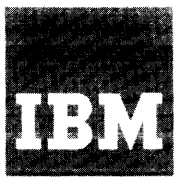
File No. 7040-25
Form C28-6800-0



Systems Reference Library

IBM 7040/7044 Remote Computing System

Preliminary Specifications

This publication provides the information necessary to plan for the use of the IBM 7040/7044 Remote Computing System. It contains a description of the system, which provides up to 40 remotely located users with concurrent access to a computer. This publication also describes the Remote Computing System language, which is similar to the FORTRAN language.

Two methods of using the Remote Computing System are mentioned. The first method, called "conversational processing," will become available first and is described in detail. The second method, called "batch processing," which will become available later, is described briefly.

Preface

This publication has been written for those users who have a basic knowledge of FORTRAN programming. It is assumed that the reader is familiar with the *General Information Manual FORTRAN*, Form F28-8074.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM Corporation, Programming Systems Publications, Dept. D39, 1271 Avenue of the Americas, New York, N. Y. 10020

Contents

Introduction to the IBM Remote Computing System

The IBM 7040/7044 Remote Computing System is a programming system designed to provide concurrent computer access to a maximum of 40 remotely located users. The programming language employed by the user is upwardly compatible with most FORTRAN IV processors and is augmented by a set of operating statements. The user communicates with the system either in a conversational manner in which his input is processed one statement at a time, or in a batch manner in which the unit of processing is an entire program.

The design of the Remote Computing System is aimed at accommodating the following demands:

The remote user does not have access to experts for programming assistance and advice. If he uses a problem-oriented language to express his problem, he requires that the request for and display of debugging data be consistent with this programming language.

Because the remote user operates the machine himself when processing his jobs, there is no machine operator to regulate the system or to perform console operations. The lack of a machine operator requires that the operating statements used to regulate the system should be similar to the programming language of the system. That is, the remote user should be able to regulate the system using statements that can be entered in the same format as program statements. Similarly, the lack of a machine operator requires that the remote user be given access to many facilities available to the machine operator in the form of console buttons, lights, and switches. He also needs the ability to stop his machine at any time without loss of data, so that he can perform such simple functions as changing printer paper, placing more input cards in a reader, or discontinuing a job.

The remote user is very sensitive to high input/output volumes. He must be able to modify decks without their complete retransmission, and he should have the option to list and inspect output data selectively, rather than be forced to transmit entire output files. Also in this spirit, he desires to keep his various decks in random storage where they are quickly and conveniently available for modification, processing, or review.

The remote user should be given the impression that he is the only user and that he is in complete control of the situation. More specifically, in a time-sharing environment, he should be totally secure from unwanted, possibly destructive, interaction by others. Finally, the remote user must be able to begin his jobs any time without waiting for other users and to continue using the system as long as is necessary.

These demands are met in the following ways:

Output data is as problem oriented as the source language. The source language is at least as easy to learn as the FORTRAN language.

Diagnostic messages and logical analysis are definite enough to allow program debugging to take place at the same level as program construction.

The user has immediate and sustained access to the computer.

The user has the ability to execute and alter programs, to change values, variables, and formulas, and to request information selectively.

Each user will be assigned an identification code that will be used by the Remote Computing System to determine which programs in its library belong to which user. Each time a user begins operating from a terminal, he will enter his identification code to gain access to his previously entered programs and to identify new programs about to be entered. The identification code also prevents users from executing, changing, or destroying other users' programs.

The print volume may be minimized, without loss of quality, on demand of the user.

The Approach

The approach to satisfying these demands fuses the old technique of interpretive execution with the relatively new one of time sharing. Interpretive execution retains all the information contained in the user's source program and thereby makes symbolic debugging possible. Time sharing allows immediate and sustained access to a computer for a large number of users. Together, these two techniques make the conversational mode of operation by remote users a practical reality.

Nevertheless, the service this system performs is not a matter of cleverly getting something for nothing: it is a justifiable trade-off. Execution time is greater, but elapsed solution-time, i.e., the time it takes to apply the computer to a problem and obtain usable results, is significantly smaller. In short, this system converts some of the raw power of the computer into condensed solution-time and greater creative power for the users.

System Concepts

The language of the Remote Computing System comprises two kinds of statements: program statements and operating statements. Program statements are upwardly compatible with FORTRAN IV and are used to construct the program. Operating statements allow the user to regulate the Remote Computing System.

When a program is being tested, executed, constructed, or modified, it is said to be active for the terminal being used. When a program is active, it is located in temporary storage, where it is known as the active image of the program.

The Remote Computing System maintains a log of operations that take place between the computer and each terminal. The log contains such information as the number of statements handled, the number and types of errors detected, and the volume of output produced. The information in the log can be used for various purposes. For example, the number of errors might indicate that additional training might be helpful. Similarly, if an individual terminal is always busy, it might indicate the need for an additional terminal. If the cost of the system is shared among terminals according to usage, the information in the log can be used for billing purposes.

Computer and Terminals

The Remote Computing System requires: (1) a computing center equipped with a data processing system having tape, drum, and disk storage, plus an exchange device, and (2) up to 40 remote terminals.

The exchange device controls the flow of information between the computer and the terminals. Characters typed at the terminals are formed into statements within the exchange device and then are sent to the computer one statement at a time. The computer returns an answer to the exchange device, which, in turn, sends it to the proper terminal. An exchange device allows each terminal to send or receive data independent of all other terminals.

Programs are permanently stored in disk storage. When the user indicates that a program he has constructed is to be saved, the Remote Computing System places it in disk storage. Thereafter, the program will be available to the user whenever needed.

Files sent from a terminal for batch processing are placed on an auxiliary disk storage device by the exchange device, in addition to those generated when operating in the batch manner. When the computer is available for batch processing, the computer operator will indicate to a subsystem under the System Monitor that files on the auxiliary disk storage device are to be processed. Under control of the subsystem, the files will be transferred from disk storage to tape, which is then used as the system input unit (S.SIN1) for normal processing under the System Monitor. The result of this processing, which is contained on the system output unit (S.SOU1), can be handled in either of the following ways: (1) it can be transferred to disk storage for subsequent transmission to a terminal, or (2) it can be printed on an off-line printer and mailed to the terminal. Additional information regarding batch processing will appear in a subsequent publication.

Command Mode

When no program is active at a given terminal, that terminal is in the command mode; and, conversely, when a user enters a COMMAND statement, he will destroy the active image of his program. Since no program can be active at the terminal and statements cannot be retained, they must be processed immediately. Consequently, the user may employ only the general operating statements, the program defining statements, or a limited form of the arithmetic-assignment statement. This latter provision allows the terminal to be used as a fast, versatile symbolic calculator. In this mode, the user enters a statement of the form X=e, where e is any expression consisting of constants and built-in functions, and the system immediately evaluates the expression and prints the result at the user's terminal.

Program Mode

When a program is active at a given terminal (see "Program Defining Statements"), that terminal is in the program mode. In this mode, the user enters program statements that make up the substance of his program, and he operates on the program (i.e., modifies, tests, executes, and debugs it) by using operating statements (see "Language").

While the terminal is in the program mode, the user can also enter single statements that are executed immediately but are not retained in storage (see "Process Code cc").

Source Language Debugging

Debugging information is requested by the user and displayed by the system in a form consistent with the source programming language.

Diagnostic Structure

Errors committed by the user may be classified in two broad categories: syntactic and semantic.

Syntactic Errors

Syntactic errors are considered the responsibility of the system and are further categorized as follows:

Composition: Typographical errors, violations of specified forms of statements and misuse of variable names (e.g., incorrect punctuation, mixed-mode expressions, undeclared arrays, etc.).

Consistency: Statements that are correctly composed but conflict with other statements (e.g., conflicting declaratives, illegal statement ending a DO range, failure to follow each transfer statement with a numbered statement, etc.).

Completeness: Programs that are incomplete (e.g., transfers to nonexistent statement numbers, improper DO nesting, illegal transfer into the range of a DO loop, etc.).

Errors of composition and consistency are detected as soon as the user enters the offending statement. He may immediately substitute a correct statement.

Errors of completeness are discovered when the user signifies that his program is complete (by entering the END statement). Some errors (e.g., invalid subscript value, reference to an undefined variable, arithmetic spills, etc.) can, of course, be detected only during execution. In this case, after a display of the error condition and its location, execution is interrupted and the terminal reverts to READY status. The user then has the option of either immediately correcting his error or proceeding with the rest of his program.

For all syntactic errors, the diagnostic message is specific (in that the variable in error is named, or the column where the error occurred is specified) and often tutorial in suggesting the procedure for obtaining correct results.

Semantic Errors

Semantic errors are concerned with the meaning or intent of the programmer and are definitely his responsibility. However, he is provided with an extensive set of debugging aids for manipulating and referencing a program when in search of errors in logic and analysis.

Value Manipulation

Some types of program statements are also useful for manipulating the values of a user's program. When the terminal is in the program mode, the user may insert special characters, called "process codes," into the first two columns so that these statements can be used as commands. For example, CC preceding a statement has the following effect: the statement is immediately executed with all the effects of normal execution, but no new variable names are created; the statement is then discarded and does not become a part of the program. Thus, the user may insert values into parameters at any time, thereby creating completely new testing situations without having to build their presence into the logic of the program or attempting to anticipate the debugging operations required.

Debugging Statements

The operating statements (see "Language") provide a wide and flexible variety of methods for manipulating the program itself. The user may:

1. Insert or delete statements.
2. Execute selectivity.
3. Print changes of values as the change occurs and transfer control as the transfer occurs.
4. Obtain a static printout of all cross-reference relationships among names and labels, and dynamic exposure of impartial or imperfect execution.

Part 2. Equipment for Remote Computing

The equipment required for use with the Remote Computing System is divided into two groups. The bulk of the equipment is located at a computing center; the remainder is located at each terminal. The equipment at the terminal will be described in detail because the user of the Remote Computing System operates from a terminal.

Computing Center Equipment

The computing center requires the following equipment:

1. An IBM 7040/7044 Data Processing System with 32K words of core storage.
2. An IBM 1301 Disk Storage Unit to be used for permanently retaining users' programs.
3. An IBM 7320 Drum Storage Unit to be used for temporary storage of users' programs.
4. Six magnetic tape units to be used for maintaining normal computer capability and for logging system transactions.
5. An IBM 7740 Communication Control System to be used for sending and receiving data.
6. Two IBM 1311 Disk Storage Drives, one Model 5 and one Model 2, connected to the IBM 7740 for use during batch processing.

The computing center also requires a communications device for connecting terminals to the computer.

Terminal Equipment

In addition to a communications device for connecting the terminal to the computer, each terminal consists of an IBM 1050 Data Communications System terminal equipped with an IBM 1051 Control Unit and an IBM 1052 Printer-Keyboard.

Optional devices that may be installed as part of a terminal are the IBM 1056 Card Reader with pack feed and the IBM 1057 Card Punch.

The User's Terminal Console

In use, the 1050 terminal console appears to be a self-sufficient "FORTRAN machine." The user is completely unaware of any assembly system or the internal organization of the central computer. The language is upwardly compatible with most versions of FORTRAN IV and is augmented by a set of operating, testing, and debugging statements. The basic unit of input to the Remote Computing System is a single statement; for conventional processors, the basic unit of input is an entire program. Since the user receives a reply from the system after he enters each statement, the mode of processing is called "conversational." The conventional mode of processing is called "batch."

Description of the IBM 1052 Printer-Keyboard

The printer portion of the 1052 Printer-Keyboard consists of a platen that controls the paper, a print element, and a carrier that positions the print element. Continuous-form paper is fed into the platen and printing is done by a ball-shaped print element that is moved horizontally to the desired position on the line. The carrier is the mechanism that moves the print element horizontally.

The switch panel portion of the 1052 Printer-Keyboard contains lights, switches, and an indicator that shows the position of the print element.

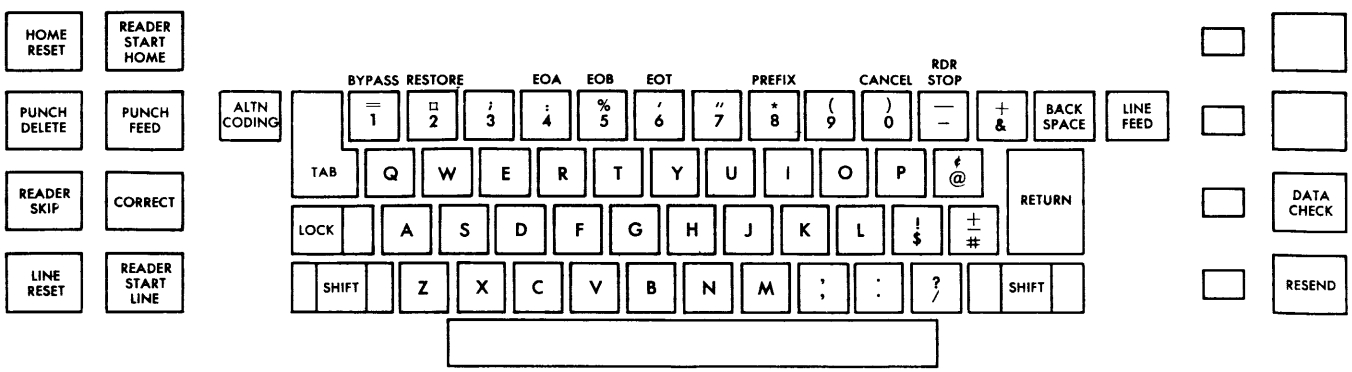


Figure 1. IBM 1052 Keyboard

The keyboard portion of the 1052 Printer-Keyboard (see Figure 1) consists of character keys and control keys. Control keys are located at both sides of the keyboard. The character keys, located between the groups of control keys, are arranged like those on a typewriter. In addition to the normal function of the keys, some of the keys in the top row have other functions (marked directly above the key). When one of these functions is required, the key marked ALTN CODING must be *held down* while the key for the desired function is pressed.

