File No. 7040-25
Form C28-6800-2

IBM Systems Reference Library

IBM 7040/7044 QUIKTRAN

User's Guide

This publication provides the information necessary to use the IBM 7040/7044 QUIKTRAN System. It contains a description of the system, which provides concurrent access to a computer for users at up to 40 remotely located terminals. Also included are the elements of the QUIKTRAN language, a programming language that is similar to the FORTRAN language.

There are two methods of using the QUIKTRAN System: "conversational processing" and "batch processing," both of which are described in detail.

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 publication *General Information Manual; FORTRAN*, Form F28-8074.

For detailed information about terminal equipment the user can refer to the publication *System Operation Reference Manual, IBM 1050 Data Communication System*, Form A24-3020.

MAJOR REVISION (August 1965)
This edition, C28-6800-2, makes obsolete the previous edition C28-6800-1, which was titled *IBM 7040/7044 QUIKTRAN User's Guide.*

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 QUIKTRAN System

The IBM 7040/7044 QUIKTRAN System is a programming system designed to provide concurrent computer access to a maximum of 40 remotely located terminals. The programming language employed by the terminal user is upward compatible with most FORTRAN IV processors and is augmented by a set of operating statements and a set of terminal commands. The user communicates with the system either in a "conversational" (on-line) manner in which the basic unit of input is a single statement, or in a "batch" (off-line) manner in which the basic unit of input is an entire program. (Batch operation is described later in this publication in "Basic Information for Batch Processing.")

The design of the QUIKTRAN System satisfies the following requirements of the user:

1. The user rarely has access to experts for programming assistance and advice. If he uses a problem-oriented language to express his problem, his request for and display of debugging data needs to be consistent with the programming language.

2. Because the user operates the machine himself when his jobs are being processed, he has to have the ability to regulate the system, using statements that are similar to the programming language of the system. 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, discontinuing a job, etc.

3. When dealing with large or small input/output volumes, the user must be able to modify decks without their complete retransmission, and he should have the option to list and inspect data selectively, rather than be forced to transmit entire output files. He also desires to keep his various decks in random storage where they are quickly and conveniently available for modification, processing, or review.

4. The user should be able to operate as though he were the only user and in complete control of the situation. More specifically, in a time-sharing environment, the user should be fully protected from unwanted, possibly destructive, interaction with others. Finally, he must be able to begin his jobs at any time without waiting for other users and to continue using the system as long as is necessary.

These requirements are met in the following ways:

1. Output data is as problem-oriented as the source language. The source language is quite similar to the FORTRAN language.

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

3. The user has immediate and continued access to the computer.

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

5. Each user is assigned an identification code that must be communicated to the system as the first step in a terminal operation. The system, in turn, uses this code to determine which programs in its library belong to which user. Each time a user begins operating from a terminal, he first enters 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 a user from executing, changing, or destroying another user's program(s).

6. The print volume may be reduced, without loss of quality, at the request of the user.

The Approach

The approach to satisfying these requirements 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 continued access to a computer for a large number of users. Together, these two techniques make the conversational manner of operation by remotely located 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 innate power of the computer into condensed solution-time and greater creative power for the users.

System Concepts

The language of the QUIKTRAN System comprises program statements, operating statements, and terminal commands. Program statements are upward compatible with FORTRAN IV and are used to construct the program. Operating statements allow the user to regulate the system. Terminal commands allow the user to regulate operations at the terminal.

When a program is being constructed, tested, executed, 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.

Computer and Terminals

The QUIKTRAN System requires a computing center equipped with a data processing system having tape, drum, and disk storage, an exchange device, and a network of up to 40 remotely located terminals.

The "Equipment" section of this publication gives a full description of the required equipment.

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

Programs can be permanently retained in disk storage. When the user indicates that a program he has constructed is to be saved, the system places it on disk storage. Thereafter, the program will be available to the user whenever needed. The maximum number of programs that can be stored depends upon the sizes of the programs. Assuming that the average program occupies 2,000 words of storage, then approximately 1,300 programs can be retained on disk storage at any one time.

Conversational Processing

The user is said to be communicating with the system in a "conversational" manner when each statement he enters through the terminal is processed (translated, verified, and, if desired, executed) immediately. The system then sends a reply to the terminal. The information contained in the reply varies. For example, it might be a message indicating that the previous statement contained an error (see "System Replies" for further information).

Operations in the conversational manner must be in either of two modes: the program mode or the command mode.

Program Mode

The program mode is that mode of operation which allows the user to construct his program on line, statement by statement. A terminal is said to be in the program mode when a program is active at that terminal (see "Program Defining Statements"). In this mode, the user enters program statements that make up the substance of his program, and he operates (i.e., modifies, tests, executes, and debugs) on the program by using operating statements (see "Language").

When the terminal is in the program mode, the user can also enter single statements that are executed immediately but are not retained as part of the active program (see "Process Codes").

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 (see "COMMAND Statement"), he destroys the active image of his program. Since, in this mode, no program can be active at the terminal and statements cannot be retained, each statement must be processed immediately. Consequently, the user may employ only those statements in the language which do not depend on any other statements. These are the general operating statements, the program defining statements, and a limited form of the arithmetic assignment statement. This last provision allows the terminal to be used as a fast, versatile symbolic calculator. In this mode, the user can enter a statement in the form $X = e$, where e is any expression consisting of constants and reserved functions, (see "Reserved Functions") and the system immediately evaluates the expression and prints the result at the user's terminal.

Batch Processing

Batch processing provides a means of sending entire FORTRAN, COBOL, MAP, etc., programs to the computing center for off-line execution. In addition, batch processing allows the user to request that one or more of the FORTRAN programs he has debugged and saved during conversational operation be compiled or compiled and executed off line. By allowing debugged programs to be executed off line, batch processing frees the terminal for the construction, debugging, execution, and saving of new FORTRAN programs. For further details on the batch processor, see "Basic Information for Batch Processing."

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.).

Errors of composition are detected as soon as the user enters the offending statement. The system rejects the offending statement, and the user can immediately substitute a correct statement (see "Status Word RJECT").

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.).

Most errors of consistency are detected as soon as the user enters the offending statement. The system rejects the offending statement and the user can immediately substitute a correct statement (see "Status Word RJECT"). However some errors of consistency (e.g., illegal branch into the range of a DO) are not immediately detected. These errors are handled in the same manner as errors of completeness (see below), and should be considered as such.

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 completeness are detected after the user has indicated that his program is complete (see "END Statement" and "ALTERX Statement"). All such errors are then extracted and immediately displayed at the terminal in a sequential list. When all the errors have been listed, the user can then individually correct or disregard them before initiating the execution of his completed program. Any disregarded errors, when redetected during execution, are considered as execution errors (see below).

Execution: Those errors detected during the execution of the user's program. These include errors that are detectable only during program execution (e.g., invalid subscript value, reference to an undefined variable, arithmetic spills, etc.) along with those errors of completeness detected because either (1) they were disregarded by the user when previously detected or (2) they were not detected in the first place because the user did not indicate that his program was complete.

An execution error causes an immediate execution interruption at the point at which the error is encountered. The error is extracted and displayed at the terminal. The user may then correct the error and resume the execution of his program. If the user chooses to ignore the error and continue the execution, he may do so.

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 his responsibility. However, he is provided with an extensive set of debugging aids that allow him to manipulate portions of 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 selected portions of a program.
3. Print changes of values as a change occurs and print transfers of control as a transfer occurs.
4. Obtain a static printout of all cross-reference relationships among names and labels, and dynamic exposure of partial or imperfect execution.

The equipment required for use of the QUIKTRAN 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 is described in detail because the QUIKTRAN user works through 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 the permanent retention of users' programs.
3. An IBM 7320 Drum Storage Unit to be used for the temporary storage of users' programs.
4. Six magnetic tape units to be used for maintaining normal 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 communications devices for connecting terminals to the computer.

Terminal Equipment

A user's terminal consists of an IBM 1050 Data Communications System and a communications device for connecting the terminal to the computer on which the QUIKTRAN System is operating.

An IBM 1050 Communications System can consist of various components. For use with QUIKTRAN, a 1050 System must include an IBM 1051 Control Unit and an IBM 1052 Printer-Keyboard. Input to the QUIKTRAN System is entered by the user via the keyboard. The printer records both input and output.

For greater flexibility and efficiency, other components can be included in the 1050 System. The addition of an IBM 1056 Card Reader makes it possible to enter QUIKTRAN input from cards. If the card reader has a pack feed, an entire deck can be read without manual intervention. A card reader with a single card feed requires that cards be entered manually, one at a time. The Card Reader Program feature makes it possible to control card reading with a punched paper tape. Specifically, this program tape can be used to prevent the reading of any identification information that may exist beyond column 72 of the input cards.

All output always appears on the printer portion of the 1052 Printer-Keyboard. However, a second copy of output can be obtained in card or listing form by including an IBM 1057 Card Punch or an IBM 1053 Printer as part of the terminal equipment. Only one of these extra output devices can be in use at one time.

Description of the IBM 1052 Printer-Keyboard

The IBM 1052 Printer-Keyboard consists of a printer, switch panel, and a keyboard.

Printing is done on continuous-form paper by a ball-shaped print element that moves horizontally to the desired position of the line.

The switch panel consists of lights, switches, and an indicator that shows the position of the print element.

The 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.

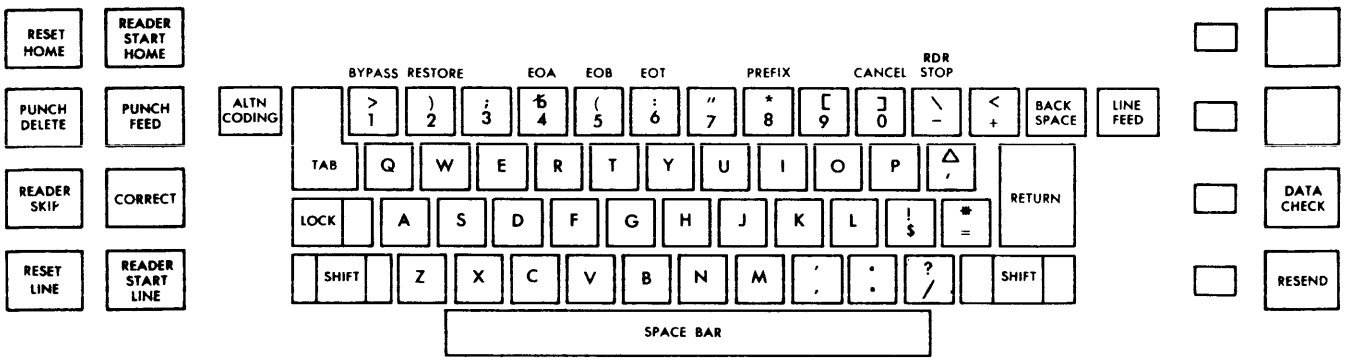


Figure 1. IBM 1052 Keyboard

Initial Setup of the Terminal

To use the QUIKTRAN 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 with an asterisk (*) are optional and may not appear 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	HOME/SEND
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 maximum line length. The first twelve print positions of each line are reserved for use by the QUIKTRAN System. The information printed in these positions is explained in the section "Basic Information for QUIKTRAN Programming."

Print position 13 is the first one available to the user and may be regarded as though 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 system pauses for a response from the user, the carrier is positioned at column 1. The user may find it convenient to set a tab stop at print position 19 so that, when a QUIKTRAN statement is to be entered, he can press the TAB key to move the carrier 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 away from 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.

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. In order to clear 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.
7. Check that the printer paper has been inserted in the paper sensing device (i.e., the paper should be under the metal bar located above and behind the platen). Set the printer carriage for single spacing.

If the 1052 is equipped with a pin-feed platen, the paper release lever must be set in the release position.

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

When the connection has been made, QUIKTRAN sends a message to the terminal. After the message is printed, the PROCEED light is turned on and the keyboard is unlocked. The system is now ready to accept the first entry from either the keyboard or the card reader. Note that the first entry in *any* application must be the communication of the user's identification code (see "Terminal Commands" for further information).

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." These instructions are also summarized in the lower half of Figure 2.

