 01    1.6368E-01   3.8254E 00
1.9500E 01      4.9789E 01     --------------------------------------+                4.9813E 01    1.8719E-01   3.8202E 00
2.0000E 01      4.9765E 01     --------------------------------------+                4.9789E 01    2.1126E-01   3.8150E 00
```

Figure 14. Illustration of print plot

```
            PROBLEM DURATION 0.0            TO  3.5009E 00

VARIABLE       MINIMUM        TIME          MAXIMUM        TIME
 C2CAP        4.3195E-01    1.3477E-01    6.3739E-01    3.1796E 00
 O2TIS        1.7541E 00    1.4062E-01    2.8520E 00    1.5263E 00
 PO4         -1.5799E-05    1.9375E 00    5.6000E-03    0.0
 CAP          1.0000E 00    0.0           1.0000E 00    0.0
 R            1.0309E-01    3.9844E-02    4.6622E 00    2.1484E-01
 FLOW         4.0653E-08    3.9844E-02    1.7008E-01    2.1484E-01
 O2CCAP       1.0799E 02    1.3477E-01    1.5935E 02    3.1796E 00
 PO2CAP       2.9595E 01    1.3477E-01    4.8263E 01    3.1796E 00
 FLOWAV       0.0           0.0           3.0166E-01    3.5009E 00
 PO2TAV       0.0           0.0           1.5478E 02    3.5009E 00
```

Figure 15. Illustration of RANGE output

For installations with plotting devices, a tape or disk may be prepared with the values of up to 49 variables in addition to TIME. Detailed information on the use of the PREPAR data set is given in the System Manual.

Other pertinent information is automatically provided by S/360 CSMP in the printer record of the problem, in the following order:

1. The S/360 CSMP input statements.
2. The output variable sequence, indicating the order of execution of the sorted functions
3. A problem statistics summary, showing the number of output variables, input variables, parameters, integrators, memory blocks, FORTRAN statements, and data and control cards
4. A listing of the subroutine data, as described in the System Manual
5. A listing of user-supplied subprograms
6. A listing of the subroutine Update, showing the sorted structure statements, as described in the System Manual
7. A listing of the data, execution control, and output control statements for each run, followed by the specified output for that run

SAMPLE PROBLEM

A simple example illustrating several S/360 CSMP features is the design of a radiating heat fin of the type shown in Figure 16. The fins are attached to the coolant tube in order to increase heat dissipation by thermal radiation. This method might, for example, be used to control the thermal environment of a space station power plant.

A typical design problem would be to dimension the fins such that each dissipated a specified amount of heat per hour, with a specified temperature along the root end of the fin. If the metal and the surface roughness are fixed by other considerations, the available design parameters are fin length, L, and fin thickness, H. Since physical constraints on fin length make that dimension less readily manipulated, the engineer would generally first seek a solution using only fin thickness as the design parameter.

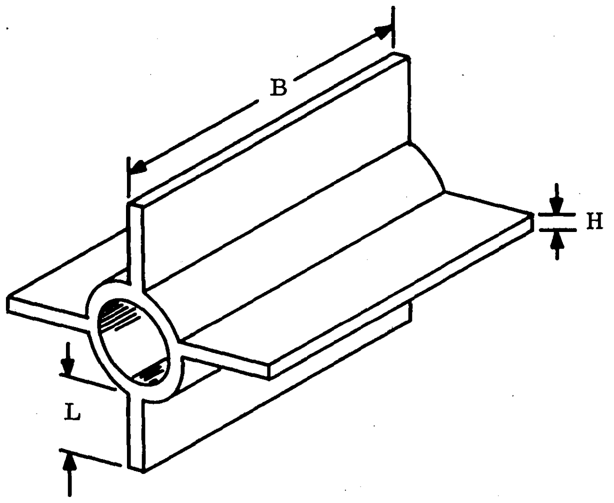


Figure 16. Radiating heat fin

In most simulation studies, it is the time or transient response that is of interest. Here, however, the engineer is concerned with the steady-state heat flow within the fin, at thermal equilibrium. Assuming negligible thermal interaction between fins, the situation is described by the following differential equation:

$$d^2T/dX^2 = 2\sigma E\,(T^4 - T_s^{\,4})\,/KH$$

where:

X = distance from fin root

T = temperature in degrees R

T_s = temperature of surrounding space

σ = Stefan-Boltzmann constant

E = thermal emissivity

K = thermal conductivity

H = fin thickness

The independent variable is X, the distance from the fin root, rather than time.

The simplicity of this differential equation is disarming. Although only a few S/360 CSMP structure statements are needed to represent the relationship, design requires solution of a two-point boundary value problem. The two conditions needed to solve a second-order differential equation are available here as one condition at X = 0.0 and the other at X = L. At the point X = 0.0, T must equal the specified, constant fin root temperature, T_0. At the edge of the fin (X = L), radiation must approach zero; that is, dT/dX must equal zero.

Solution of this type of problem involves a trial and error process. The basic procedure is to search for a value of dT/dX at X = 0.0, which, together with the specified temperature condition at X = 0.0, yields a solution that also satisfies the other end-point condition, dT/dX = 0.0 at X = L.

In this case, the actual implementation involves a search over values of the design parameter H, by use of the relationship:

$$\text{at } X = 0.0, \quad \frac{dT}{dX} = \frac{-Q_0}{KBH}$$

where Q_0 is the specified heat dissipation rate and B is the width of the fin. The fin thickness, H, is varied until the initial condition on dT/dX yields a value at the fin edge sufficiently close to zero to be considered a solution. A common approach is to make an arbitrary first guess for the design parameter and then, by means of some algorithm, successively modify that value after each run until all constraints are satisfied. This procedure may be easily automated within S/360 CSMP.

Suppose now that a solution is to be obtained for a specific fin for which;

T_s = 0°R

E = 0.8

K = 25 Btu/hr/ft/°R

B = 0.5 ft

L = 0.25 ft

T_0 = 2000.°R

Q_0 = 1000. Btu/hr

Suppose, too, that the fin thickness, H, is constrained to be less than 0.01 feet.

Figure 17 shows the complete set of S/360 CSMP statements for one possible approach to this design problem. A LABEL card provides a heading for the output that will be generated at the end of the computation. The RENAME feature is used so that, for convenience, the independent variable will be represented by the symbol X, rather than by TIME.

Note that the structure statements are separated into three segments by use of the INITIAL, DYNAMIC and TERMINAL control statements. The computations in the initial segment are performed only at the beginning of each run. For the first run, the value of H is computed using values of HIGH and LOW as specified in a CONSTANT statement. On subsequent runs during the iterative search, HIGH and LOW will be adjusted by the algorithm embedded in the terminal segment. Using this value for H, the program next calculates the variables COEF and DTDX0 for use in the dynamic segment. It should be remembered that structure statements within the initial segment are presumed to represent parallel structure. Accordingly, the actual order of the statements within this segment is of no concern. Unless the user deliberately chooses otherwise, by use of the NOSORT option or by placing statements within a PROCEDURE function, the sorting algorithm of S/360 CSMP automatically determines an appropriate computational sequence.