Description of the IBM 1056 Card Reader

The Remote Computing System allows for the use of an IBM 1056 Card Reader equipped with the 80-column Pack Feed feature and the extended character set.

All cards to be read by the 1056 must have an upper-left corner cut and must be fed into the 1056 face down with the column 1 edge first. If a card is fed incorrectly, EJECT can be pressed to cause the card to pass through the 1056 without being read.

Description of the IBM 1057 Card Punch

The IBM 1057 Card Punch is prepared for operation with the Remote Computing System as follows:

1. Place a quantity of blank cards into the feed hopper.
2. Set the Main-Line switch to ON.
3. Set AUTO FEED to ON.
4. Set AUTO SKIP, AUTO DUP to the off position.
5. Set AUTO PUNCH, KEY PUNCH to AUTO PUNCH.
6. Press the Feed key twice.

Note that a program card containing the letter A in all columns must be mounted on the program drum, and the star wheels lowered to read the card.

Initial Setup of the Terminal

To use the Remote Computing System, it is necessary to set up the terminal and then connect it to the computer at the computing center. The setup procedures are given in detail below and are summarized in the upper half of Figure 2.

1. Set the Line Control switch on the CE panel to the ON position. The CE panel is located inside the rear of the IBM 1051 Control Unit.

2. Set the Main-Line switch to the POWER ON position. This turns on the Power light on the 1052 switch panel.

3. Set the switches on the 1052 switch panel as shown below. The positions of the switches are considered to be numbered from left to right. The switches marked * are optional and may not be present on the switch panel.

SWITCH POSITION	SWITCH NAME	SWITCH SETTING
1	SYSTEM (ATTEND-UNATTEND)	ATTEND
2	* MASTER	OFF
3	PRINTER 1	SEND RCV
4	* PRINTER 2	RCV
5	KEYBOARD	SEND
6	* RDR 1	OFF
7	* RDR 2	OFF
8	* PUNCH 1	OFF
9	* PUNCH 2	OFF
10	STOP CODE	OFF
11	* AUTO FILL	OFF
12	* PUNCH	NORM
13	SYSTEM (PROG-DUP)	DUP
14	No switch is assigned to this position.	
15	SYSTEM (DIAL DISC)	In up position
16	TEST	OFF
17	SGL CYCLE	OFF
18	RDR STP	OFF

4. Set the margin stops to provide a line length of at least 85 characters. The first twelve print positions of each line are reserved for use by the Remote Computing System. The information printed in these positions is explained in Part 3.

Print position 13 is the first one available to the user and may be regarded as if it were the first column of the FORTRAN coding form, Form X28-7327. Throughout this publication, all entries to be made by the user will be described using the column numbers of the FORTRAN coding form. Whenever the Remote Computing System pauses for a response from the user, the carrier will be positioned at column 1. The user will find it convenient to set a tab stop at print position 19 so that, when a FORTRAN statement is to be entered, he can press TAB to move to column 7.

To set the margin stops, use the space bar to position the print element at the approximate center of the line. Set the Main-Line switch to POWER OFF. Lift the top cover and tilt the hinged switch panel toward the keyboard. Position each margin stop individually by pressing a blue margin-indicator toward the keyboard and sliding it to the desired position. Lower the top cover and set the Main-Line switch to POWER ON.

To set the tab stop, set both the Printer 1 switch and the Keyboard switch (on the 1052 switch panel in positions 3 and 5) to the HOME position. (Clear any tab stops that are set in the first 18 positions before proceeding. The method of clearing a tab stop is given below.) Press RETURN and then press the space bar 18 times to move the carrier to column 7. Press down the CLR-SET lever at the left side of the 1052 switch panel. The tab stop is now set for column 7. Set the Printer 1 switch to SEND RCV and the Keyboard switch to SEND.

SYSTEM		MASTER		PRINTER 1		PRINTER 2		KEYBOARD		RDR 1		RDR 2		PUNCH 1		PUNCH 2	
Attend	X	Norm	X	RCV		RCV	X	Send	X	Send		Send		RCV		RCV	
		Off		Send RCV	X	Send RCV		Off		Off	X	Off	X	Off	X	Off	X
Un-attend		Master		Home / Send		Home / Send		Home /		Home		Home		Home		Home	

STOP CODE		AUTO FILL		PUNCH		SYSTEM			SYSTEM		TEST		SGL CYCLE		RDR STP	
Sense		On		Norm	X	Prog				X	On		Line		Line	
Off	X	Off	X	Bksp		Dup	X		Dial Disc		Off	X	Off	X	Off	X
													Home		Home	

Setup Procedure

1. Set the Line Control switch on the CE panel to ON.
2. Set the Main-Line switch to POWER ON.
3. Set the switches on the 1052 switch panel as shown above.
4. Set the margin stops for maximum line length and set a tab stop at printing position 19.
5. Press DATA CHECK.
6. Press RESET LINE.
7. Connect the terminal to the computer.

Operating Instructions

Step	Keyboard Operation
1	Wait for the system to type a line number and a status word.
2	Wait for the Proceed light to go on.
3	Type one statement.
4	Hold down ALTN CODING and press EOB; this will cause the Proceed light to be turned off.
5	Repeat steps 1 through 4 for each statement to be entered.

Step	Card Reader Operation
1	Wait for the system to type a line number and a status word.
2	Wait for the Proceed light to go on.
3	Set Keyboard switch to OFF.
4	Set RDR 1 switch to SEND.
5	Press READER START LINE.
6	Insert a card into the reader until a click is heard.
7	Wait until the card is read and ejected.
8	Wait for the system to type a line number and a status word.
9	Repeat steps 6 through 8 until all cards have been read.
10	To return to keyboard operation, set RDR 1 switch to OFF and Keyboard switch to SEND.

Figure 2. Operating Procedures for Remote Computing

To clear a tab stop, press RETURN and then press TAB to position the carrier at the tab stop to be cleared. Lift up the CLR-SET lever on the 1052 control panel. When clearing tab stops, the user must set the switches on the 1052 control panel as explained above for setting tab stops.

5. Press DATA CHECK.

6. Press RESET LINE (RESET on some models). The Proceed light should now be off and the keyboard should be locked (i.e., no typing can be done). If the Proceed light is on, check the settings of the switches, especially the Line Control switch.

Terminal-Computer Connection

The final step in the initial setup of the terminal is its connection to the computer by the use of a communications device. Instructions for using the device are provided by the communications company that installs it.

Terminal Operating Procedures

After the connection has been made, the Remote Computing System will send a message to the terminal. After the printing of this message, the Proceed light

will be turned on and the keyboard will be unlocked. The Remote Computing System is now ready to accept the first entry from either the keyboard or the card reader. The operating instructions for both units are given below and are summarized in the lower half of Figure 2.

During terminal operation, input may be entered from the keyboard, from the card reader, or from both. The user may alternate between the input units, as required, by following the operating instructions given under "Procedures for Keyboard Input" and "Procedures for Card Input."

Procedures for Keyboard Input

The character keys are used to type the letters, numbers, and special characters that make up the keyboard input. The line feed and carrier return functions are provided by the Remote Computing System; therefore, the user must not press LINE FEED or RETURN. The functions of other keys are:

KEY NAME	FUNCTION
SHIFT	This key must be held down while pressing the appropriate keys to type the following characters: = *) (+ ' No other characters should be entered while SHIFT is held down. The shift lock key should *not* be used. (See "Keyboard Time-Out" for an additional use of SHIFT.)
TAB	This key moves the carrier to the next tab stop. Normally, for the Remote Computing System, a tab stop is set for column 7 only.
BACK SPACE	This key backspaces the carrier. The user can correct errors by typing the correct characters over the wrong ones. As each character is backspaced over, it is deleted; therefore, if correct characters are backspaced over to reach an incorrect one, they have to be retyped after the correction has been made. For example, to change FOEMAT to FORMAT after typing the T, backspace 4 times and type RMAT.

Some of the keyboard functions require that the user press two keys at the same time.

KEY NAME	FUNCTION
ALTN CODING	This key must be held down while a key that has an alternate coding function is pressed. (For example, EOB is obtained by holding down ALTN CODING and pressing 5.)
EOB	This key indicates the end of a line of input information. It must be pressed (while ALTN CODING is held down) at the end of every line to return control to the Remote Computing System.
CANCEL •	This key deletes the entire line currently being entered. The ALTN CODING key must be held down when CANCEL is pressed; immediately after pressing CANCEL, press EOB (and ALTN CODING).

The Remote Computing System does not use the alternate coding function of any other keys.

The procedure for entering information from the keyboard is:

1. The Remote Computing System will print a line number and a status word (e.g., READY); printing will stop with the carrier at column 1.
2. Wait for the Proceed light to be turned on, indicating that the system is ready to accept input information.
3. Type one line of input (e.g., a single statement, with or without a statement number).
4. Hold down ALTN CODING and press EOB to indicate the end of an entry and to return control to the Remote Computing System. The Proceed light will be turned off when this operation occurs.
5. Return to step 1 to continue operating. To switch to card input, follow the procedure given under "Instructions for Card Input."

Keyboard Time-Out

Keyboards are equipped with a "time-out" feature, which causes the keyboard to lock if more than 15 seconds elapse between the sending of characters. If a time-out occurs during the typing of a line, any information typed on that line up to that point will be discarded. To avoid this loss of information, the user may press and release SHIFT before the 15-second limit has been reached; this action may be repeated as necessary to prevent a time-out. By pressing SHIFT, the user can prevent a time-out without affecting the input information.

Procedures for Card Input

The procedure for entering information from the card reader is:

1. The Remote Computing System will print a line number and a status word (e.g., READY) and the carrier will stop at column 1.
2. Wait for the Proceed light to be turned on, indicating that the system is ready to accept input information.
3. Set the Keyboard switch to OFF.
4. Set the RDR 1 switch to SEND.
5. Press READER START LINE.
6. Insert a card into the card reader until a click is heard. The card (with an upper-left corner cut) must be inserted into the card reader face down with the column 1 edge first.
7. The card will be read and ejected.
8. Wait until the Remote Computing System prints a line number and a status word. If there is an error in the card, the keyboard may be used to correct the error before the next card is read; skip to step 10.

9. Repeat steps 6 through 8 until all cards have been read.

10. To return to keyboard operation, set the RDR 1 switch to OFF and the keyboard switch to SEND. Then follow the procedures given under "Instructions for Keyboard Input."

Regaining Control

The user can regain control whenever the Proceed light is on. Whether or not the Proceed light is turned on periodically is determined by the setting of the Keyboard switch. If the switch is set to OFF, the Proceed light will not be turned on. If the Keyboard switch is set to SEND, the Proceed light will be turned on periodically for about 15 seconds.

To regain control, wait until the Proceed light is turned on and then type the word HALT beginning in column 7. Next, hold down ALTN CODING and press EOB. A message will indicate that control has been returned to the terminal.

If the user does not want to regain control, he can avoid waiting 15 seconds by holding down ALTN CODING and pressing EOB as soon as the Proceed light is turned on.

End of Terminal Operation

The user should disconnect the terminal from the computer after he has finished using the terminal or after the computing center has indicated that the Remote Computing System has been discontinued for the day. The user may disconnect the terminal in either of the following ways:

1. Press down on the SYSTEM (DIAL DISC) switch.
2. Set the Main-Line switch to POWER OFF.

Basic Information

This section describes the form in which statements are entered and explains the information produced by the Remote Computing System in response to these statements.

Entry Format

The user enters statements by typing them on the 1052 Printer-Keyboard. There is no printed coding form; however, the user may use a FORTRAN coding form, Form X28-7327, for planning entries he must make. Data printed by the system will be described according to the print position(s) in which it appears; data entered by the user will be described according to the columns of the FORTRAN coding form. Print positions 1 through 12 are reserved for the following uses:

Print Positions 1 through 5

The Remote Computing System assigns a line number to each statement entered; these numbers are printed (by the system) in print positions 1 through 5. Numbers are assigned starting with 101 and increased by 1 up to a maximum of 999. The tenths position allows the user to insert additional statements where necessary. With line numbers, the user can refer to a point in a program at which he may add or delete statements, begin execution, start or end a testing operation, etc.

The Remote Computing System prints line numbers to identify data for the user. For example, when a program is being executed, output data is accompanied by the line number of the output statement that produced the data. The results of some test statements are the line numbers of the statements that are involved with the test being made. For example, when checking branching statements, the Remote Computing System will print the line numbers of the statement from where a branch originated and the ones to which it transferred.