SYSTEM		MASTER		PRINTER 1		PRINTER 2		KEYBOARD		RDR 1		RDR 2		PUNCH 1		PUNCH 2	
Attend	X	Norm		RCV		RCV		Send	X	Send		Send		RCV		RCV	
		Off	X	Send RCV	X	Send RCV		Off		Off	X	Off	X	Off	X	Off	X
Un-attend		Master		Home / Send		Home / Send	X	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
Off	X	Off	X	Bksp		Dup	X		Dial Disc		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	Step	Card Reader Operation
1.	Wait for the system to type a line number and a status word.	1.	Place the card deck in the card reader. Press the FEED button.
2.	Wait for the PROCEED light to go on.	2.	Wait for the system to type a line number and a status word.
3.	Type one statement.	3.	Wait for the PROCEED light to go on.
4.	Hold down ALTN CODING and press EOB; this will cause the PROCEED light to be turned off.	4.	Set the KEYBOARD switch to OFF.
		5.	Set the RDR1 switch to SEND.
		6.	Press READER START LINE.
5.	Repeat steps 1 through 4 for each statement to be entered.	7.	One card is read and ejected.
		8.	The system types a line number and a status word.
		9.	Steps 7 and 8 are repeated automatically until the required number of cards have been read.
		10.	To return to keyboard operation, set the RDR1 switch to OFF and the KEYBOARD switch to SEND.

Figure 2. QUIKTRAN Terminal Operating Procedure

If the user wishes to have his output also appear on punched cards, he may follow the instruction given in the section "Auxiliary Output Devices."

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 system; therefore, the user should not press LINE FEED or RETURN. The functions of the other keys are:

KEY NAME	FUNCTION
SHIFT	This key must be held down while pressing the keys needed to type any of the following special characters: *) (; No other keys should be pressed while SHIFT is held down. The characters shown, and all those special characters that do not require the SHIFT function, are valid for the QUIKTRAN System. (See Appendix C for a list of valid special characters.) The shift LOCK key should *never* 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 QUIKTRAN System, tab stops are set for columns 7 and 120 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 four 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 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, transmit EOB.
EOT	Pressing this key while the PROCEED light is on indicates to the system that the user desires to interrupt the execution of his program. The ALTN CODING key must be held down when EOT is pressed.

The ALTN CODING key can be used only to transmit EOB, EOT and CANCEL. QUIKTRAN does not use the alternate coding function of any other keys. Note, however, that the BYPASS function locks the keyboard; the RESTORE function unlocks the keyboard.

The procedure for entering information from the keyboard is:

1. The system prints a line number and a status word (e.g., READY); printing stops with the carrier at column 1.
2. Wait for the PROCEED light to be turned on. This indicates that the system is ready to accept input information.
3. Type one line of input.
4. Transmit EOB to indicate the end of a line and to return control to the system. The PROCEED light is turned off after the EOB has been transmitted.
5. Return to step 1 to continue operating. To switch to card input, follow the procedure given under "Procedures 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

All cards to be read by the IBM 1056 Card Reader must have an upper left-hand 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.

The procedure for entering information from the card reader is:

1. Place the card deck (face down with the column 1 edge first) into the hopper of the card reader and then press the FEED button.
2. A line number and a status word are printed by the system and the carrier stops at column 1.
3. Wait for the PROCEED light to be turned on. This indicates that the system is ready to accept input information.
4. Set the KEYBOARD switch to OFF. (This step is optional; the KEYBOARD switch may be left in the SEND position, if so desired.)
5. Set the RDR1 switch to SEND.
6. Press READER START LINE.
7. A card is then read and ejected by the system.
8. The system then prints a line number and a status word. If there is an error in the card just read, the key-

board must be used to correct the error (skip to step 10 to return to keyboard operation) before the next card is read.

9. Steps 7 and 8 are repeated until the required number of cards have been read.

10. To interrupt the card reader and/or to return to keyboard operation, set the RDR1 switch to OFF and the KEYBOARD switch to SEND, then follow the instructions given under "Procedures for Keyboard Input."

When the card reader is used, the transmission of an EOB is required to indicate the end of each card, just as it is required to indicate the end of each line for keyboard input. There can be up to three different methods by which an EOB may be transmitted when the card reader is used:

1. The user may punch the EOB (0-6-9 punch) at the desired position in his input cards. Any information appearing in a card after an EOB punch is ignored.

2. The user can set the AUTO EOB switch on the card reader to ON and an EOB will automatically be transmitted after column 80 of each input card.

3. If the 1056 is equipped with the Card Reader Program feature, an EOB can be transmitted by a paper program tape that has been inserted in the carriage tape control device. To transmit an EOB after column 72 of each input card (so that any identification information will be ignored), this tape must be punched as follows:

   a. Column 73 must have punches in channels 2, 3, 4, 5, and 6.

   b. Column 74 must have a punch in channel 12.

The program tape is inserted into the carriage tape control device (located within the card reader and reached from the right-hand panel) by placing column 73 over the top right pin in the wheel of the device. If this is done correctly, column 1 will be directly under the reading brushes of the device. This description is a specific case of a general procedure. For further information about this, see the publication *System Operation Reference Manual, IBM 1050 Data Communication System,* Form A24-3020.

If this procedure is used, the PROGRAM TAPE switch should be set to ON.

Inserting More Printer Paper

If the 1052 Printer runs out of paper during operations, printing will stop and the RECEIVE ALARM light will turn on. The procedure for inserting new paper is as follows:

1. Set PRINTER 1 switch to HOME.
2. Insert paper in the paper sensing device at the back of the platen and around the platen.
3. Set PRINTER 1 switch to SEND/REC.

Auxiliary Output Devices

The IBM 1057 (or 1058) Card Punch and the IBM 1053 Printer are optional output devices that the user may have installed at his terminal. Although terminal output is always registered on Printer 1 (i.e., the printer portion of the 1052), it can also be directed (under program control) to either the card punch or Printer 2 (i.e., the 1053) but not both at the same time. Only one auxiliary output device may be in operation at any one time. (See "AUXOP Statement" for program-controlled auxiliary output procedures.)

If the card punch is to be used, it should first be prepared for operation as follows:

1. Mount a program card containing the letter A in all columns on the program drum and lower the star wheels to read the card.
2. Place a quantity of blank cards into the feed hopper.
3. Set the Main-Line switch to POWER ON.
4. Set the AUTO FEED switch to ON.
5. Set AUTO SKIP, AUTO DUP to the off position.
6. Set AUTO PUNCH, KEY PUNCH to AUTO PUNCH.
7. Press the FEED key twice.
8. Press the REL key once.

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 system has been discontinued for the session. The user may disconnect the terminal in either of the following ways:

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

This part describes the basic information with which the QUIKTRAN user should be familiar in order to utilize fully the capabilities of the system. Included are descriptions of the necessary actions of both the user and the system when a program is being constructed, executed, modified, etc.

NOTE: The word "program" does not necessarily imply operations in the program mode. A "program" can consist of a single statement. Hence, operations in the command mode are by no means excluded from this discussion.

Statement Entry Format

The user enters statements by typing them on the IBM 1052 Printer-Keyboard. Unless specifically stated otherwise, all references to "typing at the keyboard" also apply to punched cards for the card reader. A FORTRAN coding form, Form X28-7327, may be used for planning entries. Information printed by the system will be discussed according to the print position(s) in which it appears. Data entered by the user will be discussed according to the columns of the FORTRAN coding form. Print positions 1 through 12 are reserved by the system for the assignment of line numbers, status indicators, and status words. Print positions 13 through 132 (i.e., columns 1 through 120) are available to the user. Details of the print positions are described below.

Print Positions 1 Through 5

The QUIKTRAN 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 101.0 and increased according to the user's specification (see "DELTA Statement") up to a maximum of 999.0. With line numbers, the user can refer to a point in the program at which he may add or delete statements, begin execution, start or end a testing operation, etc.

The 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 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 branch statements, the system prints the line number of the statement that caused a branch and the line number of the statement to which it transferred.

The system keeps track of the line numbers assigned and deleted, and prints the next available line number when it is 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 returns to the proper point after the user-requested action has been completed.

Print Position 6

The system prints a status indicator (see "Status Indicators") in print position 6.

Print Positions 7 Through 11

The system prints a status word in these positions (see "Status Words"). Various status words inform the user of the action taken by the system, the kind of output produced, etc.

Print Position 12

This print position is always left blank. It is the last position reserved for system use.

Columns 1 Through 120

Column 1, which corresponds to print position 13, is the first position available to the user for entering statements. Although any valid nonalphameric character (except a semicolon) 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 199. The user need number only those statements to which reference will be made in his program.

Column 6 must either be blank or contain a digit from 0 through 4 (see "Statement Continuation").

Columns 7 through 120 (7 through 80 for card reader input) may be used for any of the statements described in the section "Language." As explained in the section "Equipment," it is convenient to have a tab stop set for column 7.

Statement Continuation

If the user requires more than one line to complete a single statement, he can use one of the following techniques:

Keyboard or Card Input

The first line of the statement must have the integer 0 in column 6. All continuation lines (except the last) must contain any integer from 1 through 3 in column 6. The last continuation line must contain a 4 in column 6. Each line is sent to the system in the normal fashion (i.e., transmit EOB after the last character in each line has been typed). This method is useful when entries are made from the card reader.

Keyboard Input Only

For each line (except the last), the tab key is depressed after the last character has been typed. After the tab key has been depressed, transmit EOB. The last line is sent to the system in the normal fashion. This method is recommended when entries are made from the keyboard. (Column 6 of each line should be left blank.)

The above techniques are not mutually exclusive. Continuation can be initiated by either a 0 in column 6 or the pressing of the tab key followed by an EOB. All continuation lines (except the last) can be indicated by either an integer from 1 through 3 in column 6 or the pressing of the tab key followed by an EOB. Continuation can be ended by either a 4 in column 6 or a normal EOB.

NOTES: The total number of characters in a continuation-line statement must not exceed 335. Statements may not be continued in the command mode nor may the continuation techniques be applied to comments.

Input Cancellation

Input, be it a single line or a continuation-line statement, may be cancelled by the user at any time during its construction but not after it has been sent to the system. Once a line has been accepted by the system, it can only be "altered" (see "ALTER Statement").

Single-Line Cancellation

In order to cancel a single line at any time before it has been sent to the system, the user must transmit the CANCEL code (see the "Equipment" section of this publication) and then immediately follow with an EOB. The system effects the cancellation and the user may then resume on the next line.

Continuation-Line Statement Cancellation

The user may cancel an entire statement of more than one line in length at any time before the last line of the statement has been sent to the system. The procedure is to transmit *two* CANCEL codes in succession and immediately follow with an EOB. The system effects the cancellation and the user may resume on the next line.

NOTE: Any one line of a continuation-line statement is considered a single line and, thus, the procedure for its cancellation is the same as that described in "Single-Line Cancellation."

Comment Codes

When constructing a program, the user can enter various types of comments. 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

Comment Code Cb (the *b* indicates a blank) indicates to the system 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 is retained as part of the active program.

NOTE: Any valid nonalphameric character in column 1, except the semicolon, performs the same function as comment code Cb.

Comment Code CV

Comment code CV indicates to the system that the remainder of the line is a comment. The system does not retain comments identified with this comment code. This code can be used to print comments that are useful during program construction, but that are not to be part of the user's program.

Comment Code CF

Comment code CF is used to keep QUIKTRAN programs compatible with source programs for other FORTRAN processsors. The QUIKTRAN System processes statements beginning with comment code CF as though CF were not there; other FORTRAN processors regard these statements as comments.

When recomposing (i.e., listing, punching, or in any other way recording) a user's program, QUIKTRAN supplies comment code CF for all program statements that are not valid FORTRAN IV statements. If a QUIKTRAN statement is also a valid FORTRAN IV statement but has

been given a CF code by the user, then this statement will be rejected by the system at the time of entry.

Thus, if a QUIKTRAN statement is *not* a valid FORTRAN IV statement, whether or not the user has given it a CF code is unimportant because the system automatically supplies the code when the statement is recomposed.

QUIKTRAN statements that are not valid FORTRAN IV statements must not have statement numbers associated with them.

Execution

Command Mode Execution

When the terminal is in the command mode, each statement entered by the user is executed immediately upon entry and the result is printed at the terminal. Statements are not retained by the system so there can be no actual construction of a program.

Program Mode Execution

When the terminal is in the program mode, each statement or series of statements is executed only at the request of the user (see "Process Codes" and "START Statement"). Ordinarily, the user might wait until he has completed his coding; however, he does not have to wait since he can initiate execution at any time during the construction of his program. This allows the user to selectively test areas of his program before he actually completes his coding.

Process Codes

Process Codes provide the user with a semiautomatic execution facility. They can be used only when the terminal is in the program mode. Further, they can be used only in conjunction with arithmetic assignment statements and input/output statements. Process codes are not retained as part of the user's program.