```
                ****CONTINUOUS SYSTEM MODELING PROGRAM****

          ***PROBLEM INPUT STATEMENTS***

 LABEL     COOLING FIN DESIGN
   RENAME        TIME = X

 INITIAL
                    H = 0.5* ( HIGH + LOW )
                 COEF = (  2.*SIGMA*E ) / ( K*H )
                DTDX0 = -Q0 / ( K*B*H )
   CONSTANT        Q0 = 1000.,     T0 = 2000.,    LOW = 0.0,     HIGH =.01,...
                SIGMA = 0.173E-8,  E = 0.8,         K = 25.0,       B = 0.5


 DYNAMIC
               D2TDX2 = COEF * ( TEMP**4 )
                 DTDX = INTGRL( DTDX0, D2TDX2 )
                    T = INTGRL( T0, DTDX )
                 TEMP = LIMIT( 0.0, T0, T )


 TERMINAL
        IF ( (HIGH - LOW) .LT. 0.00001 ) GO TO 3
        IF ( DTDX .LT. 0.0 ) GO TO 1
                 HIGH = H
        GO TO 2
     1            LOW = H
     2 CALL RERUN
     3 CONTINUE
  TIMER           DELT = 0.0001,     FINTIM = 0.25
 END


  TIMER      OUTDEL = 0.01
  PRTPLT   TEMP(DTDX,D2TDX2,H)
 END
 STOP
```

Figure 17. Listing of cooling fin design input

The statements between the DYNAMIC and TERMINAL control cards represent the dynamics of the differential equation relationship. Note that this is merely one possible representation of this relationship. There is no single "best" formulation; many others would be equivalent and just as efficient. Some users prefer to "nest" expressions in order to achieve a very compact problem formulation; others find that fewer errors occur if a single functional relationship is expressed in each statement. Note, in particular, that the integrator that develops DTDX uses, as an initial condition, the value DTDX0 computed in the initial segment. An interesting refinement of the search procedure is the use of a limiting function, applied to T, to obtain the variable TEMP. Clearly, in a correct solution, the temperature along the fin must be less than that at the fin root and greater than the temperature of the surrounding space. The limiting function imposes this restriction explicitly, as a precaution, in the event that an estimate of H causes a "wild" solution. A TIMER control card specifies the integration interval and also specifies that the dynamic computation should terminate in each run when X reaches 0.25, the fin length. Since no integration method has been specified, the program will automatically use the variable-step RKS method.

This simple example calls for only two types of S/360 CSMP functional elements: the integrator and the limiter. Typical problems would generally use many types of functional elements in addition to algebraic statements. By definition, the structure statements within a terminal segment represent procedural programming. In this example, the terminal segment implements a binary search algorithm that adjusts the values of HIGH and LOW according to the final value of DTDX. Consistent with the physics of the problem, this algorithm reduces the estimate of H used in the next run if the final value of DTDX is positive, and increases the estimate if DTDX is negative. The algorithm is designed to ensure convergence, subject to the dimensional constraints on fin thickness imposed by the HIGH and LOW values specified on the CONST data card. The designer has made his own definition of "sufficient accuracy" by his choice of the constant term in the first IF statement. When convergence is obtained, the program bypasses the adjustment algorithm. Until convergence is obtained, the program calls RERUN, thereby signifying that yet another iteration is required.

Note that no output statements are used before the first END card. This allows the entire search to occur without either tabular or plot output until convergence. When convergence occurs, the program next encounters statements requesting a print plot of the temperature profile. S/360 CSMP then performs one additional run, using the final value of H obtained from the iterative search, and outputs the results of this run as requested on the TIMER and PRTPLT control cards.

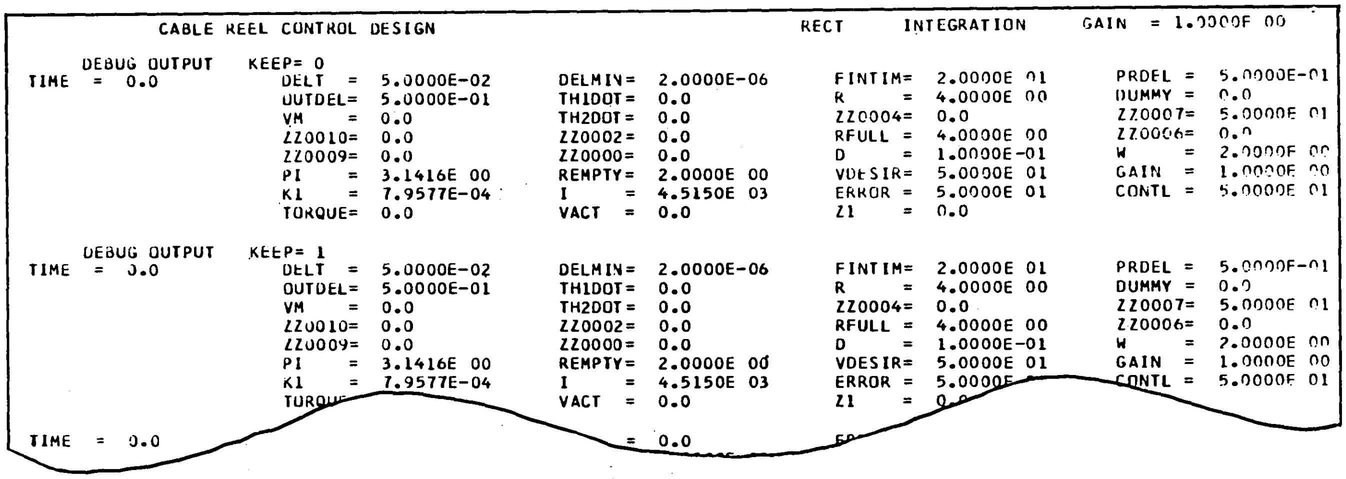
PROBLEM CHECKOUT FACILITIES

The program provides an option for the printing of the current values of variables at a specified time, as well as diagnostic messages when the program detects a suspected error.

The subroutine DEBUG, which is used to obtain the values, may have considerable utility in the problem checkout phase. The subroutine is called as follows:

```
CALL  DEBUG( n, t )
```

where n must be an integer constant and t is a real constant. The statement can be used any number of times in a run. The variable names and current values are printed for each simulation variable, including system variables and parameters. The DEBUG output will occur a maximum of n times during a run; the first output will occur when TIME equals or exceeds the real constant t. Figure 18 illustrates this checkout feature. Note that output with system variable KEEP equal to zero is a trial evaluation; that with KEEP = 1 represents a valid integration step.



```
            CABLE REEL CONTROL DESIGN                                RECT     INTEGRATION     GAIN  = 1.0000F 00

       DEBUG OUTPUT    KEEP= 0
TIME  =  0.0                DELT   =  5.0000E-02      DELMIN=  2.0000E-06       FINTIM=  2.0000E 01      PRDEL  =  5.0000E-01
                            OUTDEL=  5.0000E-01      TH1DOT=  0.0              R      =  4.0000E 00      DUMMY  =  0.0
                            VM     =  0.0            TH2DOT=  0.0              ZZ0004=  0.0              ZZ0007=  5.0000E 01
                            ZZ0010=  0.0             ZZ0002=  0.0              RFULL  =  4.0000E 00      ZZ0006=  0.0
                            ZZ0009=  0.0             ZZ0000=  0.0              D      =  1.0000E-01      W      =  2.0000F 00
                            PI     =  3.1416E 00     REMPTY=  2.0000E 00       VDESIR=  5.0000E 01       GAIN   =  1.0000E 00
                            K1     =  7.9577E-04     I      =  4.5150E 03       ERROR  =  5.0000E 01      CONTL  =  5.0000E 01
                            TORQUE=  0.0             VACT   =  0.0              Z1     =  0.0

       DEBUG OUTPUT    KEEP= 1
TIME  =  0.0                DELT   =  5.0000E-02      DELMIN=  2.0000E-06       FINTIM=  2.0000E 01      PRDEL  =  5.0000F-01
                            OUTDEL=  5.0000E-01      TH1DOT=  0.0              R      =  4.0000E 00      DUMMY  =  0.0
                            VM     =  0.0            TH2DOT=  0.0              ZZ0004=  0.0              ZZ0007=  5.0000E 01
                            ZZ0010=  0.0             ZZ0002=  0.0              RFULL  =  4.0000E 00      ZZ0006=  0.0
                            ZZ0009=  0.0             ZZ0000=  0.0              D      =  1.0000E-01      W      =  2.0000E 00
                            PI     =  3.1416E 00     REMPTY=  2.0000E 00       VDESIR=  5.0000E 01       GAIN   =  1.0000E 00
                            K1     =  7.9577E-04     I      =  4.5150E 03       ERROR  =  5.0000       CONTL  =  5.0000F 01
                            TUR                      VACT   =  0.0              Z1     =

TIME  =  0.0                                                =  0.0
```

Figure 18. Illustration of DEBUG output

DEBUG may be used only from a procedural section. While permissible from such a section of the INITIAL segment, it is ordinarily called from the DYNAMIC segment. A restriction on use of DEBUG is that it must be called at time zero; logical control statement should not be used in such manner as to "branch around" the statement at time zero. This restriction is imposed in order that the DEBUG counters may be properly initialized. DEBUG may not be used from the TERMINAL segment since this segment is not entered at time zero. This DEBUG feature of S/360 CSMP should not be confused with the FORTRAN IV (G) Debug Facility. The latter is not supported within S/360 CSMP and may cause inaccurate numerical results.

Most commonly, DEBUG is called after all other computations concerned with model structure are completed. The best way to accomplish this is to define a procedural section at the end of the DYNAMIC segment as follows:

```
        •

        •

    NOSORT
      CALL  DEBUG( 20, 15.0 )
    TERMINAL

        •
```

This usage would result in a maximum of 20 DEBUG outputs, the first occurring no earlier than 15.0 time units. All values printed at each output would represent computations for the current call to the UPDATE subroutine.

Sometimes it is helpful to check on the performance of a specific statement within the model. This is readily accomplished by grouping that statement and a call to DEBUG as part of a PROCEDURE. Suppose, for example, that a model includes the statement

```
Y = SQRT( X )
```

and that X, for reasons not apparent, assumes negative values during portions of the simulation. One might investigate this situation as follows:

```
PROCEDURE       Y = TEST( X )
      IF( (TIME .EQ. 0.0) .OR. (X .LT. 0.0) ) CALL DEBUG( 10, 0.0 )
                Y = SQRT( X )
ENDPRO
```

For testing a user-supplied function, one might extend this technique, calling DEBUG both immediately before and immediately after use of the suspect function. Note also that DEBUG may be used within a procedural MACRO or a PROCEDURE contained within a MACRO. Finally, it should be mentioned that if debugging requires only a few values, rather than those on the entire model, the FORTRAN "WRITE" statement may be used as an alternative to DEBUG.

DIAGNOSTIC MESSAGES

Diagnostic messages may occur during both the translation and execution phases of the program and are designed to be self-explanatory. Some of the diagnostic checks detect illegal characters or incorrect syntax; the symbol "$" is printed below the detected error prior to the associated diagnostic message. A "warning only" message is printed when an error is not wholly discernible in translation or does not destroy the "validity" of simulation. Some examples of these errors are:

Output variable name not unique

Control variable name not a systems variable

Parameter value not specified

Variable used as input to a section not available from any prior section

Some examples of errors causing a run halt at the end of translation are:

Incorrect structure or data statement format

Invalid data card type

Unspecified implicit loop

RELERR specification on other than an integrator output name

Examples of errors causing a run halt during execution are:

Failure of an integration or implicit function to meet the error criterion

A misspelled subroutine name

The following is a list of the diagnostic messages with their explanations and suggested corrections:

"CALL RERUN" CAN ONLY BE USED IN A TERMIN SEGMENT.
PROBLEM TERMINATED.

CALL RERUN can be used only in a TERMINAL segment. If it is used elsewhere, the run terminates.

CSMP STATEMENT INCORRECTLY WRITTEN

The translation phase has detected an error in the statement printed before this message. The statement should be checked carefully, including parentheses and commas. Although translation of the source statements will continue, the run will be terminated before the execution phase.

CSMP STATEMENT OUT OF SEQUENCE

The sequence of input statements cannot be processed and the run will be terminated before the execution phase. The statement should be checked for sequence in the input deck to see if it has been misplaced. MACRO definitions must precede all structure statements. An INITIAL segment, when used, must precede the DYNAMIC segment. If used, the TERMINAL segment must follow the DYNAMIC segment.

DATA HAS NOT BEEN SPECIFIED FOR AN AFGEN FUNCTION
DATA HAS NOT BEEN SPECIFIED FOR AN NLFGEN FUNCTION

An AFGEN (or NLFGEN) function generator has been used in a structure statement but the corresponding data has not been specified using the FUNCTION statement. The run will be terminated.

DYNAMIC STORAGE EXCEEDED. THIS CASE CANNOT BE RUN.

The 8000-word limitation on simulator data storage has been exceeded. The storage in this array includes the current values of model variables, function and error tables, central integration history, and subscripted variable values. The problem should be analyzed to determine where equations can be combined to reduce the number of required entries in the array.

ERROR - CENTRAL INTEGRATION ROUTINE NOT SUPPLIED

The user has used the word CENTRL for his integration method on the METHOD execution control card; however, he has not supplied the integration deck to the program. The run will be terminated.

ERROR IN COORDINATE ENTRIES

An error has been detected in the previously printed FUNCTION data statement. There is either an odd number of entries in the data table or an improper sequence of X-coordinate values. The run will be terminated.

ERROR IN TABLE ENTRY

In the previously printed TABLE data statement, an error has been detected. Although reading of the data statements will continue, the run will be terminated before execution.

ERROR IN PRINT-PLOT STATEMENT

An error has been detected in the PRTPLOT output control statement. The statement should be checked for a correct number of parentheses and commas for specifying lower and upper limits, particularly if one or the other is missing, and commas are used to indicate this. Although the run continues, everything on the card after the error is disregarded.

EXCEEDED MAXIMUM ITERATIONS ON IMPLICIT LOOP

One hundred iterations of the implicit loop have been run and convergence has not yet occurred. The run has been terminated. One possibility is to change the error condition, so that the convergence criteria can be met.

FINTIM IS ZERO. THIS CASE CANNOT BE EXECUTED.

FINTIM either has not been specified or has been specified as being equal to zero.

GENERATED STATEMENT No. xx

The translator has detected an error during generation of statement xx of an invoked MACRO in the structure of the model. Carefully check the corresponding statement of the MACRO definition for proper spelling and punctuation.

ILLEGAL CHARACTER OR DOUBLE OPERATOR

In the previously printed statement, an illegal character or double operator has been detected. Although translation of the source statements will continue, the run will be terminated before the execution phase.