The Remote Computing System keeps track of the line numbers assigned and deleted, and prints the next available line number when ready to accept another statement. Therefore, whenever the user executes a statement, a program, or part of a program, he need not remember the line number of the last entry, because the system will return to the proper point after the action requested by the user has been completed.

Print Position 6

The Remote Computing System prints a status indicator in print position 6. There are three indicators which are described under "Status Indicators."

Print Positions 7 through 11

The Remote Computing System prints a status word in these positions. Various status words inform the user of the action taken by the system, the kind of output produced, etc. They are described under "Status Words."

Print Position 12

This print position will always be blank. This is the last position reserved for use by the Remote Computing System.

Columns 1 through 72

Column 1, which corresponds to print position 13, is the first position available to the user for entering statements. Although any alphabetic character in column 1 will cause the line to be treated as a comment, the user should observe the conventions explained under "Comment Codes."

Columns 1 through 5 may be used for statement numbers, which must be within the range of 1 and 999. The user need number only those statements that are to be referred to in his program.

Column 6 must always be blank.

Columns 7 through 72 may be used for any of the statements listed under "Language." As explained in Part 2, it is convenient to have a tab stop set for column 7.

Status Indicators

Immediately preceding a status word, the Remote Computing System prints one of the following status indicators:

\+ − =

The + sign indicates that the system is in program mode and is awaiting a response from the terminal. The − sign indicates that the system is in command mode and is awaiting a response from the terminal.

The = sign indicates that the system is in automatic status: that is, it has been executing a program, and one of five conditions has occurred. These conditions are:

1. An error has occurred during program execution, and an error message has been typed. For example:

```
126. = ERROR VARIABLE MAY NOT BE USED UN-
       TIL SET.
```

2. A response to a debugging statement has occurred and has been typed. For example:

```
293. = INDEX SAM 103. + 195. — 196.
```

3. A request for data by an input statement has occurred and the system is waiting for the data to be typed in or read in. For example:

```
456. = I23.
```

4. A response to an output statement has occurred and the data output has been typed. For example:

```
322. = 0122     X     Y
```

5. A programmed pause, stop, or end of execution has occurred. For example, if the statement STOP 77 was executed and its line number was 592.0, the following would be printed:

```
592. = S77
```

Status Words

Print positions 7 through 11 are reserved for status words, which are printed by the system. These words either indicate something about the status of the system or indicate that the system is making response to, or diagnosis of, a statement that has been entered.

The majority of status words are responses to program debugging statements and are described under the corresponding debugging statement. The remaining status words, described below, are READY, ERROR, and CANCL.

Status Word READY: This status word indicates that the Remote Computing System is waiting for a statement entry from the terminal.

Status Word ERROR: This status word indicates that the Remote Computing System has detected an error. The various types of errors that can occur are described in Part 1 under "Source Language Debugging."

Status Word CANCL: This status word indicates that the Remote Computing System has deleted some information. The status word CANCL will be printed when:

1. The user has pressed the Cancel key (followed by EOB) at the terminal. This action causes the Remote Computing System to cancel the statement just entered at the terminal. After CANCL has been printed, the user may enter a new statement in response to the READY message on the next line.

2. The user has allowed too much time to elapse between the typing of characters (see "Keyboard Time-Out" in Part 2). The Remote Computing System will cancel the part of a statement that has been entered. After CANCL has been printed, the user may enter a statement in response to the READY message on the next line.

Comment Codes

Various types of comments can be entered from a terminal when it is in the program mode. Each type of comment is identified by a different comment code. The user types the desired comment code in columns 1 and 2, and then types the comment in the remaining positions of the line. The three comment codes are cb, CV, and CF.

Comment Code Cb

This code indicates that the remainder of the line is a comment. If a program is active for the terminal when this comment code is entered, the comment will be retained as part of the active program. (The b indicates a blank.)

Comment Code CV

This code indicates that the remainder of the line is a comment. The Remote Computing System does not retain comments identified with this comment code. Comment Code CV can be used to print comments that are useful during program construction, but that need not be part of the active program.

Comment Code CF

This code is used to keep programs for the Remote Computing System compatible with source programs for other FORTRAN processors. The Remote Computing System will process statements beginning with comment code CF as though the CF were not there. Other FORTRAN processors will regard those statements as comments.

The Remote Computing System will automatically supply process code CF for all program statements that are not valid FORTRAN IV statements.

Process Code CC

When the terminal is in the program mode, the user can execute a single statement that he does not want included in the active program. By typing process code CC in columns 1 and 2 and an arithmetic assignment statement or an output statement in columns 7 through 72, the user indicates to the system that the statement is to be executed immediately, but not retained as part

of the program. After the statement has been executed, the results are printed at the terminal.

Process code CC is useful when testing and debugging programs. The values of variables within a program can be assigned as required. To assign a value, the name of the variable must appear to the left of the equal sign in an assignment statement that follows process code CC. All values necessary to test a program can be assigned by this method. The expression may include constants, reserved functions (e.g., COS, ATAN), and other variables in the program. However, if variables are included, they must have values that were assigned either during program execution or during the execution of assignment statements of previous CC entries.

Similarly, process code CC can be used to display selectively the values of variables in a program. The user can display variables by listing their names in an output statement following process code CC. For example:

```
CC    PRINT 5, TEMP, VOL, PRES
```

Language

The language of the Remote Computing System comprises two types of statements: program statements and operating statements. Program statements, which are upwardly compatible with FORTRAN IV (see Appendix), are used to form the user's program. Operating statements allow the user to communicate with the Remote Computing System. Operating statements include modification, test, display, and output statements.

The statements that are available for use in constructing a program are described under "Program Statements." Statements that are available for changing, testing, and executing programs are described under "Operating Statements."

Program Statements

The user's program is made up of program statements. When entered from a terminal, these statements are always retained in storage as part of the active program.

If the user has a statement in his program that refers to an executable program statement within the program, he should assign a statement number to the statement referred to. Numbers 1 through 999 may be used as statement numbers, with no two statements having the same number. The statements acceptable to the Remote Computing System are described below. The description gives the general form of each statement, its purpose, and one or more examples of its use.

The following table gives the meanings of the symbols used to identify the variables in the general forms of the statements:

SYMBOL	TYPE OF VARIABLE
a	Array name
c	Constant
d	Identifier (constant, simple variable, array name, or function name)
e	Arithmetic expression
f	Function name
i	Simple integer variable (unsigned)
k	Simple integer constant (unsigned)
m	Simple integer variable (unsigned) or simple integer constant (unsigned)
n	Statement number
p	Parameter
u	Name (program or subprogram)
v	Variable (simple or subscripted)
x	Simple variable
y	Subscripted variable
z	Name (array name, function name, or simple variable)

Constants

When used in computations, a constant is any number that does not change from one execution of the program to the next. It appears in its actual numerical form in the statement. For example, in the following statement, 3 is a constant since it appears in actual numerical form:

```
J=3*K
```

Two types of constants may be written: integer constants and real constants (characterized by being written with a decimal point). The rules for writing each of these constants are given below.

Integer Constants

An integer constant is written without a decimal point, using the decimal digits 0,1, ...,9. A preceding + or − sign is optional. An unsigned integer constant is assumed to be positive.

Real Constants

A real constant is written with a decimal point, using the decimal digits 0,1, ...,9. A preceding + or − sign is optional. An unsigned real constant is assumed to be positive.

An integer exponent preceded by an E may follow a real constant. The exponent may have a preceding + or − sign. An unsigned exponent is assumed to be positive.

Variables

A variable is a symbolic representation (name) that will assume a value. This value may change either for different executions of the program or at different stages within the program. For example, in the following statement, both I and K are variables:

```
K=3*I
```

The value of I will be assigned by a preceding statement and may change from time to time, and the value of K will vary whenever this computation is performed with a new value of I.

As with constants, a variable may be integer or real, depending on whether the value which it will represent is to be integer or real, respectively.

In order to distinguish between variables which will derive their value from an integer as opposed to those which will derive their value from a real number, the rules for naming each type of variable are different.

Integer Variables

An integer variable consists of a series of not more than six alphameric characters (except special characters), of which the first is I, J, K, L, M, or N.

Real Variables

A real variable consists of a series of not more than six alphameric characters (except special characters), of which the first is alphabetic but not one of the integer indicators, i.e., I, J, K, L, M, or N.

Subscripts

An *array* is a group of quantities. It is often advantageous to be able to refer to this group by one name and to refer to each individual quantity in this group in terms of its place in the group.

For example, assume the following is an array named NEXT:

15
12
18
42
19

Suppose it is desired to refer to the second quantity in the group; in ordinary mathematical notation this would be $NEXT_2$. In FORTRAN this would be:

NEXT (2)

The quantity "2" is called a subscript. Thus:

NEXT (2) has the value 12
NEXT (4) has the value 42

Similarly, ordinary mathematical notation might use $NEXT_i$ to represent any element of the set NEXT. In a program statement, this is written as NEXT (I) where I equals 1, 2, 3, 4, or 5.

The array could be two dimensional; for example, the array named MAX:

	COLUMN 1	COLUMN 2	COLUMN 3
Row 1	82	4	7
Row 2	12	13	14
Row 3	91	1	31
Row 4	24	16	10
Row 5	2	8	2

Suppose it is desired to refer to the number in row 2, column 3; this would be:

MAX(2, 3)

"2" and "3" are the subscripts. Thus:

MAX(2, 3) has the value 14
MAX(4, 1) has the value 24

Similarly, ordinary mathematical notations might use $MAX_{i,j}$ to represent any element of the set MAX. In a program statement, this is written as MAX(I, J) where I equals 1, 2, 3, 4, or 5 and J equals 1, 2, or 3. In this system, the above notation may be extended to three-dimensional arrays.

Form of Subscripts

A subscript must be in one of the following forms only, where I represents an unsigned non-subscripted integer variable, and K1 and K2 an unsigned integer constant:

I
K
I+K1 or I−K1
K1*I
K1*I+K2 or K1*I−K2

Subscripted Variables

A subscripted variable is either an integer or real variable, followed by parentheses enclosing one, two, or three subscripts separated by commas.

Assignment Statements

General Form
v = e

These statements assign a value to a specified variable; i.e., the value of an expression, e, is determined and assigned to a variable, v. The = symbol in an assignment statement signifies the replacement or assignment of a value to a variable, rather than mathematical equality. Therefore, such statements as V=V+1 are acceptable assignment statements.

When there is a difference in mode between the variable to the left of the equal sign and the expression to the right of the equal sign, the following conventions apply:

1. If the expression is in the real mode, its result is truncated (not rounded) to the largest integer it contains and then converted to an integer value before it replaces the correct value of the integer variable.

2. If the expression is in the integer mode, the result is converted to a real value before it replaces the current value of the real variable.

Control Statements

Statements are executed in the order in which they appear in the program. Control statements may be used to cause a conditional or unconditional change in the sequence of execution.

A control statement must not refer to itself. The next executable statement following a GO TO or IF statement must be numbered.

Unconditional GO TO Statement

General Form
GO TO n

EXAMPLE:

```
GO TO 3
```

This statement causes the program to transfer control to statement n.

Computed GO TO Statement

General Form
GO TO (n_1, n_2, ..., n_p), i

EXAMPLE:

```
GO TO (30, 42, 50, 9),J
```

This statement causes transfer of control to the first, second, etc., statement number in the list n_1, n_2, . . . n_p, depending on whether the value of i is 1, 2, . . . , p, respectively. The value of i must be equal to or greater than 1 and equal to or less than p.

IF Statement

General Form
IF (e) n_1, n_2, n_3

EXAMPLES:

```
IF (X-Y) 10, 5, 3
IF (I-3) 2, 7, 4
```

This statement causes transfer of control to statement n_1, n_2, or n_3, depending on whether e is less than, equal to, or greater than zero, respectively.

DO Statement

General Form
DO n i = m_1, m_2, m_3 m_1 must be equal to or less than m_2. m_3 is optional but, if stated, m_3 must be equal to or greater than 1; if m_3 is not stated, it is assumed to be equal to 1.

EXAMPLES:

```
DO 25 J=1, M, 2
DO 35 J=1, 10
```

This statement is a command to execute repeatedly the statements that follow, up to and including statement n, which must not be a DO, IF, GO TO, STOP, or RETURN statement; a CONTINUE statement is usually used to satisfy this limitation. The "range of a DO" comprises all statements between the DO statement and statement n inclusive.

The first time the statements are executed, i is equal to m_1. For each succeeding execution, i is increased by m_3. After the statements have been executed with i equal to the highest value that does not exceed m_2, control passes to the statement following statement n. The values of i, m_1, m_2, and m_3 must not be redefined (i.e., receive a new value) by any statement within the range of a DO. Similarly, subroutines called from within the range of a DO must not change the values of i, m_1, m_2, and m_3, because such subroutines are considered to be within the range of the DO.

The use of a DO within the range of another DO is called "nesting." When a DO statement appears within the range of another DO, the range of the inner DO must be completely contained within the range of the outer DO. The nesting of DO statements must not exceed a depth of eight levels.

No transfer into the range of any DO from a point outside its range is allowed. However, transfers caused by RETURN statements in subprograms are permitted.