Among other things, process codes allow the user to assign values to variables within a program and test the execution of an input/output statement in relation to its FORMAT.

Most process codes require that a statement not be typed on the same line as the code; they refer either to statements previously typed or to statements yet to be typed. If a process code refers to a statement previously typed (i.e., process code CX or CS), that statement is executed immediately upon entry of the process code. If requested, the result of the execution is then printed at the terminal.

If a process code refers to statements yet to be entered (i.e., process code CXX or CSS), these statements are executed immediately as they are entered. If requested, the result of the execution is printed at the terminal. This type of process code remains in effect until the user discontinues it (by entering CXX* or CSS*). Thus, one process code of this type can effect the execution of many statements.

Only one process code (namely, process code CC) requires that a statement be typed on the same line as the code. This statement is executed immediately, and the result is printed at the terminal. Neither the code nor the statement is retained as part of the user's program.

Process codes are typed starting in column 1. If an accompanying statement is required, the statement must be typed starting in column 7. The following table lists each process code with its associated meaning.

CODE	MEANING
CC	Execute and print the result of the arithmetic assignment or output statement which appears starting in column 7. The statement is executed by the system and then discarded. (The statement must not have a statement number.)
CX	Execute the last arithmetic assignment or input/output statement that was entered.
CS	Execute the last arithmetic assignment or input/output statement that was entered. If an arithmetic assignment statement is executed, the result is printed at the terminal.
CXX	Execute all subsequent arithmetic assignment and input/output statements as they are entered.
CSS	Execute all subsequent arithmetic assignment and input/output statements as they are entered. If any arithmetic assignment statements are executed, the result of each is printed at the terminal.
CXX*	Discontinue the effect of CXX.
CSS*	Discontinue the effect of CSS.

As was previously mentioned, process codes are useful when testing and debugging programs. All values necessary to test a program can be assigned by using process codes in conjunction with arithmetic assignment or input statements. The "expression" of an arithmetic assignment statement may include constants, reserved functions (e.g., COS, ATAN), and variables (see "Arithmetic Assignment Statement"). However, when an arithmetic assignment statement is used in conjunction with a process code, each variable included in the expression must have previously been assigned a definite value (which is another way of saying that these variables must have been assigned values via the previous execution of arithmetic assignment or input statements). Similarly, process codes can be used to display the current value of a variable. The user can display selected variables by listing their names in an output statement following process code CC. For example:

```
CC    PRINT 0, TEMP, VOL PRES
```

Interrupting Execution

When the system is performing a continuing service such as execution of the user's program or a program listing, the user may be periodically given the opportunity to interrupt the operation. These opportunities arise during those intervals when the PROCEED light is set on by the system. (Whether or not the PROCEED light comes on periodically is determined by the setting of the KEYBOARD switch. If the switch is set OFF, the PROCEED light will not go on and there will be no opportunity to interrupt the current operation; if the switch is set to SEND, the PROCEED light will go on periodically for at least 15 seconds.)

When the PROCEED light goes on, the current operation halts, the keyboard unlocks, and the system waits for some action by the user. The user can now interrupt the current operation by transmitting EOT (see "Procedures for Keyboard Input"). (Note that since the keyboard is unlocked, the user may precede the EOT with a comment; however, if 15 seconds elapse between the typing of successive characters, the PROCEED light will go off, the keyboard will lock, the current operation will resume, and the user will, for the time being, have lost his opportunity to interrupt.)

If no action is taken by the user while the PROCEED light is on, then after 15 seconds have passed, the PROCEED light will go off and the current operation will resume. The user can avoid this 15-second wait and have the current operation resume immediately by transmitting EOB as soon as the PROCEED light light goes on. (As with EOT, the EOB may be preceded by a comment.)

System Replies

When a program is being constructed, executed, modified, etc., the system maintains communication with the user by typing certain symbols and words in print positions 6 through 11.

Status Indicators

Status indicators, which are printed in print position 6, provide the user with information regarding the general status of the system. The three status indicators are:

```
+
-
=
```

The plus (+) sign indicates that the system is in the program mode. The minus (−) sign indicates that the system is in the command mode. The equals (=) sign indicates that the system is in automatic status (i.e., it has been executing a program or performing a continuing service such as a program listing) and one of many conditions has occurred. Some of these conditions are illustrated below:

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

```
126. = XEQER     VARIABLE X MAY NOT BE USED
                 UNTIL SET
```

2. A response to an operating statement has occurred and has been typed. For example:

```
293. = TRAP      TRANSFER TO 55 (213.)
```

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

```
456. = I 23
```

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

```
322. = O122      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. = STOP 77
```

Status Words

Status words, which are printed by the system in print positions 7 through 11, are more definitive than status indicators (i.e., they provide more detailed information about the status of the system).

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

Status Word READY

This status word indicates that the system is waiting for a statement entry from the terminal.

Status Word RJECT

This status word is accompanied by a message from the system indicating that an error has been detected during the translation of the statement entered by the user. In all cases, the statement is rejected (i.e., it is not saved as part of the user's program). The line number is not increased and the user may continue on the next line after an invitation by the status word READY or ALTER (see "ALTER Statement").

The RJECT status word appears when one of the following conditions has occurred:

1. The user has allowed too much time to elapse between the typing of characters.

2. The user has typed an illegal character as part of his input.

3. The length of the user's input line is in excess of the maximum allowed.

4. The input record contains a redundancy error.

5. An error of composition or consistency was detected in translating the user's input statement.

Status Word ERROR

This status word ordinarily indicates that the system has encountered an error while (1) checking the completeness and flow of the user's program (i.e., a completeness error) or (2) checking the allocation of storage for the variables in the user's program. It is accompanied by a message indicating the nature and location of the error. The statement (or statements) that caused the error is not discarded from the program.

For completeness errors: ERROR can appear only after the END statement has been entered by the user. The reason for this is that the system cannot make this check until the user has indicated, by entering an END statement, that his program is complete. A check for completeness errors is also made after the entry of an ALTERX statement, if and only if an END statement had been entered at some point prior to the "alter sequence" or in the "alter sequence" itself (see "ALTER Statement" and "ALTERX Statement").

For storage allocation errors: ERROR can appear at any time after the user has entered the first executable statement in his program. The entry of the first executable statement indicates to the system that all storage-allocating declarations have been entered by the user. A check for storage allocation errors is also made after the entry of an ALTERX statement, if and only if the first executable statement in the program had been entered at some point prior to the "alter sequence" or in the "alter sequence" itself (see "ALTER Statement" and "ALTERX Statement").

Status Word CANCL

This status word indicates that the system has deleted the preceding line (or lines, in the case of a continuation-line statement) in response to a cancel request from the user (see "Input Cancellation"). The line number is not incremented and the user may continue on the next line after an invitation by the status word READY or ALTER (see "ALTER Statement").

Status Word OVFLO

This status word indicates that the execution of an output statement has generated an output record that exceeds the maximum allowable line length of 120 characters. The excess characters accompany this status word.

Status Word NOTE

This status word and its accompanying message provide information about the way the system is compiling or executing the user's program. The message accompanying this status word is usually self-explanatory. The statement that caused NOTE is not discarded from the user's program.

The language of the QUIKTRAN System comprises program statements, operating statements, and terminal commands. Program statements, which are upward compatible with 7040/7044 FORTRAN IV (see Appendix A), are used to form the user's program. These statements are discussed in the section "Program Statements."

Operating statements allow the user to communicate with the CPU conversational processor of the QUIKTRAN System. Operating statements include modification, test, and display statements. These statements are discussed in the section "Operating Statements."

Terminal commands allow the user to communicate with the exchange device and, in some cases, with the CPU conversational processor. They include procedures for initializing and terminating a terminal operation. These commands are discussed in the section "Terminal Commands."

Elements of the Language

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. The rules for writing each of these types are given below.

Integer Constants

An integer constant is written without a decimal point, using the decimal digits 0, 1, . . ., 9. A preceding + or − is optional. An unsigned integer constant is assumed to be positive. An integer constant may consist of no more than ten digits.

Real Constants

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

A real constant may consist of no more than eight digits and its magnitude must lie within the range of 10^{-38} to 10^{38}.

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, variables may be integer or real values. In order to distinguish between variables that derive their values from integers as opposed to those that derive their values from real numbers, 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. This rule describes an inplicit method of defining an integer variable; an explicit method is described in the section "INTEGER Statement."

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., any alphabetic character *except* I, J, K, L, M, or N. This rule describes an implicit method of defining a real variable. An explicit method is described in the section "REAL Statement."

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, e.g., MAX(I, J, K). An array may not have a dimension greater than three.

Form of Subscripts

A subscript must be in one of the following forms only, where *v* represents any unsigned, nonsubscripted integer variable, and *c* (or *c'*) any unsigned integer constant:

```
v
c
v + c or v - c
c*v
c*v + c' or c*v - c'
```

Subscripted Variables

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

Arithmetic Expressions

An arithmetic expression consists of certain sequences of constants, subscripted and nonsubscripted variables, and arithmetic function references separated by arithmetic operation symbols, commas, and parentheses.

The arithmetic operation symbols +, −, *, /, and ** denote addition, subtraction, multiplication, division, and exponentiation, respectively. The rules for forming arithmetic expressions are:

1. Figures 3 and 4 indicate which constants, variables, and functions may be combined by the arithmetic operators to form arithmetic expressions. Figure 3 gives the valid combinations for the following arithmetic operators: +, −, *, and /. Figure 4 gives the valid combinations for the arithmetic operator **. In these figures, Y indicates a valid combination and N indicates an invalid combination.

2. The simplest expression consists of a single constant, a single variable, or a subscripted variable. If the quantity is integer, the expression is said to be of the integer type. If the quantity is real, the expression is said to be of the real type.

3. Quantities can be preceded by a + or a −, which does not affect the type of the expression. Also, expressions can be connected by any of the arithmetic operators (+, −, *, /, **) to form other expressions provided:
   a. No two operators appear consecutively.
   b. All operators are explicitly expressed.

4. Any expression may be enclosed in parentheses. Parentheses can be used to specify the order in which the operations in the expression are to be computed. Where parentheses are omitted, the hierarchy of operation is:
   a. Function Reference
   b. Exponentiation **
   c. Multiplication and Division * and /
   d. Addition and Subtraction + and −

+ - * /	Real	Integer
Real	Y	N
Integer	N	Y

Figure 3.

		Exponent	
	**	Real	Integer
Base	Real	Y	Y
	Integer	N	Y

Figure 4.

For example, the expression A+B/C+D**E*F−G will be taken to mean A+(B/C)+(D^{E*F})−G.

(The expression A**B**C is not permitted; it must be written as either A**(B**C) or (A**B)**C, whichever is intended.)

Concept of "Set" and "Used" Variables

"Set" Variables

The QUIKTRAN System considers a variable "set" if a value has been assigned to it. In other words, whenever a variable appears in the list specification of an input statement or to the left of the equals sign in an assignment statement, and that statement is executed successfully, then that variable is said to be "set."

An array is considered "set" if any of its elements is "set."

For further information on the effect of "set" variables with regard to the execution of input statements, see the sections "List Specifications" and "Execution of Input/Output Statements."

"Used" Variables

With two exceptions, the QUIKTRAN System considers a variable "used' when it appears in a program statement that has been executed, whether successfully or not. The exceptions are: (1) a "not used" variable appearing in an executed input statement remains "not used," and (2) a "not used" variable appearing to the left of the equals sign in an executed assignment statement remains "not used."

A program statement is considered "used" only if it has been executed successfully.

An array is considered "used" if any of its elements is "used."

In general, a variable may not be "used" until it has been "set." However, if a variable has been assigned to a *common* storage area (see "COMMON Statement"), it need not be "set" in order to be "used."

Program Statements

The user's program is made up of program statements. When the terminal is in the program mode, 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 latter statement. Numbers 1 through 199 may be used as statement numbers, but no two statements may have the same number. The statements acceptable to the QUIKTRAN System are described below. The description gives the general form of each statement, its purpose, and usually 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 program statements:

SYMBOL	TYPE OF VARIABLE
c	Constant
e	Arithmetic expression
i	Simple integer variable (unsigned)
k	Simple integer constant (unsigned)
m	Simple integer variable (unsigned) or a simple integer constant (unsigned)
n	Statement number
p	Nonsubscripted variable name — must not be the name of an array
u	Name of a program or subprogram
v	Simple or subscripted variable
x	Real expression
y	Subscripted variable
z	Name of an array, a function, or a simple variable

Program Defining Statements

Program defining statements identify the type of program or subprogram that is to be constructed. Every program or subprogram must start with a PROGRAM, FUNCTION, or SUBROUTINE statement and should end with an END statement.