INCORRECT IMPLICIT STATEMENT

The translation phase has detected an error in the IMPL structure statement printed before this message. The statement should be checked to see that the third argument is the output name of the last statement in the definition and that the block output appears at least once to the right of an equal sign. Although translation of the source statements will continue, the run will be terminated before the execution phase.

INCORRECT MACRO STATEMENT

The translation phase has detected an error in the MACRO use statement printed before this message. The statement should be checked to ensure that the number of arguments and outputs is correct and that argument list ends with a parenthesis. Although translation of the source statements will continue, the run will be terminated before the execution phase.

INCORRECT TIMER VAR. NAME**WARNING ONLY

One of the system variable names (FINTIM, DELT, PRDEL, OUTDEL, or DELMIN) has been misspelled on the TIMER execution control card. The user should also check the possibility that the system variable has been renamed. Although the run will continue, the system variable misspelled will be unchanged.

INPUT NAME SAME AS OUTPUT NAME

The output variable name to the left of the equal sign has also been used as an input name on the right side of the equal sign. Except as output of a memory type functional element, such usage is not permissible in a parallel, sorted section. The run will be terminated.

INPUT TO FUNCTION GENERATOR nnnnnn BELOW SPECIFIED RANGE
INPUT = xxxx.xxxx

The input (xxxx.xxxx) to the function generator named nnnnnn is below the minimum specified range. The program will take the value for the minimum specified and continue. This message will be printed only once, even though the condition is reached several times.

INPUT TO FUNCTION GENERATOR nnnnnn ABOVE SPECIFIED RANGE
INPUT = xxxx.xxxx

The input (xxxx.xxxx) to the function generator named nnnnnn is above the maximum specified range. The program will take the value for the maximum specified and continue. This message will be printed only once, even though the condition is reached several times.

LABEL INCORRECTLY WRITTEN

The label used in the preceding statement cannot be recognized by the program. Check for proper spelling. The statement will be disregarded; the run will continue.

MACRO xxxxxx WITHIN MACRO yyyyyy USED IN A PROCEDURAL SECTION

MACROs, separately defined, may be invoked within the definition of other MACROs if overall parallel structure is implied. Invocation of a MACRO within a PROCEDURE within a MACRO definition is therefore not permissible. Similarly, a MACRO containing other MACROs in its definition may not be invoked from a PROCEDURE or from a procedural section.

MORE THAN 10 PRTPLOT STATEMENTS

More than ten PRTPLOT output control statements have been specified. Only the first ten will be used.

NUMBER EXCEEDS 12 CHARACTERS

In the previously printed statement, a number exceeding twelve characters in a MACRO argument or integrator block initial condition has been detected. Although translation of the source statements will continue, the run will be terminated before the execution phase.

NUMBER INCORRECTLY WRITTEN

In the previously printed statement, a number written incorrectly has been detected. If detected during the translation phase, translation of the source statements will continue; however, the run will be terminated before the execution phase.

ONLY FIRST 10 CONDITIONS FOR JOB END WILL BE TESTED

More than ten specifications have been given with the FINISH execution control statement. Although the run will continue, only the first ten specifications will be used.

ONLY FIRST 50 VALUES WILL BE USED

The multiple value form of the PARAMETER data statement has been used with more than 50 values for the parameter. A sequence of runs will be performed, but using only the first 50 values for the parametric study.

ONLY FIRST 50 VARIABLES WILL BE PREPARED

More than 50 variables (including TIME) have been specified with PREPARE or PRTPLOT output control statements. Although the run will continue, only the first 50 variables will be used.

ONLY FIRST 50 VARIABLES WILL BE PRINTED

More than 50 variables (including TIME) have been requested with PRINT execution control statements. Only the first 50 will be printed; others will be ignored.

ONLY FIRST 100 VARIABLES WILL BE RANGED

More than 100 variables (including TIME) have been specified with the RANGE output control statement. Although the run will continue, only the first 100 variables will be used.

ONLY LAST VALUE OF FAMILY USED FOR CONTIN RUN

A multiple-value parameter has been used with a CONTINUE card. THE CONTINUE control feature will be implemented only with the last value of the parameter. This is a warning message; the run will continue.

OUTPUT NAME HAS ALREADY BEEN SPECIFIED

In the previously printed statement, the output variable name to the left of the equal sign has been used before as an output variable name; that is, it has occurred to the left of the equal sign in a preceding section. The run will be continued.

PARAMETERS NOT INPUT OR OUTPUTS NOT AVAILABLE TO SORT SECTION***SET TO ZERO***

A list of variable names will be printed following this heading. The run is continued. Variables that are not parameters specified on data cards are set to zero. Output variable names that are not available to this sort section are initially set to zero, but may change as the problem is run.

PROBLEM CANNOT BE EXECUTED

At least one diagnostic message will have been printed among the source statements indicating the reason why the problem cannot be executed. The run will be terminated.

PROBLEM INPUT EXCEEDS TRANSLATION TABLE nn

During translation of the problem, a table has been exceeded and the run will terminate. The specific table is identified by nn in the following list:

nn	Translation Table
1	More than 500 statement output names
2	More than 1400 statement input names
3	More than 400 parameter names
4	More than 300 INTGRL or MEMORY outputs
5	More than 1400 input names and unique block names