CONTINUE Statement

General Form
CONTINUE

EXAMPLE:

```
CONTINUE
```

This statement is a dummy statement that causes no action. It is most frequently used as the last statement in the range of a DO.

PAUSE Statement

General Form
PAUSE d d is an optional 1-digit through 5-digit octal number, the last two digits of which are to be printed at the terminal when the statement is executed.

EXAMPLE:

```
PAUSE 3
```

This statement causes a halt in the execution of the program. Execution can be resumed with the next executable statement.

STOP Statement

General Form
STOP d d is an optional 1-digit through 5-digit octal number, the last two digits of which are to be printed at the terminal when the statement is executed.

EXAMPLE:

```
STOP 77777
```

This statement causes the program to halt. Execution cannot be resumed. The next executable statement must be numbered.

Declarative Statements

These statements provide for the allocation of storage

for arrays and for the sharing of storage locations between programs or within a program. Declarative statements are not executable. They must precede the first executable program statement. However, FORMAT statements may appear anywhere in a program. Except for FORMAT statements, declarative statements must not have statement numbers. Constants, program names, function names, and subroutine names must not appear in declarative statements. Except when used in EQUIVALENCE statements, a variable name must not appear in more than one declarative statement of the same type.

DIMENSION Statement

General Form
DIMENSION $y_1, y_2, ..., y_p$

Each subscripted variable has 1, 2, or 3 subscripts that give the maximum size of each dimension of the array being specified. The maximum size of each array is 3000.

EXAMPLES:

```
DIMENSION A(10), B(5, 5, 5)
DIMENSION J(12, 3), E(5)
```

This statement, if used, must be the first declarative statement in a program. It provides the information necessary to allocate storage for arrays in the program. The name of each array must appear in a DIMENSION statement before appearing in any other statement.

COMMON Statement

General Form
COMMON $v_1, v_2, ..., v_p$

If the variable is the name of an array, it must not include dimension information.

EXAMPLES:

```
COMMON A, B, C, D, E
COMMON X, ANGLE, MATA, MATB
```

This statement causes the variables to be assigned to consecutive storage locations within a common storage area. Programs and subprograms can share the common storage area.

EQUIVALENCE Statement

General Form
EQUIVALENCE $(v_1, v_2, ..., v_p)$ p must be equal to or greater than 2. All variables must be enclosed in one set of parentheses. If v_1 is an array name, it may have a single subscript that specifies the relative position within the array of the variable being made equivalent to v_2 through v_p. If v_1 is a simple variable, it must not have a subscript; v_2 through v_p must never have subscripts.

EXAMPLES:

```
EQUIVALENCE (F (3), E)
EQUIVALENCE (J (3), I, A, D)
```

This statement causes all of the variables specified to be assigned to the same location in storage. An actual storage area is reserved only for the first variable, v_1; variables v_2 through v_p overlay the area reserved for v_1 (they do not have individual storage areas). A variable may appear in more than one EQUIVALENCE statement only if it appears as v_1 in each of them.

Type Declaration Statements

These statements provide the means to define the mode of a variable or function name, normally defined by the first letter of the name. The name of a variable must not appear in more than one type declaration statement. The name of a function subprogram may appear in an EXTERNAL and an INTEGER statement or in an EXTERNAL and a REAL statement. Constants and reserved names must not appear in type declaration statements.

INTEGER Statement

General Form
INTEGER $z_1, z_2, ..., z_p$

EXAMPLE:

```
INTEGER RESULT, DETER, GAMMA
```

This statement specifies that the listed names are integer variables.

REAL Statement

General Form
REAL $z_1, z_2, ..., z_p$

EXAMPLE:

```
REAL MASS, MATRIX, MILNE
```

This statement specifies that the listed names are real variables.

EXTERNAL Statement

General Form
EXTERNAL $u_1, u_2, ..., u_p$

EXAMPLE:

```
EXTERNAL PLOT, POLAR, BAR
```

This statement specifies that the listed names are nonreserved subprogram names. All nonreserved subprogram names used as arguments in a program must appear in an EXTERNAL statement within that program. Variable names and reserved names; e.g., SIN, must not appear in an EXTERNAL statement.

Input/Output Statements

These statements provide for the transmission of data between core storage and the terminal. To simplify the description of formats, references will be made to a printed line; however, the information also applies to card and tape records.

Most of the input/output statements include a parameter(m) that, in conventional FORTRAN systems, specifies the input or output unit to be used for a statement. The Remote Computing System does not use the m; it is included for compatibility with conventional FORTRAN processors. When an input statement is executed, the system expects the input data to be entered from the keyboard. When an output statement is executed, the system prints the output data on the printer. However, the user may designate a different input and/or output unit by using a SELECT statement (see SELECT Statement"). Regardless of which input/output units are selected, all input and output will be monitored on the printer.

If a program is compiled using a conventional FORTRAN processor at a later time, the m will specify the input or output unit to be used by the statement.

FORMAT Statement

General Form
nFORMAT (specifications) specifications are as explained below.

EXAMPLES:

```
24 FORMAT (I2/(E12.4, F10.4))
22 FORMAT (I10)
26 FORMAT (E15.6, F10.6/5I10)
```

This statement describes the type of conversion and the format to be used in the transmission of data. Each FORMAT statement must have a statement number.

Conversion of Numeric Data: The following table shows the three types of conversion for numeric data:

INTERNAL	CONVERSION CODE	EXTERNAL
Floating Point	E	Floating Point (with exponent)
Floating Point	F	Floating Point (without exponent)
Integer	I	Integer

Numbers printed by E-type conversion are printed as a decimal fraction times a power of 10. These numbers are normalized; that is, their first significant digit is to the right of the decimal point. For example:

```
233.3   may be printed as   0.2333E 03
.003    may be printed as   0.30E−02
17.4    may be printed as   0.174E 02
```

Numbers printed by F-type conversion are printed in a "normal" fashion; that is, they appear as output in a meaningful decimal notation without an exponent. Typical output might be:

```
 12.3    −0.726    102.
−17.2     1.318   −968.
 289.1    0.009    721.
```

Numbers printed by I-type conversion are printed as integers. Typical output might be:

```
  12
 −17
2342
```

These basic numeric field specifications are given in the forms:

Iw Ew. d Fw. d

I, E, and F represent the types of conversion,
w represents the total number of print positions for the converted data, and
d represents the number of decimal places to the right of the decimal point.

The decimal point between the w and d portions of the specifications is required punctuation.

I-Conversion: The specification Iw may be used to print a number that exists in the computer as an integer quantity; w print positions are reserved for the number. It is printed in this w-space field right-justified (that is, the units position is at the extreme right). If the number converted is greater than w digits, the leftmost digits are lost; no rounding occurs. If the number has less than w digits, the leftmost spaces are filled in with blanks. If the quantity is negative, the space preceding the leftmost digit will contain a minus sign if sufficient spaces have been reserved.

The following examples show how each of the internal quantities is printed according to the specification I3:

INTERNAL	PRINTED	
721	721	
−721	721	*
−12	−12	
9	bb9	
8114	114	*
0	bb0	
−5	b−5	

* Inaccurate due to insufficient specification
(b is used here to indicate blanks)

F-Conversion: For F-type conversion, w is the total number of print positions reserved, and d is the number of places to the right of the decimal point (the fractional portion). The fractional portion is truncated from the right if insufficient spaces are reserved; zeros are filled in from the right if excessive spaces are reserved. Within the remainder of the field, the integer portion is handled in much the same fashion as numbers converted by I-type conversion.

A space for the decimal point and a space for the sign must be included in the count, w. (For output, space for at least one digit preceding the decimal point should be reserved.)

The following example shows how each of the inter-

nal quantities is printed according to the specification F5.2:

INTERNAL	PRINTED	
12.17	12.17	
−41.16	41.16	*
−.2	−0.20	
7.3542	b7.35	**
−1.	−1.00	
9.03	b9.03	
187.64	87.64	*

* Inaccurate due to insufficient specification
** Last two digits of accuracy lost due to insufficient specification (b is used here to indicate blanks)

E-Conversion: For E-type conversion, w is the total number of print positions reserved. The fractional portion is indicated by d; w includes the field d, plus two spaces for a sign and the decimal point, plus four spaces for the exponent, plus space for the integer portion. (For output, space for at least one digit preceding the decimal point should be reserved.) The exponent is the power of 10 times which the number must be raised to obtain its true value. The exponent is written with an E followed by a space for a minus sign if the exponent is negative or a plus or blank if the field is positive, and two spaces for the exponent.

The following example shows how each of the internal quantities is printed according to the specification E10.3:

INTERNAL	PRINTED
238.	0.238Eb03
−.002	−0.200E-02
.00000000004	0.400E-10
−21.0057	−0.210Eb02*

* Last three digits of accuracy lost due to insufficient specification (b is used here to indicate blanks)

It is evident from the above examples that the user must know the data in order to specify a satisfactory format. Insufficient format specifications can result in inaccurate output. In general, specifications should provide for the largest quantities to be transmitted and the greatest accuracy desired.

Additional Rules for Specifying Format:

1. Field width may be specified greater than required in order to provide spacing. Thus, if a number to be converted by I-type conversion is not expected to exceed five spaces including a sign, a specification of I10 will reserve a minimum of five leading blanks.

2. A specification may be repeated as many times as desired (within the limits of the output device) by preceding the specification with an unsigned integer constant. Thus, (2F10.4) is equivalent to (F10.4, F10.4).

3. Succeeding specifications may be written in a single FORMAT statement by separating them with commas. Thus, (I2, E10.2) might be used to convert two separate quantities, the first integer and the second floating point.

4. The specifications in a FORMAT statement must correspond in type with an item in the input/output statement; integer quantities require integer conversion, and real quantities require floating-point conversion.

Thus, the following statements are compatible:

```
      PRINT 2, A, B, I
2     FORMAT (2F6. 4, I10)
```

5. Successive items in the input/output list are transmitted by successive corresponding specifications in the FORMAT statement until all items in the list are transmitted. If there are more items in the list than there are specifications, control transfers to the first specification after the preceding left parenthesis of the FORMAT statement.

For example, suppose the following statements are written into a program:

```
      PRINT 10, A, B, C, D, E, F, G
10    FORMAT (F10.3, E12.4, F12.2)
```

The following table shows the variable transmitted and the specification by which it is converted:

VARIABLE TRANSMITTED	SPECIFICATION
A	F10.3
B	E12.4
C	F12.2
D	F10.3
E	E12.4
F	F12.2
G	F10.3

6. Quantities are transmitted to consecutive print positions, starting in print position 19. Quantities transmitted in excess of the print positions will be lost.

7. A limited parenthetical expression is permitted in order to enable repetition of data fields according to certain format specifications within a longer FORMAT statement specification. Thus, FORMAT (2(F10.6, E10.2), I4) is equivalent to FORMAT (F10.6, E10.2, F10.6, E10.2, I4). An additional level of parentheses is not permitted. Thus, FORMAT (2(3(I6, E10.2))) is *not* valid.

Multi-Line Format: To deal with a block of more than one line of print, a FORMAT specification may have several different one-line formats, separated by a slash (/) to indicate the beginning of a new line.

The following statement provides for more than one line:

```
2     FORMAT (3F9.2, 2F10.4/8E14.5)
```

This statement specifies a multi-line block of print in which lines 1, 3, 5, . . . have format (3F9.2, 2F10.4), and lines 2, 4, 6, . . . have format (8E14.5).

If a multi-line format is desired such that the first n lines will be printed according to a special format and all remaining lines according to another format, the last line specification should be enclosed in a second pair of parentheses; for example:

```
5     FORMAT (I2, 3E12.4/2F10.3, 3F9.4/(10F12.4))
```

If there are data items remaining to be transmitted after the format specification has been completely "used," the format repeats from the last left parenthesis.

Unit Record: The discussion so far has been concerned only with printed output. At this point the discussion will be extended to all input/output by introducing the concept of *unit record.* This supplies to those aspects of input/output already discussed as well as those yet to be discussed. Except where noted, all references to printed line also apply to other input/output records.

A unit record may be:

1. A printed line with a maximum of 114 characters.
2. A punched card with a maximum of 72 characters.

(Although the standard 80-column card is used, the last 8 columns are reserved for identifying information and are not usually processed by the Remote Computing System.)

3. A BCD tape record with a maximum of 133 characters. The use of tape records will be discussed below.

For example, a specification may be written for reading data from cards. Such a specification, used in conjunction with a READ statement, instructs the computer regarding the appearance of data in the external medium so that the data may properly be converted and assigned as the values of the variables listed in the input list.

Blank Fields: Blank characters may be provided in an output record, or characters of an input record may be skipped, by means of the specification wX where w is the number of blanks to be provided or characters to be skipped. When the specification is used with an input record, w characters are considered to be blank regardless of what they actually are, and are skipped over.

For example, if a card has six 10-column fields for integers, and it is not desired to read the second quantity, then the following statement may be used along with the appropriate READ statement:

```
21    FORMAT (I10, 10X, 4I10)
```

Alphameric Fields: There are two specifications available for input/output of alphameric information. The specification wH is used for alphameric data that is not going to be processed by the object program; the specification Aw is used for alphameric data that is to be operated upon by the program.