NOTE: A PROGRAM, FUNCTION, or SUBROUTINE statement can be entered only when the terminal is in the command mode — see "COMMAND Statement." Entry of a PROGRAM, FUNCTION, or SUBROUTINE statement then puts the terminal into the program mode.

Main Programs

Every main program must begin with a PROGRAM statement.

PROGRAM Statement

General Form
PROGRAM name *name* should be no more than six characters long (no special characters allowed); the first character must be alphabetic. It may not be a reserved name (see Appendix B).

This statement identifies the first statement in a main program and assigns a name to the program. Some examples are:

```
PROGRAM STRESS
PROGRAM FARADS
```

END Statement

General Form
END

The END statement serves two purposes:

1. It indicates to the system that the user's main program or subprogram has been completed. The system then extracts and lists errors that were not apparent during program construction (see "Syntactic Errors"

and "Status Word ERROR"). Note, however, that any variables or statement labels that have not been defined or had reference made to them are not extracted at this time. If the user desires to have these elements extracted he must use the system service CHECK (see "CHECK Statement").

2. If the user's program or subprogram is to be compatible with other FORTRAN processors, the END statement must be present.

If the END statement is used, it must be the last program statement of a program or subprogram.

Subprograms

A subprogram is a function or a subroutine that can be called by a main program (or another subprogram) in order to perform a specific operation. The basic rules for constructing functions and subroutines are described below. Additional information is contained in the publication *General Information Manual: FORTRAN*, Form C28-8074.

FUNCTION Statement

General Form
FUNCTION name (p_1, p_2, . . . , p_n) where: 1. *name* is the symbolic name of a single-valued function. It may not be a reserved name (see Appendix B). 2. The arguments p_1, p_2, . . . , p_n (of which there must be at least one but not more than eight), are nonsubscripted variable names, none of which may be an array name or a reserved name (see Appendix B).

The FUNCTION statement is used to signify the beginning of a single-valued function subprogram. It defines the name of the function and the arguments to be used within the function. The name of the function must receive a value within the function subprogram. The values of the arguments are not returned to the calling program.

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. However, the name of a function may not appear in a REAL or INTEGER statement within the body of a function (see "Type-Defining Declarative Statements").

The following is an example of a function subprogram:

```
FUNCTION ZMULT(A, B, C)
    .
    .
    .
ZMULT=A*B*C
    .
    .
    .
RETURN
END
```

Rules for Calling Function Subprograms: Functions are called in a program by their appearance within an arithmetic expression, for example:

```
X=ZMULT(U, V, W)-D
```

All names of function subprograms are used in this manner. Their appearance in an arithmetic expression serves to call the function; the value of the function is then computed, using the arguments supplied in the parentheses following the function name. These arguments are arithmetic expressions which may consist of one or more subscripted or nonsubscripted variables, constants, user-defined subprogram names, or library function names (see "Library Functions"), but never an array name. The type, order, and number of the arguments must be the same as that of the function definition.

NOTE: If an argument in a call to a function subprogram is the name of a user-defined subprogram or the name of a library function, an EXTERNAL statement must be included with the declarative information in the calling program (see "EXTERNAL Statement" for details).

SUBROUTINE Statement

General Form
SUBROUTINE name (p_1, p_2, . . . , p_n) where: 1. *name* is the symbolic name of a subroutine subprogram. It may not be a reserved name (see Appendix B). 2. The arguments p_1, p_2, . . . , p_n (of which there need not be any, but not more than eight), may not be an array name or a reserved name (see Appendix B).

The SUBROUTINE statement is used as the first statement of a subroutine subprogram to define its name and arguments.

Unlike a function subprogram, which returns only a single value, a subroutine subprogram may use one or more of its arguments to return results to the calling program. The arguments may be considered dummy variable names that are replaced at the time of execution by the actual arguments supplied in the CALL statement that refers to the subroutine subprogram (see "Rules for Calling Subroutine Subprograms").

The actual arguments must agree with the dummy arguments in number, order, and type.

Two examples of the use of the SUBROUTINE statement follow:

```
SUBROUTINE MATMPY (A, N, B, L, K, M)
SUBROUTINE NOPAR
```

Rules for Calling Subroutine Subprograms: The CALL statement is used to transfer control from the program in which it appears to the subroutine to which it refers.

General Form
CALL name (e_1, e_2, . . . , e_n) where: 1. *name* is the symbolic name of a subroutine subprogram. It may not be a reserved name (see Appendix B). 2. The arguments e_1, e_2, . . . , e_n (of which there need not be any, but not more than eight) are arithmetic expressions that may consist of one or more subscripted or non-subscripted variables, constants, user-defined subprogram names, or library function names (see "Library Functions"), but never an array name.

Some examples of the use of the CALL statement are:

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

NOTES:

1. If an argument in a call to a subroutine subprogram is the name of another user-defined subprogram, or the name of a library function, an EXTERNAL statement must be included with the declarative information in the calling program (see "EXTERNAL Statement" for details).

2. Certain operating statements provide services that can be called by the user's program (see "Program Called Services" for additional information).

Return from a Function or Subroutine Subprogram

General Form
RETURN

This statement must appear at least once in a subprogram and is used to return control to the calling program. A RETURN statement must be the last executed statement of a subprogram, but not necessarily the physical last. The next executable statement following a RETURN statement must be numbered.

NOTE: Failure to include a RETURN statement within a subprogram will cause an execution error when that subprogram is executed.

Reserved Functions

A reserved function name should never be used either as the name of a variable or as the name of a subprogram. Because the first letter of the name of a reserved function determines its type, a reserved function name must never appear in REAL or INTEGER statements.

There are two types of reserved functions: library functions and built-in functions.

Library Functions

Library function names may appear in EXTERNAL statements and as the actual arguments of a function or subroutine called by the main program. The following table lists each of the library functions and its definition:

NAME	DEFINITION
EXP(x)	Computes e^x for a real argument x with an accuracy up to eight significant digits.
ALOG(x)	Computes the natural logarithm for a real argument x with an accuracy up to eight significant digits.
ALOG10(x)	Computes the common logarithm for a real argument x with an accuracy up to eight significant digits.
ATAN(x)	Computes the arc tangent of a real argument x. The result, in radians, is accurate up to eight significant digits.
ATAN2(x_1, x_2)	Computes the arc tangent of the real value x_1/x_2. The result, in radians, is accurate up to eight significant digits.
SIN(x)	Computes the sine of a real argument x, where x is expressed in radians. The result is accurate up to eight significant digits.
COS(x)	Computes the cosine of a real argument x, where x is expressed in radians. The result is accurate up to eight significant digits.
TANH(x)	Computes the hyperbolic tangent of a real argument x, where x is expressed in radians. The result is accurate up to eight significant digits.
SQRT(x)	Computes the square root of a real argument x, where x is greater than or equal to zero. The result is accurate up to eight significant digits.
ARSIN(x)	Computes the arc sine of a real argument x. The result, in radians, is accurate up to eight significant digits.
ARCOS(x)	Computes the arc cosine of a real argument x. The result, in radians, is accurate up to eight significant digits.

Built-in Functions

The name of a built-in function may not appear in an EXTERNAL statement, nor may it appear as the actual argument of a function or subroutine called by the main program. The following table lists the name and definition of each built-in function. In the table, a j denotes an integer expression and an x denotes a real expression.

NAME	DEFINITION
ABS(x)	Finds the absolute value of the real argument x.
IABS(j)	Finds the absolute value of the integer argument j.
AINT(x)	Performs the following computation: *result* = sign of x [largest integer $\leqq$ ABS(x)]
INT(x)	Performs the same function as AINT.
AMOD(x_1, x_2)	Performs the following computation: *result* = x_1 − AINT(x_1/x_2) * x_2
MOD(j_1, j_2)	Performs the following computation: *result* = $j_1 - (j_1/j_2) * j_2$

NAME	DEFINITION
AMAX0(j_1, j_2)	Chooses the largest value in the given set of integer arguments (j_1, j_2).
AMAX1(x_1, x_2)	Same as AMAX0 except that the arguments must be real.
MAX0(j_1, j_2)	Same as AMAX0 except that MAX0 is an integer function.
MAX1(x_1, x_2)	Same as AMAX0 except that the arguments must be real and MAX1 is an integer function.
AMIN0(j_1, j_2)	Chooses the smallest value in the given set of integer arguments (j_1, j_2).
AMIN1(x_1, x_2)	Same as AMIN0 except that the arguments must be real.
MIN0(j_1, j_2)	Same as AMIN0 except that MIN0 is an integer function.
MIN1(x_1, x_2)	Same as AMIN0 except that the arguments must be real and MIN1 is an integer function.
FLOAT(j)	Converts the value expressed by *j* to a real value.
IFIX(x)	Same as INT.
SIGN(x_1, x_2)	Performs the following computation: *result*=sign of x_2 [ABS(x_1)]
ISIGN(j_1, j_2)	Same as SIGN except that arguments are integer values.
DIM(x_1, x_2)	Performs the following computation: *result*=x_1−AMIN1(x_1, x_2)
IDIM(j_1, j_2)	Performs the following computation: *result*=j_1−MIN0(j_1, j_2)

Declarative Statements

Declarative statements provide the system compiler with information regarding certain properties of names appearing in other statements, such as the dimension of an array, or the type assumed by a variable.

Declarative statements are not executable and must precede the first executable statement in the user's program. They must not have statement numbers.

The two types of declarative statements are storage-allocating declarative statements and type-defining declarative statements.

Storage-Allocating Declarative Statements

There are three storage-allocating declarative statements: DIMENSION, COMMON, and EQUIVALENCE. These declarative statements provide for the allocation of storage for arrays and for the sharing of storage locations between programs or within a program. Constants, program names, function names, and subroutine names must not appear in a storage-allocating declarative statement. 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 $y_1, y_2, \ldots, y_n$ where: each subscripted variable can have one, two, or three subscripts (each subscript being an unsigned integer constant).

The DIMENSION statement provides the information necessary to allocate storage for arrays in the object program. It defines the maximum size of arrays; references to these arrays must never exceed the specified dimension.

If the DIMENSION statement is used, it must be the first declarative statement in the user's program.

Two examples of the use of the DIMENSION statement are:

```
DIMENSION ALPHA (10), BETA (5, 5, 15)
DIMENSION GAMMA (20, 30), X(5), NEXT (10, 10, 10)
```

COMMON Statement

General Form
COMMON $v_1, v_2, \ldots, v_n$

The COMMON statement causes the declared variables to be assigned to consecutive storage locations within a common storage area. Main programs and subprograms can share this common storage area.

Subscripted variables are not allowed in a COMMON statement. Each variable $v_1, v_2, \ldots, v_n$ must be the name of an array or a simple variable.

If a variable appearing in a COMMON statement is the name of an array, then this array must previously have been defined in a DIMENSION statement.

Two examples of the use of the COMMON statement are:

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

EQUIVALENCE Statement

General Form
EQUIVALENCE ($v_1, v_2, \ldots, v_n$) where: 1. *n* must be greater than or equal to 2. 2. Variables v_2 through v_n (sometimes called the "tenants") may not be subscripted. Each must be wholly contained within v_1 and each must not be described in a COMMON statement. 3. Variables v_1 through v_n may be array names. 4. Variable v_1 (sometimes called the "host") may have a single subscript if and only if v_1 is the name of an array. This subscript should specify the element of the array that is to be made equivalent to v_2 through v_n. 5. All variables must be enclosed within one set of parentheses.

The EQUIVALENCE statement causes all specified variables to be assigned to the same location in storage. An actual storage area is reserved only for the first variable, v_1; variable v_2 through v_n overlay the area reserved for v_1. Of all the variables specified, only v_1 may appear in other EQUIVALENCE statements (if so, it must appear only as v_1).

Two examples of the use of the EQUIVALENCE statement are:

```
EQUIVALENCE (F(3), E)
EQUIVALENCE (J(10), I, A, D)
```

NOTE: If the variable "E" in the first example had been defined as a single-dimensioned array of five elements, the given EQUIVALENCE statement would result in the following: E(1) is equivalent to F(3), E(2) is equivalent to F(4), E(3) is equivalent to F(5), etc.

Type-Defining Declarative Statements

There are three type-defining declarative statements: INTEGER, REAL, and EXTERNAL. The INTEGER and REAL statements provide the means to define explicitly the type of a variable or user-defined function, normally defined by the first letter of the name. (However, in a function subprogram, the name of the function may not appear in a type-defining statement.) The same name may not appear in both a REAL and INTEGER statement.