6	More than 20 FIXED variable names
7	More than 100 initial condition numeric values
8	More than 10 FORTRAN specification cards
9	More than 100 unique block names and symbolic names with first letter I, J, K, L, M, or N but not appearing on FIXED statements
10	More than 25 STORAGE variable names
11	More than 15 sections (SORT or NOSORT)

nn	Translation Table
12	More than 100 MACRO arguments, outputs, and statement numbers for one MACRO
13	More than 50 MACRO functions
14	More than 120 MACRO definition cards
15	More than 50 HISTORY or MEMORY functions
16	More than 15 MEMORY functions
17	More than 85 variables that are neither parameters specified on data cards nor outputs of a following SORT section
18	More than 150 duplicate names in COMMON (outputs or inputs to INTGRL blocks)
19	More than 100 parameters in one SORT sequence
21	More than 600 structure statements in a single SORT section

PRTPLOT, PREPARE, AND RANGE VARIABLES EXCEED 100. ALL RANGE VARIABLES STARTING WITH xxxxxx HAVE BEEN DELETED.

More than 100 variables (including TIME) have been specified with PRTPLOT, PREPARE, and RANGE output control statements. Although the run will continue, only the first 100 variables will be used for this run.

RERUN FROM TERMIN CANCELED FOR CONTIN RUN

The TERMINAL segment cannot be used to cause a rerun when a CONTINUE translation control statement started the run. The TERMINAL computation statements will be executed but any CALL RERUN will be ignored.

SIMULATION HALTED

The run was terminated because a FINISH condition was satisfied. The variable name and its value are printed.

SIMULATION INVOLVES AN ALGEBRAIC LOOP CONTAINING THE FOLLOWING ELEMENTS

A list of output variable names will be printed following this diagnostic. The sort subprogram has been unable to find an integration or memory block in the loop involving these variables. The run will be terminated before the execution phase.

SYMBOLIC NAME xxxxxx NOT DEFINED

An error has been detected on the PARAMETER, INCON, CONSTANT, or TIMER card printed before this message. Although input to the execution phase will continue, the simulation will not be run.

SYMBOLIC NAME EXCEEDS 6 CHARACTERS

In the previously printed statement, a symbolic name exceeding six characters has been detected. Although translation of the source statements will continue, the run will be terminated before execution.

SYMBOLIC NAME INCORRECTLY WRITTEN

In the previous statement, an incorrectly written symbolic name has been detected. The run will be terminated.

TOO MANY CONTINUATION CARDS. MAX=n

The previously printed statement has been continued on too many cards. If N = 3, a MACRO label statement has over three continuation statements. If N = 8, a structure statement has over eight continuation statements. The user should make multiple statements or use more columns on individual cards. Although translation of the source statements will continue, the run will be terminated before the execution phase.

TOO MANY LEFT PARENTHESES
TOO MANY RIGHT PARENTHESES

Too many left (or right) parentheses have been detected in the statement printed before this diagnostic. Although the translation of the source statements will continue, the run will be terminated before the execution phase.

VARIABLE STEP DELT LESS THAN DELMIN. SIMULATION HALT.

The simulation will not be continued because the specified DELT is less than the specified DELMIN.

METHODS

This section of the manual describes some of the basic techniques and restrictions of the program. The topics included are:

1. Integration techniques
2. Dynamic storage allocation
3. Program restrictions
4. Reserved words
5. Statement ordering
6. System macros

INTEGRATION TECHNIQUES

S/360 CSMP uses centralized integration. This means that all integration statements are placed at the end of the structure coding, so that all current inputs to integration functional blocks are defined before integration. A single routine is then used to update each of the integrator output variables used in the simulation.

Several different types of routines are available to perform the integration operation. They include both fixed integration step-size routines and variable step-size routines. Five fixed step-size routines are available: fixed Runge-Kutta, Simpson's, trapezoidal, rectangular, and second-order Adams. Two variable step-size routines are available: fifth-order Milne predictor-corrector and fourth-order Runge-Kutta. In these latter routines, the integration interval Δt is varied, under program control, during problem execution. Both routines provide an estimate of the integration error, which is compared with a user-specified error bound. The step size is adjusted accordingly by the program.

If none of the above methods satisfies the user's requirement, a dummy integration routine named CENTRL is provided to allow the user to specify his own integration method. He enters it into S/360 CSMP by giving it the name CENTRL. The System Manual contains a complete discussion of the steps involved.

The mathematics of the integration methods are as follows:

1. Milné Fifth-Order Predictor-Corrector

Predictor:

$$Y^p_{t+\Delta t} = Y_{t-\Delta t} + \frac{\Delta t}{3}\left(8 \cdot X_t - 5X_{t-\Delta t} + 4 \cdot X_{t-2\Delta t} - X_{t-3\Delta t}\right)$$

Corrector:

$$Y^c_{t+\Delta t} = \frac{1}{8}\left(Y_t + 7 \cdot Y_{t-\Delta t}\right) + \frac{\Delta t}{192}\left(65 \cdot X_{t+\Delta t} + 243. X_t + 51. X_{t-\Delta t} + X_{t-2\Delta t}\right)$$

Estimate:

$$Y_{t+\Delta t} = .96116 \cdot Y^c_{t+\Delta t} + .03884 \cdot Y^p_{t+\Delta t}$$

Integration interval control is based on the following criteria:

$$\frac{0.04\,|Y^c - Y^p|}{R \cdot |Y^c|} \leq 1 \qquad |Y^c| > 1$$

$$\frac{0.04\,|Y^c - Y^p|}{R} \leq 1 \qquad |Y^c| \leq 1$$

2. Runge-Kutta (Fourth Order)

$$Y_{t+\Delta t} = Y_t + \frac{1}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right)$$

$$K_1 = \Delta t \cdot f\left(t, Y_t\right)$$

$$K_2 = \Delta t \cdot f\left(t + \frac{\Delta t}{2}, Y_t + \frac{K_1}{2}\right)$$

$$K_3 = \Delta t \cdot f\left(t + \frac{\Delta t}{2}, Y_t + \frac{K_2}{2}\right)$$