Information handled with the A specification is given a variable or array name so that it can be referred to by this name for processing and/or modification. Information handled with the H specification is not given a name and may not be referred to or manipulated in any way.

H-Conversion: The specification wH is followed in the FORMAT statement by w alphameric characters. For example:

```
24HbTHISbISbALPHAMERICbDATA
```

Note that blanks are considered alphameric characters and must be included as part of the count w. This is the only case (except for column 6) where blanks are not ignored in program statements.

The effect of wH depends on whether it is used with input or output.

1. *Input:* The w characters are extracted from the input record and replace the w characters included with the specification.
2. *Output:* The w characters following the specification (or the characters that replaced them as a result of input operations) are written as part of the output record.

For example, suppose that the following statements are executed:

```
      PRINT 2
2     FORMAT (20HTIME/QUANTITYbREPORT)
```

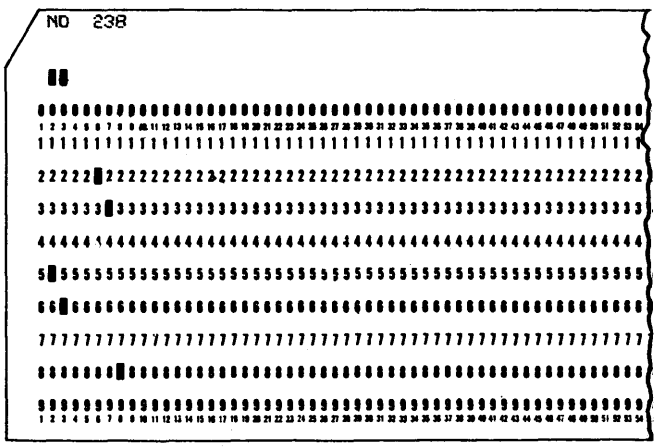
These statements would cause the following output to be printed:

```
TIME/QUANTITY REPORT
```

On the other hand, suppose the following statements are executed:

```
      READ 1, I
1     FORMAT (3HYES, I5)
```

Assume that the following data card is read by these statements:



Next, suppose that the following statement is executed:

```
PRINT 1, I
```

This would cause the following printed output:

```
bNObb238
```

(b is used here to indicate blanks)

A-Conversion: The specification Aw causes w characters to be read into, or written from, a variable or array name. The name must be constructed in the same

manner as an integer or floating-point variable name. For example, suppose that the following statements are executed:

```
PRINT 15, A, B, C, D, E, F
FORMAT (3HXY=, F8.3, A5/)
```

Those statements might produce the following lines:

```
XY=b-93.210bbbbb
XY=9999.999OVFLW
XY=bb28.768bbbbb
```

This example assumes that there are steps in the source program that read the BCD word "OVFLO," store this data in the word to be printed in the format A5 when overflow occurs, and store blanks in the word when overflow does not occur.

PRINT Statement

General Form
PRINT n, list n is the statement number of a FORMAT statement. list specifies the quantities to be transmitted.

EXAMPLES:

```
PRINT 12, A, I, J (3)
PRINT 2, (A(I), I=1, 10,2)
```

This statement causes the items in the list to be written on the currently selected output unit in the format specified by statement n. The number of characters specified in the FORMAT statement must not exceed 114 per line.

PUNCH Statement

General Form
PUNCH n, list n is the statement number of a FORMAT statement. list specifies the quantities to be transmitted.

EXAMPLES:

```
PUNCH 10, A, I, J (3)
PUNCH 12, (A(I), I=1, 10, 3)
```

This statement causes the items in the list to be written on the currently selected output unit in the format specified by statement n. The FORMAT statement must not specify more than 80 characters per record.

READ(m) Statement

General Form
READ(m) list m is the number of an input unit. list specifies the quantities to be transmitted.

EXAMPLES:

```
READ (24) X, Y, Z
READ (16) NUM, TEMP, MASS
An example of data to be read is: 12=98.6/215
```

This statement causes numeric data to be read from the currently selected input unit and to be assigned as the values of the variables. Because there is no FORMAT statement, each value read must be separated from the next value by one of the following characters:

```
+  -  /  =
```

All non-numeric characters, except the following, are deleted (not entered into storage):

```
+  -  .  E
```

If the program is compiled using a regular FORTRAN processor at a later time, the READ(m) statement will read data in binary form.

READ(m, n) Statement

General Form
READ(m,n) list m is the number of an input unit. n is the statement number of a FORMAT statement. list specifies the quantities to be transmitted.

EXAMPLES:

```
READ (24,3) K, A(J)
READ (J,8) JOBNO, X, Y
```

This statement causes data to be read from the currently selected input unit and to be assigned as the values of the variables in the list. The FORMAT statement must not specify more than 114 characters per record.

If the program is compiled using a regular FORTRAN processor at a later time, the input will be read in BCD form.

READ n Statement

General Form
READ n, list n is the statement number of a FORMAT statement. list specifies the quantities to be transmitted.

EXAMPLES:

```
READ 4, A, B, C, D
READ 7, (I(J), J=1, 12)
```

This statement causes data to be read from the currently selected input unit and to be assigned as the values of the variables in the list.

WRITE(m) Statement

General Form
WRITE(m) list m is the number of an output unit. list specifies the variables to be transmitted.

EXAMPLE:

```
WRITE (10) X, Y, MASS, TEMP
```

This statement causes the values of the listed variables to be written on the currently selected output unit. Because there is no FORMAT statement, the values will be listed, using a specification of E15.8 for real values and I11 for integer values. These specifications can be changed as explained under "EDIT Statement."

If the program is compiled using a regular FORTRAN processor at a later time, the WRITE(m) statement will write data in binary form.

WRITE(m, n) Statement

General Form
WRITE(m, n) list m is the number of an output unit. n is the number of a FORMAT statement. list specifies the quantities to be transmitted.

EXAMPLES:

```
WRITE (24,5) K, A (L)
WRITE (J,3) JOBNO, ANS, X
```

This statement causes the values of the items in the list to be written on the currently selected output unit. The FORMAT statement must not specify more than 114 characters per record.

If the program is compiled using a regular FORTRAN processor at a later time, the output will be written in BCD form.

Program Defining Statements

Program defining statements identify the type of program or subprogram and indicate the statements that are contained in it. Every program must start with a PROGRAM, FUNCTION, or SUBROUTINE statement and must physically end with an END statement.

Main Programs

Every main program must begin with a PROGRAM statement and end with an END statement.

PROGRAM Statement

General Form
PROGRAM name name is a 1-character through 6-character name, the first character of which is alphabetic (none may be a special character).

EXAMPLES:

```
PROGRAM STRESS
PROGRAM FARAD8
```

This statement identifies the first statement in a main program and assigns the name to it.

END Statement

General Form
END

EXAMPLE:

```
END
```

This statement indicates the end of every program or subprogram and can never appear elsewhere in a program.

After an END statement has been entered, the Remote Computing System will print diagnostic messages to indicate errors that are not apparent until the program has been completed, e.g., references to statement numbers that do not exist.

Functions and Subroutines

The statements pertaining to functions and subroutines are given below. Additional information is contained in the general information manual FORTRAN, Form C28-8074.

To allow for error checking, the Remote Computing System requires that FUNCTION and SUBROUTINE statements appear as declarative information within the program that calls them. The mode, order, and number of arguments stated as declarative information must agree with the definitions of the FUNCTION and SUBROUTINE statements.

FUNCTION Statement

General Form
FUNCTION name (p_1, p_2, ..., p_p) name is the name of a function; from one through eight arguments may be used.

EXAMPLES:

```
FUNCTION ARCSIN (RADIAN)
FUNCTION ROOT (B, A, C)
```

This statement is used at the beginning of a single-valued FUNCTION subprogram to define its name and arguments. The name of the function must appear at least once as an element of an executable statement within the subprogram; however, the values of the arguments must not be changed within the subprogram.

Normally, the initial letter of a function name determines the type of the function. If desired, the user may explicitly specify the type of the function by beginning the FUNCTION statement with the appropriate word, i.e., REAL FUNCTION or INTEGER FUNCTION.

SUBROUTINE Statement

General Form
SUBROUTINE name (p_1, p_2, ..., p_p) name is the name of a subroutine; from zero through eight arguments may be used.

EXAMPLES:

```
SUBROUTINE MATMPY (A, N, M, B, L, C)
SUBROUTINE QDRTIC (B, A, C, ROTTI, ROTT2)
```

This statement is used at the beginning of a SUBROUTINE subprogram to define its name and arguments. The name of the subroutine must not appear within the subprogram.

Reserved Functions

Reserved function names must not be used as the names of variables nor as the names of other functions.

Because the first letter of the names of reserved functions determines their types, the names of reserved functions must not appear in REAL, INTEGER, or EXTERNAL statements.

The table below lists the available reserved functions. In the table, I denotes an integer expression and X denotes a real expression. The first letter of the name specifies the type of the function.

NAME	DEFINITION
ATAN(X)	Compute the principal arctan value of the real argument X in radians.
ATAN2(X1, X2)	Compute ATAN(X1/X2).
EXP(X)	Compute e^x for a real argument X (whose absolute value must be less than 88).
SIN(X) COS(X)	Compute the sine or cosine of a real argument X in radians (whose absolute value must be less than 2^{36}).
TANH(X)	Compute the tangent of the real argument X in radians.
ALOG(X)	Compute the natural logarithm for a positive nonzero real argument X.
ALOG10(X)	Compute the common logarithm of X.
SQRT(X)	Compute the square root of a real argument X greater than zero.
ARSIN(X)	Compute the arcsine of a real argument X.
ARCOS(X)	Compute the arccosine of a real argument X.
ABS(X) LABS(I)	Absolute value of the argument.
AINT(X) INT(X)	Truncate, sign of the argument times the largest integer less than or equal to the absolute value of the argument.
AMOD(X1, X2) MOD(I1, I2)	Remaindering. For example: AMOD(X1, X2) = X1 − AINT (X1/X2) *X2 MOD(I1, I2) = I1 − INT(I1/I2)*I2
AMAX0(I1, I2) AMAX1(X1, X2) MAX0(I1, I2) MAX1(X1, X2)	Choose the larger value of the pair of arguments.
AMIN0(I1, I2) AMIN1(X1, X2) MIN0(I1, I2) MIN1(X1, X2)	Choose the smaller value of the pair of arguments.
FLOAT(I)	Convert from integer to real.
IFIX(X)	Same as INT.
SIGN(X1, X2) ISIGN(I1, I2)	Sign of the second argument times the absolute value of the first argument.
DIM(X1, X2) IDIM(I1, I2)	Positive difference. For example: DIM(X1, X2) = X1 − AMIN1(X1, X2) IDIM(I1, I2) = I1 − MIN0(I1, I2)

RETURN Statement

General Form
RETURN

EXAMPLE:

```
RETURN
```

This statement, which can appear only in subprograms, is used to return control from a subprogram to the main program that called it. Each subprogram must contain at least one RETURN statement. The next executed statement following a RETURN statement must be numbered.

CALL Statement

General Form
CALL name (e_1, e_2, ..., e_p) name is the name of a Subroutine subprogram, which can appear only in CALL or EXTERNAL statements. From zero through eight arguments may be used. Each argument must be an expression, a function, or a subprogram.

EXAMPLES:

```
CALL MATMPY (X, 5, 10, Y, 7, 2)
CALL QDRTIC (P*9.732, Q/4.536, R-S**2.0, X1, X2)
```

This statement is used to call Subroutine subprograms. The CALL transfers control to the subprogram and makes the arguments available to it.

Operating Statements

The operating statements allow the user to control, modify, test, and display programs.

Figure 3 shows all of the acceptable forms of each operating statement. The purpose of each statement is given later. The following table gives the meanings of the symbols used to identify the arguments in the operating statements:

SYMBOL	TYPE OF ARGUMENT
c	Constant
g	Line number
h	Line number or statement number
m	Input or output unit number
n	Statement number
u	Name of a program
v	Variable (simple or subscripted)

The portion of an operating statement that is shown in capital letters is called the statement operator. Before describing the individual operating statements, general information that applies to the different forms of statements will be given. Information that applies to specific operating statements will be included with the description of that statement.

Statement Operator with No Argument

When no argument follows the statement operator, the function of the statement operator is initiated immediately and is in effect over the entire range of the program. A statement operator that ends with an X cancels the operation of the corresponding operating statement. For example:

OPERATING STATEMENT	DESCRIPTION
TRAP	Initiates a TRAP
TRAPX	Cancels the TRAP
SNAP	Initiates a SNAP
SNAPX	Cancels the SNAP

ALTER (h)	INDEX (n)	SNAPX (v)
ALTER (h_1, h_2)	INDEX (v)	SNAPX (h_1, h_2)
ALTER X	LIST	START
AUDIT	LIST (h)	START 0
CHECK	LIST (h_1, h_2)	START (h)
CLEAR	LOAD	STEP
COMMAND	NUMBER	STEPX
COPY	NUMBER (g)	TRACE
COPY (h)	NUMBER (g, c)	TRACE (h)
COPY (h_1, h_2)	PDUMP	TRACE (v)
DELETE (v)	PURGE (u)	TRACE (h_1, h_2)
EDIT (s)	QDUMP	TRACEX
EDIT (s_1, s_2)	RESET	TRACEX (h)
EXIT	SAVE	TRACEX (v)
GUARD	SAVE (u)	TRACEX (h_1, h_2)
GUARD (h)	SELECT (m)	TRAIL
GUARD (h_1, h_2)	SNAP	TRAILX
GUARDX	SNAP (h)	TRAP
GUARDX (h)	SNAP (v)	TRAP (h)
GUARDX (h_1, h_2)	SNAP (h_1, h_2)	TRAP (h_1, h_2)
HALT	SNAPX	TRAPX
INDEX	SNAPX (h)	TRAPX (h)
		TRAPX (h_1, h_2)

Figure 3. Acceptable Forms of Operating Statements

Statement Operator with One Argument

One argument is enclosed in parentheses following the statement operator.