The EXTERNAL statement is used to declare that the name of a user-defined function or subroutine, or a library function (see "Library Functions") is being used as an argument in a subprogram call.

Constants and dimension information may not appear in any type-defining declarative statements.

INTEGER Statement

General Form
INTEGER $z_1, z_2, \ldots, z_n$ where: each z_i ($i=1, 2, \ldots, n$) may be the name of a variable, an array, or a user-defined function subprogram to be called. It cannot be the name of a reserved function (see "Reserved Functions") or the name of the program or subprogram in which the INTEGER statement appears.

The INTEGER statement specifies that the listed names are integer variables. An example of an INTEGER statement is:

```
INTEGER ALPHA, BETA, RESULT
```

REAL Statement

General Form
REAL $z_1, z_2, \ldots, z_n$ where: each z_i ($i=1, 2, \ldots, n$) may be the name of a variable, an array, or a user-defined function subprogram to be called. It cannot be the name of a reserved function (see "Reserved Functions") or the name of the program or subprogram in which the REAL statement appears.

The REAL statement specifies that the listed names are real variables. An example of a REAL statement is:

```
REAL IND, MATRIX, LAST
```

EXTERNAL Statement

General Form
EXTERNAL $u_1, u_2, \ldots, u_n$ where: each u_i ($i=1, 2, \ldots, n$) may be the name of a user-defined subprogram or the name of a library function (see "Library Functions").

The EXTERNAL statement specifies that each listed user-defined subprogram or library function is to be used as an argument in a call to a subprogram. Variable names must not appear in an EXTERNAL statement.

Example: In the following illustration, the main program, EXAMP, calls upon a function named UFN three times. In each call, the first argument is the name of another function because UFN has been constructed to call upon functions passed to it by the calling program. Hence, an EXTERNAL statement must be included in the main program to indicate that user-defined subprograms (i.e., ASSIGN and ASORT) and a library function (i.e., COS) are being used as arguments in subprogram calls.

```
CALLING PROGRAM                 SUBPROGRAM
PROGRAM EXAMP                   FUNCTION UFN (FUNC, X)
      .                               .
      .                               .
      .                               .
EXTERNAL COS, ASSIGN,                 .
  ASORT
      .                               .
first executable statement      first executable statement
      .                               .
      .                               .
Y=.2736519                            .
                                      .
A=UFN (COS, Y)                        .
      .                               .
      .                               .
      .                               .
B=UFN (ASSIGN, Y)                     .
      .                               .
      .                         UFN=FUNC(X)
      .                               .
C=UFN (ASORT, Y)                      .
      .                               .
      .                         RETURN
      .                               .
      .                         END
      .
END                             SAVE
```

NOTE: The mnemonic FUNC is a dummy name that assumes a new name (i.e., first COS, then ASSIGN, and finally ASORT) each time UFN is called. The mnemonic X is a dummy variable that assumes the value of the variable Y each time UFN is called.

Executable Statements

Executable statements form the essential part of the user's program. The arithmetic assignment statement,

all of the control statements and all of the input/output statements are considered as executable statements. (However, the input/output statements are described in a separate section called "Input/Output.")

Arithmetic Assignment Statement

General Form
v = e

This statement assigns a value to a specific variable; i.e., the value of an arithmetic expression, *e*, is determined and assigned to a variable, *v*.

The equals symbol in an arithmetic 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 arithmetic assignment statements.

When the variable to the left of the equals sign differs in type from the expression to the right of the equals sign, the following conventions apply:

1. If *v* is an integer variable and *e* is a real expression, then the result of *e* is truncated (not rounded) to the largest integer it contains and then converted to an integer variable, which, in turn, becomes the value of *v*.

2. If *v* is a real variable and *e* is an integer expression, then the result of *e* is converted to a real variable, which, in turn, becomes the value of *v*.

Some examples of arithmetic assignment statements are:

```
X = K
A(I) = B(I) - SIN(C(I))
I = 5*MU + 2
I = A**(B/C) + 10.
```

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

The unconditional GO TO statement causes the program to transfer control to the statement numbered *n*. An example is:

```
GO TO 3
```

Computed GO TO Statement

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

Control is transferred to the statement numbered $n_1, n_2, \ldots, n_p$, depending on whether the value of *i* at the time of execution is 1, 2, . . . , p, respectively. The value of *i* must be greater than 0, but less than or equal to *p*.

In the following example, if J is 3 at the time of execution, a transfer to the third statement of the list, namely statement 50, occurs:

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

IF Statement

General Form
IF (e) n_1, n_2, n_3

The IF statement results in one of the following:

1. If the value of *e* is less than zero, control is transferred to the statement numbered n_1.
2. If the value of *e* is equal to zero, control is transferred to the statement numbered n_2.
3. If the value of *e* is greater than zero, control is transferred to the statement numbered n_3.

In the following example, if (X-Y) is equal to zero when this statement is executed, control is transferred to the statement numbered 7:

```
IF (X-Y) 2, 7, 4
```

DO Statement

General Form
DO n i=m_1, m_2, m_3 where: each m_i must be greater than or equal to 1; m_1 must be less than or equal to m_2; and if m_3 is not stated, it is assumed to be equal to 1.

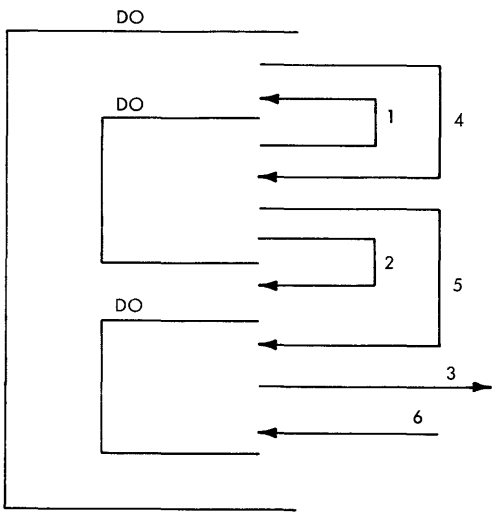
The DO statement results in the repeated execution of the statements that follow the DO, and up to and including the statement numbered *n*. Statement *n* must not be a DO, IF, GO TO, STOP, or RETURN statement; a CONTINUE statement can be used to satisfy this limitation. The "range of a DO" comprises all statements following the DO up to and including statement *n*.

The first time the statements are executed, $i=m_1$. (Execution does not occur if m_1 equals zero or is greater than m_2.) For each succeeding execution, *i* is increased by m_3; i.e., the second time, $i=m_1+m_3$, the third time, $i=m_1+2m_3$, etc. This pattern of execution takes place for all values of *i* that do not exceed m_2. Control then passes to the statement following statement *n*.

The values of i, m_1, m_2, and m_3 must not be set by any statement, other than the DO, within the range of the DO.

Among the statements in the range of a DO may be other DO statements; such a configuration is called "nesting." If the range of a DO includes 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.

Transfers into the range of a DO from a point outside its range are not permitted. Thus, in the following configuration, 1, 2, and 3 are permitted transfers, but 4, 5, and 6 are not.



Some examples of the use of the DO statement are:

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

CONTINUE Statement

General Form
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 c where: c is an optional one-digit through five-digit octal number that is to be printed at the terminal when the statement is executed.

Execution of a PAUSE statement causes a halt in the execution of the user's program. Execution can be resumed with the next executable statement by use of a START statement (see "START Statement"). An example of a PAUSE statement is:

```
PAUSE 3
```

STOP Statement

General Form
STOP c where: c is an optional one-digit through five-digit octal number that is to be printed at the terminal when the statement is executed.

Execution of a STOP statement causes a halt in the execution of the user's program. Execution can be resumed only by a START statement (see "START Statement") that indicates the point at which execution is to be resumed. Hence, the first executable statement after a STOP statement must be numbered. An example of a STOP statement is:

```
STOP 77777
```

Input/Output

This section provides the information necessary for transmitting data between the system and the terminal.

Introductory Information

Most of the input/output statements include a parameter k that, in conventional FORTRAN systems, indicates the specific input or output unit to be used. However, because the QUIKTRAN System operates on the theory that all input/output statements refer to input/output devices at a terminal (i.e., keyboard, printer, card reader, and card punch), the k has no meaning for the QUIKTRAN System. It is included for compatibility with conventional FORTRAN processors.

When QUIKTRAN executes a user's input statement, the system expects that the input records are to be entered from either the terminal keyboard or the terminal card reader. The actual input device used is determined by the way the user has set the terminal switches (see the section "Equipment" of this publication). Regardless of the input device chosen, all input data also appears on the terminal printer.

When QUIKTRAN executes a user's output statement, the system always transmits the output data to the terminal printer. However, the user can have his output also directed to other devices at the terminal by using the method discussed in the section "AUXOP Statement."

Size of Input/Output Records

The size of an input or output record varies with the device chosen. Keyboard input records and printer out-

put records are printed lines; each line contains a maximum of 120 characters. Card reader input records and card punch output records are punched cards; each card contains a maximum of 80 characters (see "Procedures for Card Input" for further information on the size of a card input record). Note, however, that if the user requests that the system recompose his program on punched cards, a maximum of 72 characters per card will be punched.

For the remainder of this section, references to records as printed lines also apply to punched cards unless specifically stated otherwise.

List Specifications

The statements that cause transmission of information require a list of quantities to be transmitted. This list must be in the same order as that of the words of information (for input). For output, the list determines the order on the output medium.

The following example illustrates the formation and meaning of an input/output list:

```
A, B(3), (C(I), D(I, K), I=1, 10), ((E(I, J), I=1,
  10, 2), F(J, 3), J=1, K)
```

If this list is used with an output statement, the information will be transmitted to the terminal in the following order:

```
A, B(3), C(1), D(1, K), C(2), D(2, K), . . . , C(10),
  D(10, K),
E(1, 1), E(3, 1), . . . , E(9, 1), F(1, 3),
E(1, 2), E(3, 2), . . . , E(9, 2), F(2, 3), . . . , F(K, 3).
```

Similarly, if this list is used with an input statement, the successive words, as they are read from the terminal, are placed into storage in the above sequence.

The list reads from left to right, with repetition for variables enclosed within parentheses. The list items are separated by commas. Only subscripted or nonsubscripted variables or an implied DO may appear in an input/output list. The execution is exactly that of a DO loop, as though each left parenthesis (except subscripting parentheses) were a DO, with indexing given immediately before the matching right parenthesis and with the DO range extending up to that indexing information.

An implied DO is best defined by an example. In the preceding input/output list, the list item (C(I), D(I, K,), I=1, 10) is an implied DO. The range of the implied DO must be clearly defined by parentheses. A constant may appear in an input/output list only as a subscript or an indexing parameter. Indexing information, as in a DO, consists of three constants or integer variables, and the last of these may be omitted, in which case it is assumed to be 1.

For an input list of the form K, (A(I), I=1, K), where an indexing parameter (i.e., K) appears in the list earlier than its actual use as an indexing parameter, the indexing is carried out with the newly read-in value. However, this element must previously have been "set" (see "Concept of 'Set' and 'Used' Variables").

Any number of quantities may appear in a single list. During a read operation, the list controls the quantity of data read. If a record contains more quantities than there are in the list, only the number of quantities specified in the list are transmitted, and any remaining quantities are ignored. Conversely, if the list contains more quantities than are given on one BCD input record (as defined by the FORMAT statement), additional records are read.

Input/Output of Entire Arrays

When input/output of an entire matrix is desired, an abbreviated notation can be used in the list of the input/output statement. Only the name of the array need be given; the indexing information can be omitted. Thus, if A has previously been listed in a DIMENSION statement, the statement:

```
READ (5, 10)A
```

is sufficient to read in all of the elements of the array A.

The elements read in by this notation are stored in accordance with the description of the arrangement of arrays in storage. Arrays are read or written in columnar format, with the first of their subscripts varying most rapidly and the last varying least rapidly.

Carrying the illustration one step further, assume that A had been previously defined by the following DIMENSION statement:

```
DIMENSION A(3, 2)
```

Then, the following READ statement would be equivalent to the READ (5, 10)A statement shown above:

```
READ (5, 10) ((A(I, J), I=1, 3), J=1, 2)
```

and each one would cause data to be transmitted in the following order:

```
A(1, 1), A(2, 1), A(3, 1), A(1, 2), A(2, 2), A(3, 2)
```

Format Specifications

Most of the BCD input/output statements required, in addition to a list of quantities to be transmitted, reference to a FORMAT statement that describes the type of conversion to be performed between the internal machine language and the external notion for each quantity in the list.