$$K_4 = \Delta t \cdot f\left(t + \Delta t, Y_t + K_3\right)$$

If variable-step integration is used, the interval will be reduced to satisfy the following criterion:

$$\frac{|Y_{t+\Delta t} - Y^s|}{A + R \cdot |Y_{t+\Delta t}|} \cong \frac{\text{Error}}{A + R \cdot |Y_{t+\Delta t}|} \leq 1$$

where Y^s is $Y_{t+\Delta t}$ calculated by Simpson's Rule, and A and R are the absolute and relative errors corresponding to the particular integrator value.

3. Adams-Secónd Order

$$Y_{t+\Delta t} = Y_t + \frac{\Delta t}{2}\left(3 \cdot \dot{Y}_t - \dot{Y}_{t-\Delta t}\right)$$

4. Simpson's Rule

Predictor:

$$Y^p_{t+\frac{\Delta t}{2}} = Y_t + \frac{\Delta t}{2} X_t$$

$$Y^p_{t+\Delta t} = Y^p_{t+\frac{\Delta t}{2}} + \frac{\Delta t}{2} X_{t+\frac{\Delta t}{2}}$$

Corrector:

$$Y^{C}_{t+\Delta t} = Y_t + \frac{\Delta t}{6}\left(X_t + 4\,X_{t+\frac{\Delta t}{2}} + X_{t+\Delta t}\right)$$

5. Trapezoidal

Predictor:

$$Y^{p}_{t+\Delta t} = Y_t + \Delta t \cdot X_t$$

Estimate:

$$Y_{t+\Delta t} = Y_t + \frac{\Delta t}{2}\cdot\left(X_t + X_{t+\Delta t}\right)$$

6. Rectangular

$$Y_{t+\Delta t} = Y_t + \Delta\, t \cdot X_t$$

NOTE: In these equations, the common terminology is $X_t \equiv \dot{Y}_t \equiv f(t)$. A value $X_{t+\Delta t}$ used in the estimation is based on the prediction $Y^{p}_{t+\Delta t}$.

DYNAMIC STORAGE ALLOCATION

S/360 CSMP uses dynamic storage allocation. Storage areas are assigned, at execution time, to tables, integration history, etc., on the basis of the actual problem requirement. Standard S/360 CSMP functions as well as FORTRAN functions are loaded only if used. The alternative approach would be to assign fixed areas to these tasks and functions. Dynamic allocation leaves more space available for the user's program because routines and tables not required do not take up core.

PROGRAM RESTRICTIONS

Dynamic storage allocation is not practical during the translation phase. The size of certain tables used during that phase, therefore, have been set at fixed values. Consequently, for some of the components, there are limitations as to how many can be used in a simulation. The following list of restrictions may be of value when a very large problem is to be solved with S/360 CSMP:

	Item	Maximum
1.	Output variable names, including intermediate output names generated by S/360 CSMP for MACRO and INTGRL functions	500
2.	Input variable names, including parameter names	1400
3.	Parameter names	400
4.	Integrators plus statements with memory and history functions	300
5.	User-supplied memory and history functions	50
6.	User-supplied memory functions	15
7.	Tables (STORAGE variables)	25
8.	Structure statements, including those generated by S/360 CSMP for MACRO and INTGRL functions in a single SORT section	600
9.	Simulator data storage locations, including areas for the current values of model variables, function and error tables, centralized integration history, and subscripted variable values	8000
10.	Print output variables, including TIME	50
11.	Print-plot output variables, including TIME	50
12.	PREPARE output variables, including TIME	50
13.	Range variables, including print-plot and PREPARE variables	100
14.	FINISH specifications, not including FINTIM	10
15.	Statements sent directly to FORTRAN (identified by a / in cc 1)	10
16.	Fixed-point variable names	20

These restrictions can be modified for specific uses of available core or to take advantage of additional core. The method is discussed in the System Manual.

RESERVED WORDS

Because the words listed below have been reserved by the FORTRAN compiler, they should be used in the input program only as specified by FORTRAN. They may not be used as variable or subprogram names.

ABS	GO
	GOTO
BACKSPACE	
CALL	IABS
COMMON	IDIM
CONTINUE	IF
	IFIX
DABS	INTEGER
DBLE	ISIGN
DEFINE	
DIM	
DIMENSION	PAUSE
DFLOAT	
DO	
DOUBLE	READ
DSIGN	REAL
	RETURN
END	REWIND
ENDFILE	
EQUIVALENCE	
EXIT	SIGN
EXTERNAL	SNGL
	STOP
FIND	SUBROUTINE
FLOAT	
FORMAT	
FUNCTION	WRITE

In addition to the words reserved by the FORTRAN compiler, S/360 CSMP has the following restrictions on variable names:

1. Certain variable names are reserved for use of the system, and cannot appear in a S/360 CSMP structure statement. These names are NALARM, IZxxxx, and ZZxxxx, where x is any digit.

2. KEEP is a COMMON variable, but it can be used consistent with its intended purpose.

3. The names DELT, DELMIN, DELMAX, FINTIM, TIME, PRDEL, and OUTDEL are system reserved names and must appear only in their intended context unless RENAMEd. This has been explained under "Input Language", in the paragraphs on TIMER execution and RENAME translation control statements.

4. TIME is the name for the independent variable and should be used only for that purpose.

5. S/360 CSMP subroutines must be used only as intended. These are MAINEX, INTRAN, RKS, MILNE, RECT, TRAPZ, SIMP, CENTRL, ADAMS, INITLZ, NUMER, ALPHA, DEBUG, UPDATE, F, PLOTR, CSTORE, BUILDR, SPLITR, UND, ZOR, COMPL, MLEFT, MRIGHT, BOOLE, SHIFT, and the standard S/360 CSMP functional block names.

STATEMENT ORDERING

S/360 CSMP was designed as a nonprocedural language to free the user from the task of sequencing his input. As has been noted, however, there are some situations when he must observe statement ordering rules. They are summarized here as follows:

1. PROCEDURE functional blocks are not sorted internally.
2. NOSORT functional blocks are not sorted.
3. MACRO block definitions must be placed in the deck before any structure statements, including any in an initializing section. Data and control statements cannot be embedded in MACRO definitions.
4. STORAGE, HISTORY, and MEMORY translation control cards must appear before the first reference to the related functions.
5. RENAME translation cards must appear before the first reference to the variable being renamed or to their new name.
6. Translation control cards and the corresponding statements identifying an initializing computation must be placed in the deck before any other structure statements.
7. To ensure proper RESETing, the RESET card should be placed immediately after the END or CONTINUE card.