When the argument is a line number or a statement number, the function of the statement operator will be in effect for that portion of the program beginning with that number and continuing to the end of the program.

When the argument is the name of a variable, the function of the statement operator will be in effect for that variable.

A statement operator that ends with an X cancels the operation of the corresponding operating statement as specified by the argument. For example:

OPERATING STATEMENT	DESCRIPTION
SNAP(176., 276.)	Initiates a SNAP from line 176 through line 276
SNAPX(212.)	Cancels SNAP above line 212
SNAPX	Cancels SNAP for entire program

Statement Operator with Two Arguments

Two arguments separated by a comma are enclosed in parentheses following the statement operator.

When a pair of line numbers or statement numbers is used, it specifies the range over which the function of the statement operator will be in effect. The first argument specifies the start of the range and the second argument specifies the end of the range. Both arguments may specify the same line or statement if the user wants to refer to only one statement.

A statement operator that ends with an X cancels the operation of the corresponding operating statement as specified by the arguments. The arguments need not be identical to the range specified at the time the operating statement was put into effect. For example:

OPERATING STATEMENT	DESCRIPTION
TRACE(123., 456.)	Initiates tracing between lines 123 and 456
TRACEX(234., 278.)	Cancels tracing between lines 234 and 278 (tracing continues 123-233 and 279-456)
TRACEX(425.)	Cancels tracing above line 425 (tracing continues 123-233 and 279-424)
TRACEX	Cancels all tracing

Control Statements

LOAD Statement

The LOAD statement can be used only in the command mode; an error will be indicated if it is used in the program mode. The LOAD statement causes the terminal to enter the program mode and specifies that a program, subroutine, or function is to be placed in active status. Note that the argument of a LOAD statement must not be placed in parentheses. The argument may be the name of any program, subroutine, or function in the user's library.

START Statement

The START statement initiates execution of the program. To begin executing a program from the beginning, the user may type START 0. To resume execution from the next statement to be executed after the last termination of execution, the user may type START. To begin execution at any statement, the user may specify a line or statement number in parentheses following START.

HALT Statement

The HALT statement interrupts program execution. When HALT is specified, execution is interrupted and the terminal reverts to ready status. The system prints a message informing the terminal of the line number at which execution has been halted.

RESET Statement

The RESET statement cancels the effect of executing the active program or part of the active program. The Remote Computing System considers a variable "set" if a value has been assigned to it; a variable is considered "used" if it has been referred to in an executed statement. Similarly, the Remote Computing System considers a statement "used" if it has been executed. The RESET statement causes the Remote Computing System to regard all statements as "not used" and all variables as "not set" and "not used."

CLEAR Statement

The CLEAR statement cancels the effect of all previous test and display statements, and also causes the Remote Computing System to regard all statements as "not used" and all variables as "not set" and "not used." Specifying the CLEAR statement is equivalent to specifying TRACEX, TRAPX, etc., and RESET. Note that the CLEAR statement cancels the effect of test and display statements that have appeared in the operand of executed CALL statements.

SAVE Statement

The SAVE statement places the currently active program, subroutine, or function into the user's library. If the name of the currently active program, subroutine, or function is the same as a name already in the library, it cannot be entered into the library with that name. (Unless, of course, the program, subroutine, or function is first removed from the library as explained under "PURGE Statement.") The user can place the currently active program, subroutine, or function into the library by assigning a new name to it. The new name of the program, subroutine, or function to be added to the library is specified in parentheses following SAVE. After the SAVE statement has been executed, the program, subroutine, or function added to the library is retained as the currently active program at the terminal.

PURGE Statement

The PURGE statement removes the named program, subroutine, or function from the user's library. When the PURGE statement is specified, the named program, subroutine, or function is no longer available to the terminal and may not be the argument of a CALL or LOAD statement.

EXIT Statement

When the EXIT statement is specified, the active program image is destroyed. Hence, if active programs are to be recalled later, the EXIT statement must be preceded by a SAVE statement.

Further procedures for terminating active status are contained under "End of Terminal Operation." The EXIT statement may be specified in both command and program modes.

COMMAND Statement

The COMMAND statement places the terminal in the command mode. When the COMMAND statement is specified, the active program image is destroyed. Hence, if active programs are to be recalled later, the COMMAND statement must be preceded by a SAVE statement.

Modification Statements

The language includes modification statements that provide a means for changing the program. These statements are not retained as part of the program and are not compatible with FORTRAN IV. The user employs these statements to add, change, and delete program statements. Figure 3 shows the acceptable forms of the statements.

ALTER Statement

The ALTER statement permits the user to add, change, or delete program statements. If one argument appears after ALTER, statements may be inserted *after* the statement designated by the argument. The line number of the statement at the specified argument is increased by 0.1 to provide for the first insertion. After each statement is inserted, the line number is increased by 0.1.

If two arguments appear after ALTER, the statements from the first argument through the second argument will be deleted from the program. The user may then insert statements using line numbers *beginning* with the line number of the statement located at the first argument. The line numbers will be increased by 0.1 as explained above.

Status Word ALTER: After the ALTER statement has been specified, the system will print the status word ALTER at the terminal. The user may then type a statement he wishes to insert or may specify another ALTER location.

DELETE Statement

The DELETE statement specifies a variable that is to be deleted from the program. The argument that appears after DELETE may be the name of any variable specified in the program. All statements referring to the variable named must be adjusted *by the user* to reflect the deletion of the variable from the program.

NUMBER Statement

The NUMBER statement renumbers (sequentially by line number) all statements in the program currently active at the terminal. The statements will be renumbered beginning with line number 101.; the line numbers will be incremented by 1.

If one argument appears after NUMBER, renumbering will begin with the line number designated by the argument. The line number specified by the argument will be assigned to the first statement in the program.

If two arguments appear after NUMBER, renumbering will begin with the line number designated by the first argument. Subsequent line numbers will be incremented by the value designated by the second argument, which may be any number between 0.1 and 9.

The user can obtain a listing of the renumbered program or part of the renumbered program by using a LIST statement (see "LIST Statement").

Test Statements

The language includes statements that enable the user to test a program while it is being composed and after it has been completed. Test statements provide information concerning the changes in value of variables, the execution of transfers, etc. Most of the test statements can also be included as part of a program through the use of a CALL statement. Figure 3 shows the acceptable forms of the statements.

SNAP Statement

The SNAP statement causes the printing of the value of the leftmost variables in arithmetic assignment statements whenever the value of the variables changes during execution.

Status Word SNAP: During execution, when a message is to be printed in response to a SNAP statement or to the SNAP facility of a TRACE statement, the message will be preceded by the status word SNAP.

TRAP Statement

The TRAP statement causes the printing of the line numbers of the origin and destination of every transfer of control that takes place during program execution.

Status Word TRAP: During execution, when a message is to be printed in response to a TRAP statement or to the TRAP facility of the TRACE statement, the message will be preceded by the status word TRAP.

TRACE Statement

The TRACE statement causes the printing of the line number of each statement executed in a program. Specifying the TRACE statement is equivalent to specifying both a SNAP and a TRAP statement.

Status Word TRACE: When a statement that falls within the range of a TRACE statement is executed, the line number of the statement executed and the status word TRACE will be printed at the terminal. Whenever a transfer of control takes place during execution within the range of a TRACE statement, a message preceded by the status word TRAP will be printed at the terminal. Whenever the value of the leftmost variable in an arithmetic assignment statement changes during execution and the statement containing the variable is within the range of a TRACE statement, a message preceded by the status word SNAP will be printed at the terminal.

TRAIL Statement

The TRAIL statement causes a message to be printed at the terminal whenever a transfer of control takes place to a function, a subroutine, a program in the user's library, or a program supplied by the system.

GUARD Statement

The GUARD statement causes a halt in program execution and causes control to return to the terminal. A message will be printed before the execution of statements that are "guarded." The message will indicate that the execution of the program is in a "guarded" region and will return control to the terminal. The status word READY will then be printed at the terminal. If the user types START, one more statement will be executed and control will again be returned to the terminal.

Status Word GUARD: Before a statement within a region specified by a GUARD statement is executed, a message will be printed at the terminal. The message will be preceded by the status word GUARD.

STEP Statement

The STEP statement permits the user to interrupt execution after the printing of every line of output from a program. When a STEP statement has been specified, the Proceed light will go on at the terminal after each line of output is printed. The user may then interrupt execution.

Display Statements

The language includes display statements that permit the user to obtain information about a program at any stage of its composition or execution. The information printed through the use of display statements enables the user to analyze the progress of his program. This information includes, for example, listings of statements, defined variables, unused variables, and unexecuted statements. When composing a program, the user is able to detect errors and omissions from the information produced by the display statements. When executing a program, the user can employ display statements to observe the operation of the program by monitoring the effect of execution on one or more variables and/or statements.

A display statement can produce a great amount of information. At times, the user does not need all the information; for example, an error may be apparent after only a few lines have been printed. Therefore, the display statements are designed to permit the user to interrupt them at any time during their execution. Figure 3 shows the acceptable forms of the statement.

LIST Statement

The LIST statement causes a listing of program statements retained by the system to be printed at the terminal.

Status Word LIST: Each statement printed at the terminal as a result of a LIST statement will contain the status word LIST.

COPY Statement

The COPY statement causes a listing of program statements to be typed at the terminal. The COPY statement produces the same type of listing as does the LIST statement, except that line numbers and status words are not included as part of the listing. The COPY statement is also useful, when specified after a SELECT statement (see "SELECT Statement"), for producing a FORTRAN compatible source program deck.

INDEX Statement

The INDEX statement produces a listing of the variable names and statement numbers specified in a program. If no argument appears after INDEX or if the argument is zero, the following will be printed at the terminal:

1. A numerical listing of all statement numbers appearing in a program. The listing indicates, by line number, those statements that are identified by or referenced by statement numbers. A minus sign preceding a line number signifies that the statement number is referred to in a statement; a plus sign preceding a line number indicates that the statement number is used to identify the statement. If a statement number has not been both used as an identification and referred to elsewhere in the program, it will be flagged.

2. An alphabetical listing of all the variables in a program. The listing indicates, by line number, those statements in which a particular variable has been specified. A minus sign preceding a line number signifies that the variable has been referred to in the statement; a plus sign preceding a line number indicates that the variable is defined by that statement; a blank (no sign) indicates that the variable was declared by that statement. If a statement has not been both defined and referred to, it will be flagged.

If the argument following INDEX is a statement number, a listing like the one in item 1 above will be produced for that statement number. If the argument following INDEX is the name of a variable, a listing like the one described in item 2 above will be produced for that variable.

Status Word INDEX: For every statement number or variable name, each initial line of output resulting from an INDEX statement will contain the status word INDEX.

CHECK Statement

The CHECK statement produces a listing of every statement number and variable name that would have been flagged because of an INDEX statement. The CHECK statement may be specified before or during program execution. The listing contains variables and statement numbers in the same order and format as described in the INDEX statement.

Status Word CHECK: For every statement number or variable name, each initial line of output resulting from a CHECK statement will contain the status word CHECK.

AUDIT Statement

The AUDIT statement produces a numerical listing of all statement regions that have not been executed and an alphabetical listing of the names of all variables that have not had values assigned or used in a program.

Status Word AUDIT: Each line of output resulting from an AUDIT statement will contain the status word AUDIT.

PDUMP Statement

The PDUMP statement produces an alphabetical listing of the names of all variables, with the current value of each variable or an indication that the variable has not yet been assigned a value.

Status Word PDUMP: Every line of output resulting from a PDUMP statement will contain the status word PDUMP.

QDUMP Statement

The QDUMP statement produces an alphabetical listing of the names of all variables with the current values of those variables that have changed in value since the last execution of a PDUMP or QDUMP statement. The QDUMP statement may be specified during or upon completion of program execution.

Status Word QDUMP: Every line of output resulting from a QDUMP statement will contain the status word QDUMP.

Input/Output

The language contains statements that allow the user to specify input/output units and to specify the format of the output typed at the terminal.

SELECT Statement

The SELECT statement specifies the input or output unit to be used. The unit to be used is specified by the argument. All information sent to an output unit will also be printed on the printer.

EDIT Statement

The EDIT statement specifies the format of output for all real and integer variables that are not under control of a FORMAT statement.