General Form
FORMAT (s_1, s_2, . . . , s_n/s'_1, s'_2, . . . , s'_n/ . . .) where: each field, s_i, is a format specification.

The FORMAT statement specifies the types of data conversion to be performed.

1. FORMAT statements can be placed anywhere in the source program. Each FORMAT statement must be given a statement number.

2. The FORMAT statement indicates, among other things, the maximum size of each record to be transmitted. In this connection, it must be remembered that the FORMAT statement is used with the list of some particular input/output statement, except when a FORMAT statement consists entirely of H and/or X fields. In all other cases, control in the object program switches back and forth between the list (which specifies whether data remains to be transmitted) and the FORMAT statement (which contains the specifications for the transmission of that data).

3. Slashes are used to terminate records. In each case, the record length specified must be no longer than 120 characters. For example, FORMAT (3F9.2, 2F10.4/8E14.5) specifies that the first, third, fifth, etc., records have the format (3F9.2, 2F10.4) and that the second, fourth, sixth, etc., records have the format (8E14.5).

4. During the input/output of data, the object program interprets the FORMAT statement to which the relevant input/output statement refers. When a specification for an A-, E-, F-, I-, or O-type field is found, and list items remain to be transmitted, input/output takes place according to the specification, and interpretation of the FORMAT statement resumes. If no items remain, transmission ceases and execution of that particular input/output statement is terminated. Thus, a BCD input/output operation ends when there are no items remaining in the list.

Another example of a FORMAT statement is:

```
FORMAT (I2/(E12.4, F10.2))
```

Numerical Fields

Four forms of conversion for numerical data are available:

INTERNAL	TYPE	EXTERNAL
Real	E	Real, with exponent
Real	F	Real, without exponent
Integer	I	Integer
Octal Integer	O	Octal Integer

These types of conversion are specified in the forms:

Ew.d, Fw.d, Iw, and Ow

where:

E, F, I and O

represent the type of conversion

w

is an unsigned integer constant that represents the field width for converted data; this field width may be greater than required to provide spacing between numbers.

d

is an unsigned integer or a zero that represents the number of positions of the field that appears to the right of the decimal point, not including the E exponent field, if present.

The format of a numerical field is specified by giving, from left to right (beginning with the first character of the field):

1. The control character (E, F, I, or O) for the field.
2. The width (w) of the field. Leading zeros in the integer part of an output number are suppressed, and a blank or a minus sign is placed in front of the first integer digit. If the entire integer part of the number is zero, the digit is zero. The specified width can be greater than required to provide spacing between numbers.
3. The number of positions (d) of the decimal fraction that appears to the right of the decimal point for E- and F-type conversion. If d is greater than the maximum number of digits permitted by the machine (i.e., 9 for E- and F-type conversion), the maximum number of digits are carried and unused digits are truncated.

For example, the statement FORMAT (I2,E12.4, O8,F10.4) might cause the following to be printed:

```
I2 E12.4        O8      F10.4
27b-0.9321Eb0257734276bbb-0.0076
```

where:

b indicates a blank space

Specification for successive fields are separated by commas. A format specification that provides for more characters than the maximum input or output unit record size should not be given. Thus, a format for printed output should not provide for more characters per line (including blanks) than may be printed on one line by the terminal printer.

E-, F-, I-, and O-Conversion: E-conversion results in the transmission of w characters containing a real number and its exponent field, e.g., 5.02E2. The exponent, which must be used with E-conversion, is the power of 10 to which the number must be raised to obtain its true value. The exponent is written with an E, followed by a minus sign if the exponent is negative, or a plus sign or a blank, if the exponent is positive, and then followed by the two numbers that are exponent. For example, the number .002 is equivalent to the number .2E-02.

F-conversion results in the transmission of w characters containing a real number only.

I-conversion results in the transmission of w characters containing an integer number of up to ten decimal digits.

When an output number converted by E-, F-, or I-conversion requires more spaces than are allowed by the field width w, the excess on the high-order side is lost and no rounding occurs. When the number requires

fewer than w spaces, the leftmost spaces are filled with blanks. When the number is negative, the space preceding the leftmost digit contains a minus sign if sufficient spaces have been reserved.

O-conversion results in the transmission of w characters of octal information. If $w>12$, only the twelve rightmost characters are transmitted and w-12 blanks precede the field (output) or w-12 preceding characters are skipped (input). If $w\leqq12$, the rightmost w characters of the word are transmitted (output) or the next w characters are right-adjusted in the word and the word is filled out on the left with zeros (input).

The field width w for E- and F-conversion of output must include a space for the decimal point and a space for the sign. Thus, for E-conversion, $w\geqq d+7$, and for F-conversion, $w\geqq d+3$.

Information to be transmitted with O-conversion may be given either a real name or an integer variable name; information to be transmitted with E- and F-conversion must have real names; information to be transmitted with I-conversion must have an integer name. The names must be used as specified above; any other practice is invalid.

Alphameric Fields

QUIKTRAN provides two specifications to transmit alphameric information: Aw and nH. Both result in storing the alphameric information internally in BCD form.

1. The specification Aw causes w characters to be read into, or written from, a variable or array name, without conversion.

2. The specification nH introduces alphameric information into a FORMAT statement.

The basic difference between A- and H-conversion is that information handled by A-conversion is given a variable name or an array name and, hence, can be referred to by this name for processing and modification, whereas information handled by H-conversion is not given a name and, therefore, cannot be referred to or manipulated in storage in any way.

A-Conversion: The variable name to be converted by A-conversion must conform to the normal rules for naming QUIKTRAN variables; the name may be any type.

1. Input: nAw means that the next n successive fields of w characters each are to be stored as BCD information. If $w>6$, only the six rightmost characters are significant. If $w<6$, the characters are left-adjusted, and the word is filled out on the right with blanks.

2. Output: nAw means that the next n successive fields of w characters each are to be the result of transmission from storage without conversion. If $w>6$, only six characters of output are transmitted, preceded by w-6 blanks. If $w<6$, the w leftmost characters of the word are transmitted.

H-Conversion: The specification nH is followed in the FORMAT statement by n alphameric characters, as seen in the following:

```
31HbTHISbISbALPHAMERICbINFORMATION
```

Blanks are considered alphameric characters and must be included as part of the count n. The effect of nH depends on whether it is used with input or with output.

1. Input: The n characters are extracted from the input record and replace the n characters immediately following the character H in the FORMAT specification.

2. Output: The n characters following the character H in the specification, or the characters that replaced them, are written as part of the output record.

For example, the statement, FORMAT (4HbXY=, F8.3,A8), might produce the following lines:

```
XY=b-93.210bbbbbbbb
XY=9999.999bbOVFLOW
XY=bb28.768bbbbbbbb
```

This example assumes that there are steps in the source program that read the data (i.e., OVFLOW), store this data in the word to be printed (as six BCD characters) in the format A8 when overflow occurs, and store six blanks in the word when overflow does not occur.

Blank Fields—X-Conversion

The specification nX affects an input or an output record, as follows:

1. Input: nX causes n characters in the input record to be skipped.

2. Output: nX causes n blanks to be introduced into the output record.

Repetition of Field Format

It may be desired to transmit n successive fields within one record in the same format. This may be done by specifying n, an unsigned integer constant, before A, E, F, I, or O. The following format field will then be repeated n times. Thus, the field specification 3E12.4 has the same effect as the specification E12.4, E12.4, E12.4.

Repetition of Groups

A limited parenthetical expression is permitted to enable repetition of data fields according to certain format specifications within a longer FORMAT statement specification. The first character in the expression specifies the number of repetitions and is known as the *group count*. Thus, FORMAT (2(F10.6,E10.2),I4) is equivalent to FORMAT (F10.6,E10.2,F10.6,E10.2,I4).

Multiple-Record Formats

To deal with a block of more than one record, a FORMAT specification may have several different one-record formats, each separated by a slash (/) to indicate the beginning of a new record.

Thus, the statement FORMAT (3F9.2,2F10.4/8E14.5) would specify a multirecord block in which records 1, 3, 5, . . . , have the format (3F9.2,2F10.4), and records 2, 4, 6, . . . , have the format (8E14.5).

If a multirecord format is desired in which the first *n* records are to have particular formats and all remaining records are to have another format, the specification for the latter should be enclosed in a second pair of parentheses; for example:

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

Thus, in the above example, the first record has the format (I2,3E12.4); the second record has the format (2F10.3,3F9.4); all remaining records needed to satisfy the list of the input/output statement have the format (10F12.4).

NOTE: The list of an input/output statement always controls the number of items (and, hence, the number of records) that will be transmitted. If the FORMAT statement specifies more items than the list, data transmission is terminated when the list is exhausted. If the list specifies more items than the FORMAT statement, the FORMAT statement is recycled; i.e., a new record is read and the format is interpreted from the last left parenthesis of the FORMAT statement. However, assuming that the list specifies more than one record, it is the FORMAT statement that indicates at what point a new record is to be transmitted. Those sequences in a FORMAT statement that lie between a left parenthesis and a slash, a slash and a slash, an initial left parenthesis and a final left parenthesis, or a slash and a final left parenthesis are said to determine the format for one complete record.

Blank Records: Blank records may be introduced into a multirecord FORMAT statement by listing consecutive slashes. When *n* consecutive slashes are encountered at the end of a FORMAT statement, they have the following effect: for input, *n* records are skipped; for output, *n* blank records are written. When *n* consecutive slashes are encountered in the middle of a FORMAT statement, then for input, no further items are taken from the current record and n—1 additional records are skipped; for output, after the current record item has been written, n—1 blank records are written.

Data Input to the Object Program

Data input to the object program is prepared according to the following specifications:

1. The data must correspond in order, type, and field, to the field specifications in the FORMAT statement.

2. Plus signs can be omitted, or they can be indicated by a blank.

3. Blanks in numerical fields are regarded as zeros.

4. Numbers for E- and F-conversion can contain any number of digits, but only the high-order digits are retained. For E- and F-conversion, the number is rounded to the eight high-order digits of accuracy.

To permit economy in typing and punching, certain relaxations in input data format are permitted:

1. Numbers for E-conversion need not have four columns occupied by the exponent field. The start of the exponent field must be marked by the E or, if that is omitted, by a + (plus) or a − (minus), not a blank. Thus, E2, E+2, +2, +02, and E+02 are all permissible exponent fields.

2. Numbers for E- and F-conversion need not have the decimal point punched. If it is not punched, the FORMAT specification supplies the number of decimal places expected. For example, the number −09321+2 with the specification E12.4 is treated as though the decimal point had been punched between the 0 and the 9. If the decimal point is punched on the card, its position overrides the position indicated in the FORMAT statement.

Execution of Input/Output Statements

Input Execution

When an input statement is executed, the system asks for entry of the input data (by typing Innn, where *nnn* is the statement number of the associated FORMAT statement). The user enters the record and signals that the end of the record has been entered by transmitting EOB (if the card reader is used for input, an EOB can be transmitted in any one of three different ways; see "Procedures for Card Input" for additional information). The system then resumes control, accepting the record according to the input specifications. If more than one input record is required to satisfy the list, the system requests another input record. This process continues until the user's input specifications have been satisfied, at which point the system resumes with the execution of the rest of the program.

NOTES:

1. If an EOB is transmitted before an input record has fully satisfied the requirements of the FORMAT statement, an error message is printed at the terminal. Conversely, if an input record is longer than that required by the FORMAT statement, the excess characters are ignored by the system.

2. If, for some reason, the execution of an input statement is only *partially* completed, it might be expected that all the variables in the list of the input statement would remain unchanged. However, this is not the case; all variables appearing in the input list prior to the point of termination, will have new values assigned to them. Of these variables, only those

which were previously "set" may now be "used" (see "Concept of 'Set' and 'Used' Variables"); those that were "not set" remain "not set."

Output Execution

When an output statement is executed, the system transmits the output data to the terminal in accordance with the user's specifications. If more than one line of output is required, the transmission is continuous (one record per line). The first line of output is identified by Onnn (where *nnn* is the statement number of the associated FORMAT statement). Subsequent lines of output are sequentially numbered (beginning with the number 2) in print positions 7 through 12. The output operation continues until all of the data specified by the output list has been transmitted. Execution of the user's program then resumes with the next executable statement.

Periodically, during the flow of output data to the terminal, the PROCEED light will be set on by the system (this occurs only if the KEYBOARD switch at the terminal is set to the SEND position). During these periods, the user can interrupt the flow of output data if he desires (see "Interrupting Execution").