8. Statements defining the operations in an implicit loop must be placed immediately after the corresponding IMPL structure statement.
9. Multiple SORT sections require the inputs to have been computed before each section or within the section.
10. The translation control statement END or CONTINUE delineates the set of statements defining a run.
11. The translation control statement STOP must follow the last END card in order to define a sequence of runs of the same structure statements.
12. The translation control statement ENDJOB or ENDJOB STACK must be used to signify the end of a job. It must follow either the STOP card or user-supplied subprograms, if any.
13. Continue cards (. . .) must immediately follow the cards they continue.
14. FORTRAN FORMAT continuation cards must have a $ in cc 6 and must immediately follow the cards they continue.

SYSTEM MACROS

The four functional blocks REALPL, CMPXPL, MODINT, and LEDLAG are system macros and cannot be used in user-defined subprograms. These four functions produce inline structure statements using the INTGRL function and are not called as subroutines, as are the other functions. (Similarly, INTGRL, cannot be called from user-defined subprograms, since all INTGRL statements are calculated at the end of each iteration.

APPENDIX: INDEX OF S/360 CSMP FUNCTIONS AND STATEMENTS

FUNCTIONS

GLOSSARY

Block diagram. A diagrammatic representation of the interconnection of functional blocks constituting the simulation model.

Continuous system. A system that can be adequately modeled by a set of differential equations in which time is the independent variable.

Continuous system simulator. A simulation language that provides the block modeling capability of the digital analog simulators plus a powerful algebraic and logical modeling capability.

Digital analog simulator. A simulation language that provides a complement of functional block elements and a block-oriented language for specifying their interconnection.

Discrete system. A system that can be adequately modeled by a sequence of events that occur at discrete points in time.

Functional block (function). The basic structural unit of the simulation configuration.

History function. A function for which the output depends on both the present value of the input, and the past values of the input and output.

Memory function. A function for which the output depends only on past values of the input and output.

Nonlinear element. An element in which the output is not directly proportional to the input but is some complex function of it--for example, square, exponential, sine.

Parameter. A value associated with a specific block to particularize the desired functional operation. For example, the parameters associated with a limiter define its upper and lower limits.

Procedural program. A program in which the order of presentation of language statements determines the order of execution of the corresponding computer operations.

Simulation (modeling). The act of representing some aspects of the real world by numbers or by symbols that may be easily manipulated to facilitate their study.

Time-variant system. A system in which the parameters defining the functional relationships between variables are not constant but vary with time.

BIBLIOGRAPHY

1130 Continuous System Modeling Program, Application Description (H20-0209-1), IBM Corporation, Data Processing Division, 112 East Post Road, White Plains, New York, 1966.

Brennan, R.D., and Linebarger, R.N., "A Survey of Digital Simulation: Digital Analog Simulator Programs", Simulation, vol. 3, no. 6, December 1964, pp. 22-36.

Clancy, J.J., and Fineberg, M.S., "Digital Simulation Languages: A Critique and Guide", 1965 Fall Joint Computer Conference, vol. 27, pp. 23-26.

Syn, W.M., and Linebarger, R.N., "DSL/90--A Digital Simulation Program for Continuous System Modeling", 1966 Spring Joint Computer Conference, April 26-28, 1966.

James, M.L., Smith, G.M., and Wolford, J.C., Analog and Digital Computer Methods in Engineering Analysis, International Text Book Co., Scranton, Pennsylvania, 1964, pp. 171-179.

READER'S COMMENT FORM

GH20-0367-3

S/360 Continuous System Modeling

Program User's Manual

Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you wish a reply, be sure to include your name and address.

COMMENTS

fold

fold

fold

fold

• Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
FOLD ON TWO LINES, STAPLE AND MAIL.

GH20-0367-3

YOUR COMMENTS PLEASE...

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

Please note that requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or the IBM branch office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY...

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Technical Publications

fold fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

S/360 Continuous System Modeling Program User's Manual Printed in U.S.A. GH20-0367-3

IBM Technical Newsletter

Base Publ. No. GH20-0367-3, -2

This Newsletter No. GN20-2329

Date February 26, 1971

Previous Newsletter Nos. None for GH20-0367-3
GN20-1938 and GN20-2039
for GH20-0367-2.

SYSTEM/360 CONTINUOUS SYSTEM
MODELING PROGRAM
USER'S MANUAL

PROGRAM NUMBER 360A-CX-16X

This Technical Newsletter, a part of Version 1, Modification Level 2, of System/360 Continuous System Modeling Program, provides replacement pages for the subject manual. These replacement pages remain in effect for subsequent versions and modifications unless specifically altered. Pages to be inserted and/or removed are listed below.

Table of Contents
5-6
6.1 (added)
17 - 18
25 - 31
31.1 (added) - 32
35 - 38
45 - 46

Changes are indicated by a vertical rule in the left margin.

Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Technical Publications Dept., 1133 Westchester Avenue, White Plains, N. Y. 10604

Printed in U.S.A.

GH20-0367-3

S/360 Continuous System Modeling Program User's Manual Printed in U.S.A. GH20-0367-3

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N. Y. 10601
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)