Initially, each terminal is set to the standard real format of E15.8 and the standard integer format of I11. These formats are used for the output of all variables not under control of a FORMAT statement. The formats specified in the EDIT statement override the standard formats for the terminal. If one argument appears after

EDIT, it may be either a real or an integer format specification. If two arguments appear, one argument must be a real format specification and the other an integer format specification. The EDIT statement may be specified in both command and program modes.

Program Called Services

The operating statements listed below function as system subroutines and may be used as the operand of a CALL statement, which is retained as part of the active program. Any acceptable form (see Figure 3) of the following statements can be used:

AUDIT	STEP
CLEAR	STEPX
COPY	TRACE
LIST	TRACEX
PDUMP	TRAIL
QDUMP	TRAILX
RESET	TRAP
SNAP	TRAPX
SNAPX	

The user may cancel the effect of a statement by using a statement operator that ends with an X. For example:

PROGRAM STATEMENT	DESCRIPTION
CALL TRAP(234., 456.)	Initiates trapping between lines 234 and 456
CALL TRAPX(400.)	Cancels trapping above line 400 (trapping continues 234-399)
CALL TRAPX	Cancels all trapping

The user may also cancel the effect of the statement by interrupting program execution and typing a statement operator that ends with an X. For example:

STATEMENT	DESCRIPTION
CALL TRACE(345., 567.)	Initiates tracing between lines 345 and 567
HALT	Halts program execution
TRACEX(345., 500.)	Cancels tracing between lines 345 and 500 (tracing continues 501-567)
START	Resumes program execution
HALT	Halts program execution
TRACEX	Cancels all tracing
START	Resumes program execution

Note, however, that operating statements used as arguments of CALL statements work like any other program retainable statement. That is, if program control returns to a CALL statement containing an operating statement as its operands, the effect of the operating statement will be reinitiated.

Part 4. Examples

To illustrate the use of the Remote Computing System, examples from an experimental system are given below. These examples are intended only to show how remote computing could be implemented; they are not intended to show how the IBM 7040/7044 Remote Computing System operates. The experimental system from which these examples were taken has statement operators, statement formats, and messages that differ from those in the Remote Computing System. For example, UNLOAD, TRACE, and DUMP in the experimental system correspond to SAVE, SNAP, and PDUMP in the Remote Computing System. The reader should allow for such differences as he follows the explanations of the examples.

The examples in this section are divided into two groups: (1) The first group contains a program that illustrates many of the features available in the experimental system when used for composing a program; (2) The second group contains several examples of the execution of statements, consisting entirely of constants and FORTRAN functions, in the command mode. Figures 4 and 5 illustrate the formation of a program; Figure 6 illustrates the use of the command mode.

Figure 4 shows the final, correct version of the program. Figure 4B shows the correct output produced as a result of execution (see START statement, Figure 4A, line 128).

Figure 5 depicts a preliminary attempt to create and test this program. (All references that follow are to Figure 5.)

Input to the system may be from the keyboard or card reader at the remote terminals. At line 106 a mispunched card causes printing of an error message. The user now suspends automatic input, substitutes a correct statement via the keyboard and then resumes automatic input. Of course, the substitution could have been made later by an ALTER statement (see below).

At lines 119 and 120, the user initiates intermediate execution and verifies his FORMAT statements before going further. In this manner, any statement, sequence of statements, DO loop, etc., may be debugged as the program is entered. Similarly, sections may be tested independently of the remainder of the program.

Execution of the entire program at line 140 discloses a number of bugs. Inspection of line 137 discloses the use of K as a subscript. K could be printed selectively, but the user decides to dump all variables (see line 141). After DUMP starts, he interrupts it to change the format of the display and then dumps again (see line 143). In the event that the dump showing K-51 is not a sufficient clue to the error, the user establishes a TRACE on K and a TRAP on the entire program, and starts again (see lines 144-146). This produces, together with his programmed output lines, a dynamic listing of control and data flow, before ending with the same error message as on line 137.

At line 147, the statements in error are corrected, but a statement number is inadvertently omitted. When the user ends the ALTER status, a message is printed pointing out that the DO at line 128 refers to a nonexistent label. This error is corrected and a subsequent running of the program (line 151) shows that the subscript is now behaving properly.

However, there are other changes to be made. The NUMBER at line 152 yields a clean, renumbered listing of the current state of the program. Line 231 shows a complete INDEX, and line 232, the results of a CHECK statement. All of these will be helpful in reorganizing and documenting the final, correct version of the program.

```
                  COMMAND
101 -READY C      THIS IS A SAMPLE PROGRAM.
101 -READY C
101 -READY        PROGRAM SAMPLE
102 +READY        DIMENSION ZPLOT(52), TABLE(500)
103 +READY        X = 0
104 +READY        Y = 1.
105 +READY        I = 1
106 +READY        READ 101, DELX,CHAR,ZPLOT
107 +READY   101   FORMAT(F7.4,53A1)
108 +READY        PRINT 102
109 +READY   102   FORMAT(5X1HX7X1HY)
110 +READY      2 TABLE(I) = X
111 +READY        TABLE(I+1) = Y
112 +READY      1 PRINT 103, X, Y
113 +READY   103   FORMAT(2XF7.4,F8.5)
114 +READY        IF(X-1.)5,3,3
115 +READY      5 I = I + 2
116 +READY        X = X + DELX
117 +READY        DELY = X * Y * DELX
118 +READY        Y = Y + DELY
119 +READY        GOTO 2
120 +READY      3 DO 4 J = 1, I, 2
121 +READY        X = TABLE(J)
122 +READY        K=1.+((TABLE (J+1)-TABLE(2))/(TABLE(I+1)-TABLE(2))*50.)
123 +READY        ZPLOT(K) = CHAR
124 +READY        PRINT 101, X, ZPLOT
125 +READY      4 ZPLOT(K) =ZPLOT(K+1)
126 +READY        STOP 77
127 +READY        END
128 +READY        START 0
```

Figure 4A. Sample Program: Final Form

```
106 =I101   00.0625*
108 =0102         X         Y
112 =0103       0.      1.00000
112 =0103       0.0625  1.00391
112 =0103       0.1250  1.01175
112 =0103       0.1875  1.02361
112 =0103       0.2500  1.03960
112 =0103       0.3125  1.05990
112 =0103       0.3750  1.08475
112 =0103       0.4375  1.11441
112 =0103       0.5000  1.14923
112 =0103       0.5625  1.18963
112 =0103       0.6250  1.23610
112 =0103       0.6875  1.28922
112 =0103       0.7500  1.34965
112 =0103       0.8125  1.41819
112 =0103       0.8750  1.49574
112 =0103       0.9375  1.58339
112 =0103       1.0000  1.68235
124 =0101    0.     *
124 =0101    0.0625*
124 =0101    0.1250*
124 =0101    0.1875 *
124 =0101    0.2500  *
124 =0101    0.3125    *
124 =0101    0.3750      *
124 =0101    0.4375        *
124 =0101    0.5000          *
124 =0101    0.5625              *
124 =0101    0.6250                  *
124 =0101    0.6875                      *
124 =0101    0.7500                          *
124 =0101    0.8125                              *
124 =0101    0.8750                                  *
124 =0101    0.9375                                      *
124 =0101    1.0000                                          *
126 =S77
129 +READY        UNLOAD
```

Figure 4B. Sample Program: Final Execution

```
                COMMAND
101 -READY C    THIS IS A SAMPLE PROGRAM
101 -READY      PROGRAM SAMPLE
102 +READY      DIMENSION ZPLOT (51), TABLE(500)
103 +READY      DELX = .2
104 +READY      X = 0
105 +READY      Y = 1.
106 +READY  2 =  1
106 +ERROR 04200. STATEMENT NOT IN LANGUAGE.
107 +READY CV   TYPOGRAPHICAL ERRORS MAY BE CORRECTED IMMEDIATELY
108 +READY CV   BY SUBSTITUTING A CORRECT STATEMENT VIA KEYBOARD.
109 +READY      I = 1
110 +READY      READ 101, CHAR, ZPLOT
111 +READY  101   FORMAT(52A1)
112 +READY      PRINT 102
113 +READY  102   FORMAT(5X1HX5X1HY)
114 +READY      GOTO 2
115 +READY    1 PRINT 103, X , Y
116 +READY  103   FORMAT(2XF5.2,F8.5)
117 +READY CV    ANY STATEMENT OR SEQUENCE OF STATEMENTS MAY BE
118 +READY CV    VERIFIED BY IMMEDIATE EXECUTION AFTER ENTRY.
119 +READY      START 0
110 =I101  *
112 =0102       X      Y
114 =ERROR  TRANSFER POINT N DOES NOT EXIST
120 +READY      START 1
115 =0103     0.    1.00000
116 =CYCLE  END OF PROGRAM ENCOUNTERED DURING EXECUTION
121 +READY      I = I + 2
122 +READY      X = X + DELX
123 +READY      DELY = X*Y*DELX
124 +READY      Y = Y + DELY
125 +READY    2 TABLE(I) = X
126 +READY      TABLE(I+1) = Y
127 +READY      IF(X - 1.)1,1,3
128 +READY    3 DO 4 J = 1, I, 2
129 +READY      X = TABLE (J)
130 +READY      K=1+((TABLE(J+1)-TABLE(2))/(TABLE(I+1)-TABLE(2))*50.)
130 +ERROR  ARITHMETIC DECOMPOSITION ERROR(S)
130 +ERROR  MIXED MODE
131 +READY CV   STATEMENTS IN ERROR AT TIME OF ENTRY ARE NOT ACCEPTED.
132 +READY CV    SUBSTITUTION MAY BE MADE WITHOUT RE-ENTERING PROGRAM.
133 +READY      K=1.+((TABLE(J+1)-TABLE(2))/(TABLE(I+1)-TABLE(2))*50.)
134 +READY      ZPLOT(K) = CHAR
135 +READY      PRINT 104, X , ZPLOT
136 +READY  104  FORMAT(F5.2,51A1)
137 +READY    4 ZPLOT(K) = ZPLOT(K+1)
138 +READY      STOP 77
139 +READY      END
```

Figure 5A. Sample Program: Creating and Testing

```
140 +READY         START 0
110 =I101  *
112 =0102        X      Y
115 =0103      0.    1.00000
115 =0103      0.20 1.04000
115 =0103      0.40 1.12320
115 =0103      0.60 1.25798
115 =0103      0.80 1.45926
115 =0103      1.00 1.75111
135 =0104    0.   *
135 =0104    0.20 *
135 =0104    0.40      *
135 =0104    0.60            *
135 =0104    0.80                  *
135 =0104    1.00                              *
135 =0104    1.20                                                  *
137 =ERROR   VALUE OF SUBSCRIPT IS ZERO, NEGATIVE, OR EXCEEDS DIMENSION
141 +READY         DUMP
    =              CHAR=-0.14191581E-08
    =              DELX= 0.20000000E-00
    =              DELY= 0.42026726E-00

142 +READY         EDIT(F8.5)
143 +READY         DUMP
     ERROR    ILLEGAL CHARACTER IN TEXT
                   DUMP
    =              CHAR=-0.00000
    =              DELX= 0.20000
    =              DELY= 0.42027
    =              I=            13
    =              J=            13
    =              K=            51
    =              X= 1.20000
    =              Y= 2.17138
    =              TABLE(1)= 0.
    =              TABLE(2)= 1.00000
    =              TABLE(3)= 0.20000
    =              TABLE(4)= 1.04000
    =              TABLE(5)= 0.40000
    =              TABLE(6)= 1.12320
    =              TABLE(7)= 0.60000
    =              TABLE(8)= 1.25798
    =              TABLE(9)= 0.80000
    =              TABLE(10)= 1.45926
    =              TABLE(11)= 1.00000
    =              TABLE(12)= 1.75111
    =              TABLE(13)= 1.20000
    =              TABLE(14)= 2.17138
    =              TABLE(500)= 0.
    =              ZPLOT(1)=-6.09524
    =              ZPLOT(2)=-6.09524
    =              ZPLOT(3)=-6.09524
    =              ZPLOT(4)=-6.09524
    =              ZPLOT(5)=-6.09524
  DUMP ALWAYS MAY BE INTERRUPTED.
```

Figure 5B. Sample Program: Creating and Testing (Continued)

```
144 +READY        TRACE K
145 +READY        TRAP 101./138.
146 +READY        START 0
110 =I101  *
112 =0102        X      Y
114 =TRAP  TRANSFER TO  2 (125)
127 =TRAP  TRANSFER TO  1 (115)
115 =0103      0.    1.00000
127 =TRAP  TRANSFER TO  1 (115)
115 =0103      0.20 1.04000
127 =TRAP  TRANSFER TO  1 (115)
115 =0103      0.40 1.12320
127 =TRAP  TRANSFER TO  1 (115)
115 =0103      0.60 1.25798
127 =TRAP  TRANSFER TO  1 (115)
115 =0103      0.80 1.45926
127 =TRAP  TRANSFER TO  1 (115)
115 =0103      1.00 1.75111
127 =TRAP  TRANSFER TO  3 (128 )
133 =TRACE        K=            1
135 =0104  0.    *
133 =TRACE        K=            2
135 =0104  0.20   *
133 =TRACE        K=            6
135 =0104  0.40       *
133 =TRACE        K=           12
135 =0104  0.60             *
133 =TRACE        K=           20
135 =0104  0.80                    *
133 =TRACE        K=           33
135 =0104  1.00                                *
133 =TRACE        K=           51
135 =0104  1.20                                                  *
137 =ERROR VALUE OF SUBSCRIPT IS ZERO, NEGATIVE, OR EXCEEDS DIMENSION
```