NOTE: In the QUIKTRAN System, the terminal printer carriage cannot be controlled by the user's program. Terminal printer output is always single spaced or double spaced depending on the manner in which the carriage has been set up.

Input/Output Statements

The QUIKTRAN input/output statements do not provide for the reading and writing of data in binary form. All input and output is considered to be BCD information.

PRINT Statement

General Form
PRINT n, list where: *n* is the statement number of a FORMAT statement. If *n* is 0, QUIKTRAN assumes that there is no FORMAT statement. *list* specifies the quantities to be transmitted.

This statement can be used with or without a FORMAT statement. If *n* is a statement number, then the values of the items on the list are written on the terminal printer in accordance with FORMAT statement *n*.

If *n* is 0, then the values of the items in the list will be written on the terminal printer either in accordance with the specifications set by the system (i.e., E15.8 for real values and I12 for integer values) or in accordance with the specifications set by the user in an EDIT statement (see "EDIT Statement"). Only numerical values can be transmitted for this use of the PRINT statement.

Some examples of the use of the PRINT statement are:

```
PRINT 2, (AC(I), I=1, 10, 2)
PRINT 0, TEMP, LAST, X(J)
```

PUNCH Statement

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

This statement causes the items in the list to be written on the terminal printer (and the punch unit, only if selected by an AUXOP statement; see "AUXOP Statement") in the format specified by statement *n*. If this statement is used, its associated FORMAT statement must not specify more than 80 characters per record, regardless of which output unit is selected.

Some examples of the use of the PUNCH statement are:

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

READ(k, n) Statement

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

This statement causes data to be read from the currently selected input unit (keyboard or card reader) and to be assigned as the values of the variables in the list. Some examples of the use of this statement are:

```
READ (7, 3) M, A(J)
READ(5, 8) JOBNO, X, Y
```

READ n Statement

General Form
READ n, list where: *n* may be the statement number of a FORMAT statement. If *n* is 0, QUIKTRAN assumes that there is no FORMAT statement. *list* specifies the quantities to be transmitted.

This statement can be used either with or without an appropriate FORMAT statement. When used with a FORMAT statement, data is read in from the currently selected input unit (keyboard or card reader) and is assigned as the values of the variables in the list. If this statement is used, the list and its associated FORMAT statement (if any) must not specify more than 80 characters per record.

Nonformatted Input: If *n* is 0, this statement causes only numerical fields to be read from the currently

selected input device and their values to be assigned according to the list specification.

For the purposes of this discussion, a numerical field is one that consists of a string of numerical characters preceded by a "separator" character (i.e., + − / =) and followed by either another separator character or a nonnumerical character.

A string of numerical characters can consist of one or more decimal digits 0 through 9 and (if necessary) either a decimal point or a decimal point and an E (in proper combination for E-conversion).

QUIKTRAN always assumes that column 1 of each input record is preceded by a "separator" character. Thus, a string of numerical characters beginning in column 1 would be a valid numerical field.

The "separator" characters + and − also serve to indicate the sign of a numerical field. An unsigned numerical field is assumed to be positive.

The foregoing discussion is illustrated by the following examples:

1. Statement: READ 0,I,J,K,W
   Input: TAB/5,ENT/3,HRS=10,WT=−15.6
   Result: variable 'I'=+5
   variable 'J'=+3
   variable 'K'=+10
   variable 'W'=−15.6
2. Statement: READ 0,I,J,C
   Input: IN A5/IN B/6IN C=7 IN D−54
   Result: variable 'I'=+6
   variable 'J'=+7
   variable 'C' = −54
3. Statement: READ 0,A,I,X(J)
   Input: ////// //1.2/2/−567.5E−10
   Result: variable 'A'=+1.2
   variable 'I'=+2
   variable 'X(J)'=−.0000005675

WRITE (k, n) Statement

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

This statement causes the values of the items in the list to be written on the terminal printer. Some examples of the use of this statement are:

```
WRITE (7, 5) K, A(L)
WRITE (5, 3) JOBNO, ANS, X
```

BACKSPACE, END FILE, and REWIND Statements

These statements have no meaning in the QUIKTRAN language. However, they are not rejected by the system. They are retained as part of the user's program but they produce no effect.

Operating Statements

The operating statements enable the user to control, modify, test, and display programs. Because of their nature, *operating statements are not retained as part of the user's program.* However, some operating statements also function as system subroutines (see "Program Called Services"), and when so used they are retained as part of the user's program.

The acceptable forms and purpose of each statement are discussed later in this section. The following table gives the meaning of the symbols used to identify the arguments in the operating statements:

SYMBOL	TYPE OF ARGUMENT
d	real constant set by the user; its value must lie between 0.1 and 10.0 inclusive
g	line number
h	line number or statement number; if a statement number is substituted for *h*, it can be adjusted in the form n+c (where *c* is an integer constant whose value lies between 1 and 10 inclusive). For example, if the user wished to refer to the third statement after the statement identified by number 25, then the expression 25+3 could be substituted for *h*.
n	statement number
u	name of a program
v	nonsubscripted variable name

Any argument(s) associated with an operating statement must be enclosed within one set of parentheses. If more than one argument is used with a statement, the arguments must be separated by a comma. The argument(s) may specify a particular program name, variable, statement, or region of a program. If a region is to be specified, two arguments indicating the boundary points of the region must be used. All boundary points are considered as part of a region.

Control Statements

Program Control Statements

There are four program control operating statements: COMMAND, LOAD, SAVE, and PURGE. If a SAVE, LOAD, or PURGE statement is used, the user's identification code must be active or else the statement will not take effect (see "Terminal Commands").

COMMAND Statement

General Form
COMMAND

The COMMAND statement places the terminal into the command mode. When the COMMAND statement is used, the active program image is destroyed. If an active program is to be recalled later, the COMMAND statement must be preceded by a SAVE statement (see "SAVE Statement").

LOAD Statement

General Form
LOAD(u)

The LOAD statement can be used only when the terminal is in the command mode; an error is indicated if it is used when the terminal is in the program mode. The LOAD statement causes the terminal to enter the program mode while specifying that a program or subprogram is to be placed in active status. Thus, the argument *u* must be the name of any program, subroutine, or function in the user's library. An example of the use of the LOAD statement is:

```
LOAD (ZMULT)
```

As a result, the program that had been "saved" (see "SAVE Statement") under the name of ZMULT would be made active at the terminal.

SAVE Statement

General Forms
SAVE SAVE (u)

The SAVE statement places the currently active program or subprogram into the user's library. If no argument is specified, the current program name is used. If an argument is specified, then the currently active program is assigned to the user's library under the name given by the argument (i.e., *u*). However, the name in the program defining statement of the "saved" program remains unchanged.

After a SAVE statement has been executed, the program or subprogram that was "saved" remains active at the terminal under its original name. For example, assume that the currently active program is named ALPHA; then the statement:

```
SAVE (BETA)
```

would cause the active program to be added to the user's library under the name of BETA. Any existing program in the library that had been occupying the space reserved for the name BETA is thus replaced. If there had not been space reserved for the name BETA, then space is created when the SAVE (BETA) statement is executed. However, the active image of the program is still called ALPHA, so that if a subsequent SAVE without an argument is given, the currently active program is "saved" under the name of ALPHA.

The name under which a program is "saved" may be the argument of a future CALL or LOAD statement.

PURGE Statement

General Form
PURGE (u)

The PURGE statement removes the named program or subprogram (i.e., *u*) from the user's library. After this statement is used, the specified program or subprogram is no longer available to the user and, thus, must not be the argument of a future CALL or LOAD statement. An example of the use of the PURGE statement is:

```
PURGE (PROGNM)
```

Execution Control Statements

The four execution control operating statements are: START, RESET, CLEAR, and XEQER.

START Statement

General Forms
START(0) START(h) START

The START statement initiates execution of the currently active program or subprogram. There are three forms in which this statement may be used:

1. The START(0) statement initiates execution of a program or subprogram from its beginning (i.e., from the first executable statement).

2. The START(h) statement initiates execution at the particular statement specified by *h* (where *h* is a line or statement number).

3. The START statement with no argument resumes execution of a program after a termination of execution has occurred. (This statement must not be used to initiate execution; in other words, it can be used only to resume execution after an interruption has occurred.) Execution resumes at the next executable statement following the point of termination. This statement can be used in all cases of termination except when termination was caused by a STOP or END statement, in which case a START(h) statement must be used to resume execution.

An example of the use of a START statement is:

```
START(143.)
```

where the argument, 143., is the line number of the first statement to be executed.

One of the following status words is printed at the terminal to indicate the cause of a termination of execution:

Status Word PAUSE: Indicates that execution was terminated because a PAUSE statement was encountered in the program (see "PAUSE Statement").

Status Word STOP: Indicates that execution was terminated because a STOP statement was encountered in the program (see "STOP Statement").

Status Word HALT: Indicates that execution was terminated because the last physical statement of the program was encountered.

Status Word BREAK: Indicates that execution was interrupted by the user (see "Interrupting Execution"). This status word is accompanied by a message specifying the line number of the last statement executed.

Status Word XEQER: Indicates that termination occurred because an error was encountered while the program was being executed. This status word is accompanied by a message specifying the line number of the statement that caused the error along with the nature of the error.

RESET Statement

General Form
RESET

The RESET statement cancels the effects produced by the execution of the active program or any of its parts. The RESET statement causes the system to regard all statements as "not used" and all variables as "not set" and "not used" (see "Concept of 'Set' and 'Used' Variables").

The service provided by the RESET statement can be made a permanent part of the user's program because this service also functions as a system subroutine (see "Program Called Services" for details).

CLEAR Statement

General Form
CLEAR

The CLEAR statement cancels the effect of all previous test statements (except a TRAIL statement – see "Test Statements"), and also causes the system to regard all statements as "not used" and all variables as "not set" and "not used." In other words, specifying the CLEAR statement is equivalent to specifying SNAPX, TRAPX, GUARDX, STEPX, and RESET. Note that the CLEAR statement also cancels the effect of test services that have been in use as system subroutines.

The service provided by the CLEAR statement can be made a permanent part of the user's program because this service also functions as a system subroutine (see "Program Called Services").

XEQER Statement

General Forms
XEQER(FINISH) XEQER

The XEQER statement specifies possible options for the system when an error is encountered in program execution. There are two forms in which this statement may be used:

1. The XEQER (FINISH) statement specifies that, if an error is encountered during execution, execution is terminated, the terminal reverts to the command mode, and the user's identification code is deactivated.
2. The XEQER statement with no argument specifies that each error encountered during execution is to be treated according to the standard procedure; i.e., each error returns control to the user with an appropriate message accompanying the XEQER status word (see "START Statement" for a discussion of this status word).

Form Control Statements

The three form control operating statements are: AUXOP, DELTA, and EDIT.

AUXOP Statement

General Forms
AUXOP(PUNCH) AUXOP(PR2) AUXOP

The AUXOP statement can be used to specify that an additional terminal device is to be used for registering output (output is also always registered on Printer 1). There are three forms in which this statement may be used:

1. The AUXOP(PUNCH) statement specifies that the card punch is to be used for registering output.
2. The AUXOP(PR2) statement specifies that the auxiliary printer (i.e., Printer 2) is to be used for registering output.
3. The AUXOP statement with no parameters specifies that output transmission is to revert to the standard; i.e., output is to be sent only to Printer 1.

Note that for the first two forms to take effect, the terminal switches for PRINTER 2 or PUNCH 1 must be set to RCV. This setting should be made immediately after the corresponding AUXOP statement has been entered. At the time that the third form of the AUXOP statement is given, the switch for the auxiliary device that was in use should be in the OFF position.

Only one auxiliary device can be in use at one time. The publication *System Operation Reference Manual; IBM 1050 Data Communication System,* Form A24-3020, contains detailed information about these auxiliary devices.

DELTA Statement

General Forms
DELTA(d) DELTA(d_1, d_2) DELTA

The DELTA statement specifies an increment for the system to use in generating line numbers. Initially, the system uses a standard increment of 1.0 for generating line numbers in the READY status and .1 for generating line numbers in the ALTER status. There are three forms in which this statement may be used:

1. The DELTA(d) statement specifies an increment (i.e., *d*) to be used by the system for generating line numbers in the READY status.
2. The DELTA(d_1, d_2) statement specifies the following:
   - d_1 is an increment to be used by the system for generating line numbers in the READY status. If the standard increment is desired, d_1 must be specified as 0 or 1.
   - d_2 is an increment to be used by the system for generating line numbers in the ALTER status.