Figure 5C. Sample Program: Creating and Testing (Continued)

```
147 +READY        ALTER 137./137.
137 +ALTER        ZPLOT(K) = BLANK
1371+ALTER        ALTER 110.
1101+ALTER        BLANK = ZPLOT(1)
1102+ALTER        ALTER *
1102+ERROR   DO   128.0 REFERENCES UNDEFINED LABEL  4
148 +READY        ALTER 137./137.
137 +ALTER     4  ZPLOT(K) = BLANK
1371+ALTER         ALTER*
149 +READY        TRACE* K
150 +READY         TRAP* 101./138.
151 +READY        START 0
110 =I101  *
112 =0102        X      Y
115 =0103      0.    1.00000
115 =0103      0.20 1.04000
115 =0103      0.40 1.12320
115 =0103      0.60 1.25798
115 =0103      0.80 1.45926
115 =0103      1.00 1.75111
135 =0104    0.  *
135 =0104    0.20 *
135 =0104    0.40     *
135 =0104    0.60           *
135 =0104    0.80                  *
135 =0104    1.00                              *
135 =0104    1.20                                          *
138 =S77
152 +READY
```

Figure 5D. Sample Program: Creating and Testing (Continued)

```
            NUMBER 201.
201 =    CF    PROGRAM SAMPLE
202 =          DIMENSION ZPLOT(51),TABLE(500)
203 =          DELX=.2
204 =          X=0
205 =          Y=1.
206 =          I=1
207 =          READ 101,CHAR,ZPLOT
208 =          BLANK=ZPLOT(1)
209 =      101 FORMAT(52A1)
210 =          PRINT 102
211 =      102 FORMAT(5X1HX5X1HY)
212 =          GO TO 2
213 =        1 PRINT 103,X,Y
214 =      103 FORMAT(2XF5.2,F8.5)
215 =          I=I+2
216 =          X=X+DELX
217 =          DELY=X*Y*DELX
218 =          Y=Y+DELY
219 =        2 TABLE(I)=X
220 =          TABLE(I+1)=Y
221 =          IF(X-1.)1,1,3
222 =        3 DO 4 J=1,I,2
223 =          X=TABLE(J)
224 =          K=1.+(TABLE(J+1)-TABLE(2))/(TABLE(I+1)-TABLE(2))*50.
225 =          ZPLOT(K)=CHAR
226 =          PRINT 104,X,ZPLOT
227 =      104 FORMAT(F5.2,51A1)
228 =        4 ZPLOT(K)=BLANK
229 =          STOP 77
230 =          END
```

Figure 5E. Sample Program: Creating and Testing (Continued)

```
231 +READY       INDEX
    =            1       +213.   -221.
    =            2       +219.   -212.
    =            3       +222.   -221.
    =            4       +228.   -222.
    =       *    5       +207.
    =          101       +209.   -207.
    =          102       +211.   -210.
    =          103       +214.   -213.
    =          104       +227.   -226.
    =       BLANK        +208.   -228.
    =       CHAR         +207.   -225.
    =       DELX         +203.   -216.   -217.
    =       DELY         +217.   -218.
    =       I            +206.   +215.   -215.   -219.   -220.
    =                    -222.   -224.
    =       J            +222.   -223.   -224.
    =       K            +224.   -225.   -228.
    =       *SAMPLE       201.
    =       TABLE         202.   +219.   +220.   -223.   -224.
    =       X            +204.   -213.   +216.   -216.   -217.
    =                    -219.   -221.   +223.   -226.
    =       Y            +205.   -213.   -217.   +218.   -218.
    =                    -220.
    =       ZPLOT         202.   +207.   -208.   +225.   -226.
    =                    +228.
232 +READY       CHECK
    =       *    5       +207.
    =       *SAMPLE       201.
233 +READY
```

Figure 5F. Sample Program: Creating and Testing (Continued)

```
                COMMAND
101 -READY        Y=2.5065*10.**(10.+1.)*EXPF (-10.)
101 =             Y= 0.11379489E 08
101 -READY        HENRY=2.E-9*50.*(LOGF(2.*50./10.)-1.0+10./50.
101 -ERROR  04117. PARENTHESES NOT IN BALANCE.
101 -READY        HENRY =2.E-9*50.*(LOGF(2.*50./10.)-1.+10./50.)
101 =             HENRY= 0.15025850E-06
101 -READY        ROOT1=(-25.+SQRTF(25.**2-4.*1.*2.))/(2.*1)
101 -ERROR   ARITHMETIC DECOMPOSITION ERROR(S)
101 -ERROR   MIXED MODE
101 -READY        ROOT1=(-25.+SQRTF(25.**2-4.*1.*2.))/(2.*1.)
101 =             ROOT1=-0.80257654E-01
101 -READY        HENRY=2.E-9*50.*(LOGF(2.*50./10.)-1.0+10./50.)
101 =             HENRY= 0.15025850E-06
101 -READY        VAL=1./COSF(50.)+LOGF(ABSF(SINF(50./2.)/COSF(50./2.)))
101 =             VAL=-0.97714996E 00
101 -READY        AREA=2.*10.*5.*SINF(3.1416/10.)
101 =             AREA= 0.30901768E 02
101 -READY        ARC=2.*SQRTF(4.**2+1.3333*2.**2)
101 =             ARC= 0.92375753E 01
101 -READY        ARC=2.*(4.*4.+4.*2.*2./3.)**0.5
101 =             ARC= 0.92376041E 01
101 -READY        S=-COSF(40.)**(20.+1.)/20./1.)
101 -ERROR  04117. PARENTHESES NOT IN BALANCE.
101 -READY        G=0.5*LOGF((1.+SINF(45.))/(1.-SINF(45.)))
101 =             G= 0.12594177E 01
101 -READY        S=SINF(45.)
101 =             S= 0.85090352E 00
101 -READY        G=0.5*LOGF((1.+.7071)/(1.-.7071))
101 =             G= 0.88135999E 00
101 -READY        E=20.*ATANF(20./4.)-4./2.*LOGF(4.**2+20.**2)
101 =             E= 0.15406644E 02
101 -READY        Q=(2./(3.1416*10.))**0.5*SINF(10.)
101 =             Q=-0.13726357E-00
101 -READY        Q=0.7978/SQRTF(10.)*SINF(10.)
101 =             Q=-0.13724918E-00
101 -READY C
101 -READY
```

Figure 6. Examples of Command Mode Operation

Appendix: Comparison with FORTRAN Systems

Serious attention has been paid to maintain consistency between the Remote Computing System and other FORTRAN processors. Programs written in the language described in this publication are acceptable without change to conventional FORTRAN IV processors. Conversely, FORTRAN IV programs are acceptable to the Remote Computing System with the limitations given below. The chart at the end of this appendix compares FORTRAN statements for other systems with those used in the Remote Computing System.

The following limitations apply to FORTRAN IV programs to make them acceptable to the Remote Computing System:

1. The user's program must be written with statements from the subset defined for the system.

2. As is the case for all one-pass translators, all declarative statements must precede the executable statements. Of course, FORMAT statements and comment statements may appear anywhere in the program.

3. As in most compilers, the sequence of machine instructions produced for arithmetic expressions may differ from those produced by other compilers; therefore, slight discrepancies caused by variations in truncations may occur.

4. Similarly, some minor differences in the internal representation of program constants, caused by different conversion routines, may also create slight differences in numerical results.

5. Individual source programs are limited to about 400 statements. However, the user can overcome this limitation by dividing oversized programs into smaller subprograms.

6. Limitations regarding program statements are:
   a. No arithmetic function statements.
   b. No logical, complex, or double-precision variables.
   c. Number of constants, variables, arrays, and functions must be less than 190.
   d. No continuation cards.
   e. No magnetic tape input or output. However, the 1052 will be used for statements that require tape units.
   f. Real constants up to eight digits, with magnitude within range of 10^{-38} to 10^{38} or zero.
   g. Integer constants up to ten digits.
   h. Array names must appear in a DIMENSION statement before appearing in any other statement.
   i. Maximum size of input/output record is 114 characters, except card records, which are limited to 80 characters.
   j. Arrays cannot be arguments of functions and subroutines, but must be passed through COMMON.
   k. Statement numbers must not exceed three digits in length.
   l. Restrictions on use of EQUIVALENCE.
   m. Library function names are reserved.
   n. Declarative statements must precede first executable statement.
   o. Reserved functions must not appear in an EXTERNAL statement.
   p. A program or subprogram must contain a FUNCTION or SUBROUTINE statement for each function or subroutine which the program or subprogram calls.

Statements in the Remote Computing System		Statements for performing similar functions in FORTRAN for other systems									
			COMMON	650	650 FOR TRANSIT	1620	705	BASIC 7070/7074	7070/7074	704	709/7090
EXTERNAL $u_1,u_2,\ldots,u_p$	X										
INTEGER $z_1,z_2,\ldots,z_p$	X										
REAL $z_1,z_2,\ldots z_p$	X										
PROGRAM name	X										
		ACCEPT n,List				X					
		ACCEPT TAPE n,List				X	X		X	X	X
		ASSIGN i TO n					X		X	X	X
		BACKSPACE i					X	X	X	X	X
CALL name $(e_1,e_2,\ldots,e_p)$	X	CALL NAME $(a_1,a_2,\ldots,a_n)$							X	X	X
COMMON $v_1,v_2,\ldots,v_p$	X	COMMON $(a_1,a_2,\ldots,a_n)$							X	X	X
CONTINUE	X	CONTINUE	X	X	X	X	X	X	X	X	X
DIMENSION $y_1,y_2,\ldots y_p$	X	DIMENSION $v_1,v_2,\ldots,v_n$	X	X	X	X	X	X	X	X	X
DO n $i=m_1,m_2,m_3$	X	DO n $i=m_1,m_2,m_3$	X	X	X	X	X	X	X	X	X
END	1	END(l_1,l_2,l_3,l_4,l_5)		1	1	2		2	2	2	2
		END FILE i					X	X	X	X	X
EQUIVALENCE $(v_1,v_2,\ldots,v_p)$	X	EQUIVALENCE (a,b,c,...),(d,e,f,...),...			X			X	X	X	X
FORMAT (specifications)	X	FORMAT $(s_1,s_2,\ldots,s_n)$				X	X	X	X	X	X
		FREQUENCY n(i,j,...),m(k,l,...),...							3	X	X
FUNCTION name $(p_1,p_2,\ldots,p_p)$	X	FUNCTION Name $(a_1,a_2,\ldots,a_n)$							X	X	X
GO TO n	X	GO TO n	X	X	X	X	X	X	X	X	X
		GO TO n, $(n_1,n_2,\ldots,n_m)$					X		X	X	X
GO TO $(n_1,n_2,\ldots,n_p),i$	X	GO TO $(n_1,n_2,\ldots,n_m)$, i	X	X	X	X	X	X	X	X	X
		IF ACCUMULATOR OVERFLOW n_1,n_2					X	X	X	X	X
		IF DIVIDE CHECK n_1,n_2					X	X	X	X	X
		IF QUOTIENT OVERFLOW n_1,n_2					X	X	X	X	X
IF (e) n_1,n_2,n_3	X	IF (a) n_1,n_2,n_3	X	X	X	X	X	X	X	X	X
		IF (SENSE LIGHT i) n_1,n_2					X		X	X	X
		IF (SENSE SWITCH i) n_1,n_2				X	X		X	X	X
PAUSE d	X	PAUSE n	4	5	5	4	5	5	5	5	5
PRINT n, list	X	PRINT n, List				X	X	X	X	X	X
PUNCH n, list	X	PUNCH n, List	6	6	6	X	X	X	X	X	X
		PUNCH TAPE n, List				X					
READ n, list	X	READ n, List	6	6	6	X	X	X	X	X	X
		READ DRUM i,j, List					3			X	X
READ (m,n) list	X	READ INPUT TAPE i,n, List					X	X	X	X	X
READ (m) list	7	READ TAPE i, List					X	X	X	X	X
RETURN	X	RETURN							X	X	X
		REWIND i					X	X	X	X	X
		SENSE LIGHT i					X		X	X	X
STOP d	X	STOP n	4	4	5	4	X	X	X	X	X
SUBROUTINE name$(p_1,p_2,\ldots,p_p)$	X	SUBROUTINE Name $(a_1,a_2,\ldots,a_n)$							X	X	X
		TYPE n, List				X		X	X		
		WRITE DRUM i,j, List					3			X	X
WRITE (m,n) list	X	WRITE OUTPUT TAPE i,n, List					X	X	X	X	X
WRITE (m) list	7	WRITE TAPE i, List					X	X	X	X	X

1. l_i are not permitted.
2. l_i are optional and may be ignored.
3. May be included but will be ignored.
4. The n is not permitted.
5. The n is optional and may be ignored.
6. The n is optional and is ignored.
7. The m is optional.

Index

C28-6800-0

IBM

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