   For example, if the user wants the standard increment in the READY status but an increment of .5 in the ALTER status, the statement, DELTA (0, .5), would be used.
3. The DELTA statement with no arguments specifies that the system is to revert to the standard increments.

EDIT Statement

General Forms
EDIT(f) EDIT(f_1, f_2) EDIT where: f, f_1, and f_2 are format specifications for numerical fields.

The EDIT statement is used to specify an output format for all real and integer variables that are not under control of a FORMAT statement.

Initially, each terminal is set to the standard formats of E15.8 for real variables and I12 for integer variables. These formats are used for the output of all variables not under control of a FORMAT statement. The formats specified in the EDIT statement replace the standard formats for the terminal (or those specified in a previous EDIT statement).

If one argument appears after the EDIT (i.e., if the EDIT (f) form of the statement is used), then the argument, *f*, may be either a real or integer format specification.

If two arguments appear after the EDIT (i.e., if the EDIT(f_1, f_2,) form of the statement is used), then one argument must be a real format specification and the other an integer format specification. In other words, if f_1 is a real format specification, f_2 must be an integer format specification; if f_1 is an integer format specification, f_2 must be a real format specification.

If no arguments appear in an EDIT statement, then the system reverts to the standard format specifications.

The service provided by the EDIT statement can be made a permanent part of the user's program because this service also functions as a system subroutine (see "Program Called Services" for details).

The EDIT statement may be used in both the command and program modes. However, a particular EDIT statement is effective only for the mode in which it has has been specified.

An example of an EDIT(f_1,f_2) statement is:

```
EDIT(I5, F8.2)
```

Modification Statements

These operating statements provide a means for modifying the program. The user employs these statements to add, change, delete, and renumber program statements. The three modification statements are ALTER, ALTERX, and NUMBER.

ALTER Statement

General Forms
ALTER(h) ALTER(0, h) ALTER(h_1, h_2)

The ALTER statement permits the user to insert, replace, or delete a program statement or a series of program statements.

A particular ALTER statement, and any subsequent program statements associated with it, is known as an "alter sequence." Each program statement in an "alter sequence" is assigned a line number. These line numbers are incremented either by the standard increment of .1 or by the increment specified in a previous DELTA statement (see "DELTA Statement"). A particular "alter sequence" is terminated by another ALTER statement (in which case another "alter sequence" would be initiated) or by an ALTERX statement (see "ALTERX Statement").

An ALTER statement places the terminal into the ALTER status. The terminal remains in this status until an ALTERX statement is used, at which point the terminal reverts to the READY status.

NOTE: The following is an exception to this rule: when the last statement in a program is deleted, statements may be inserted in the ALTER status up to the highest line number that was reached previously. If and

when this point is reached in the ALTER status, the terminal will *automatically* revert to the READY status.

There are three forms in which the ALTER statement can be used:

1. The ALTER(h) form of the statement specifies that the subsequent statements in the "alter sequence" are to be inserted into the program beginning at line or statement number *h* (where *h* must not be associated with another statement in the program). For example, if a statement were to be inserted between the statements identified by line numbers 149. and 150., an ALTER(149.1) statement would enable this to be done, provided that 149.1 was not already associated with a statement.

2. The ALTER(0, h) form of the statement specifies that the region of the program, beginning with the first statement after the program-defining statement and ending with the statement identified by *h*, is to be deleted. Any subsequent program statements in the "alter sequence" will replace the deleted region with the first inserted statement assigned the line number of the first deleted statement.

3. The ALTER(h_1, h_2) form of the statement specifies that the region of the program, beginning with the statement identified by h_1 and ending with the statement identified by h_2, is to be deleted.

NOTE: If h_2 is equal to h_1, then only one statement will be deleted.

Any subsequent statements in the "alter sequence" will replace the deleted region beginning at h_1.

Program defining statements cannot be "altered."

Status Word ALTER: After an ALTER statement has been entered, the system prints the status word ALTER at the terminal. The user may then begin typing any subsequent statements of the "alter sequence."

ALTERX Statement

General Form
ALTERX

The ALTERX statement terminates an "alter sequence." Immediately after the entry of an ALTERX statement, one or both of the following may occur:

1. If the first executable statement in the user's program had been entered at some point prior to the entry of the ALTERX statement, then any storage required by the "alter sequence" would be allocated at this time. Any errors encountered during this process are listed at the terminal accompanied by the status word ERROR.

2. If an END statement had been entered at some point prior to the entry of the ALTERX statement, a check for completeness errors is made at this time. If any completeness errors are found, they are listed at the terminal accompanied by the status word ERROR.

After all required storage allocation and error checking has been done, the terminal reverts to the READY status. (For a detailed discussion of the above procedures, see the section "Status Word ERROR.")

"Altering" the Type of a Variable: If reference to a variable has not been made, the type of that variable can always be changed by an appropriate INTEGER or REAL statement. However, if it is necessary to change the type of a variable after reference to it has been made, the following method may be used:

1. Delete all references to the variable by way of an "alter sequence."
2. Terminate the "alter sequence" by an ALTERX.
3. Through another "alter sequence," insert the appropriate type-defining statement at the desired point in the program.
4. As part of that same "alter sequence," or of another, if necessary, replace the previously deleted references to the variable.

An analogous procedure can be used if it is necessary to change a variable to an array or an array to a variable.

NUMBER Statement

General Forms
NUMBER NUMBER(g) NUMBER(g, d)

The NUMBER statement causes the system to renumber sequentially (by line numbers) all statements in the active program. There are three forms in which this statement can be used:

1. The NUMBER statement with no arguments specifies that the line number of the first statement is to remain unchanged, but all subsequent statements are to be assigned new line numbers. The new line numbers will be determined either by the standard increment of 1.0 or by an increment specified in a DELTA statement.

2. The NUMBER(g) statement specifies that the first statement in the program is to be assigned line number *g* and all subsequent statements are to be renumbered according to either the standard increment of 1.0 or the increment specified in a DELTA statement. If *g* is specified as 0, this statement will produce the same results as those described in item 1 above.

3. The NUMBER(g, d) statement specifies that the first statement in the program is to be assigned the line number *g* and all subsequent statements are to be renumbered according to the increment specified by *d*. If *g* is specified as 0, then the line number of the first statement remains unchanged and all sub-

sequent line numbers will be changed according to the increment *d*.

When a NUMBER statement is executed, the system automatically produces a new program listing which shows the effect of the renumbering. The user can interrupt this listing process if he desires (see "Interrupting Execution"). Partial program listings can be obtained by using a LIST statement (see "LIST Statement").

Test Statements

The test operating statements provide a means by which the user can selectively control the execution of his program. He can closely follow the flow of the program, and be kept informed of all changes in the values of variables.

There are five test statements: SNAP, TRAP, GUARD, STEP, and TRAIL. The service provided by each of these statements remains in effect only as long as the user wants it to. An X appended to a test statement will cancel its effect. That is, a SNAPX statement cancels the effect of the corresponding SNAP statement; a TRAPX statement cancels the effect of the corresponding TRAP statement, etc. These X-type statements are constructed in the same way that the parent statements are constructed (i.e., an X-type statement must have the same number of arguments as its parent statement). The same conventions apply to the arguments in an X-type statement as they do to those in the parent statement.

The service provided by each of the test statements (X-type included) can be made a permanent part of the user's program because each can also be used as a system subroutine (see "Program Called Services").

SNAP Statement

General Forms
SNAP SNAP(h) SNAP(h_1, h_2) SNAP(v)

Each form of the SNAP statement essentially performs the same service; i.e., each causes the printing of the value of the leftmost variable of an arithmetic assignment statement whenever that value changes during execution. The forms differ only in the range over which they remain in effect. This difference allows the user to determine what regions of his program are to be serviced by the SNAP statement.

The table below gives each form of the SNAP statement and its corresponding range. Note that a SNAPX statement always nullifies the effect of a corresponding SNAP statement over the region specified by the arguments of the SNAPX statement.

FORM	RANGE
SNAP	In effect for the entire program.
SNAP(h)	This statement services the region bounded by statement *h* and the end of the program. If *h* is specified as 0, the region serviced is the entire program.
SNAP(h_1,h_2)	In effect for the region bounded by h_1 and h_2. If h_1 is specified as 0, the region extends from the beginning of the program up to and including statement h_2. If h_1 is equal to h_2, the region consists of only one statement.
SNAP(v)	The region serviced by this statement consists of every arithmetic assignment statement in which the variable specified by *v* appears at the left of the equal sign.

An example of each form is given below:

```
SNAP
SNAP(6+2)
SNAP(201., 210.5)
SNAP(VARIB)
```

Note that the second example contains an adjusted statement reference. In this case, the SNAP takes effect with the second statement following statement 6. Only references by statement numbers may be adjusted in this way.

Status Word SNAP: During execution, when a message is printed at the terminal as a result of a SNAP service, the message is always preceded by the status word SNAP.

TRAP Statement

General Forms
TRAP TRAP(h) TRAP(h_1, h_2)

Each form of the TRAP statement essentially performs the same service; i.e., each causes the printing of the origin (line number only) and destination (line and statement number) of every transfer that takes place during program execution. The forms differ only in the range over which they remain in effect. This difference allows the user to determine what regions of his program are to be serviced by the TRAP.

The table below gives each form of the TRAP statement and its corresponding range. Note that a TRAPX statement always nullifies the effect of a corresponding TRAP over the region specified by the arguments of the TRAPX statement.

FORM	RANGE
TRAP	In effect for the entire program.
TRAP(h)	This statement services the region bounded by statement *h* and the end of the program. If *h* is specified as 0, the region serviced is the entire program.
TRAP(h_1,h_2)	This statement services the region bounded by h_1 and h_2. If h_1 is specified as 0, the region extends from the beginning of the program up to and including h_2. If h_1 is equal to h_2, the region consists of only one statement.

An example of each form follows:

```
TRAP
TRAP(26+5)
TRAP(0, 117.)
```

Note that the second example contains an adjusted statement reference. In this case, the TRAP service would take effect with the fifth statement following statement 26. Only references by statement numbers may be adjusted in this manner.

Status Word TRAP: During execution, when a message is printed at the terminal as a result of a TRAP service, the message is always preceded by the status word TRAP.

GUARD Statement

General Forms
GUARD GUARD(h) GUARD(h_1, h_2)

Each form of the GUARD statement essentially performs the same service; i.e., each causes the system to treat certain regions or statements in the program as "guarded." When a "guarded" region or statement is encountered during execution, the system does not execute it. Rather, it prints a message at the terminal indicating that a "guarded" region has been reached and then returns control to the user (indicated by the printing of the status word READY). If the user then types START, the first statement in the "guarded" region is executed, after which control is again returned to the user. Another START would then cause the next "guarded" statement to be executed and so on. When the end of the "guarded" region has been reached, execution continues in the normal fashion.

NOTE: The system always treats a *called* subprogram as "not guarded," regardless of what was specified.

The table below gives each form of the GUARD statement and its corresponding range (i.e., the region it "guards"). Note that a GUARDX statement always nullifies the effect of a corresponding GUARD over the region specified by the arguments of the GUARDX statement.

FORM	RANGE
GUARD	The region "guarded" is the entire program.
GUARD(h)	The region "guarded" is bounded by statement *h* and the end of the program. If *h* is specified as 0, the entire program is "guarded."
GUARD(h_1,h_2)	The region "guarded" is bounded by h_1 and h_2. If h_1 is specified as 0, the region extends from the beginning of the program up to and including h_2. If h_1 equals h_2 the region "guarded" is only one statement.

An example of each form follows:

```
GUARD
GUARD(179.)
GUARD(22+3, 56)
```

Note that the third example contains an adjusted statement reference. In this case the "guarded" region would begin with the third statement after statement 2 and end with statement 56. Only references by statement numbers may be adjusted in this manner.

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

STEP Statement

General Form
STEP

The STEP statement permits the user to interrupt the flow of output caused by a SNAP, TRAP, or a program output statement, whenever he chooses to do so. Use of the STEP statement causes the PROCEED light to be turned on after each line of output has been printed, provided that the KEYBOARD switch is in the SEND position. While this light is on, the user can interrupt the current operation (see "Interrupting Execution" for the exact procedure).

A STEPX statement nullifies the effect of a STEP statement.

TRAIL Statement

General Form
TRAIL

The TRAIL statement causes a message to be printed at the terminal whenever the active program transfers control to a function, a subroutine, or a program in the user's library.

A TRAILX statement nullifies the effect of a TRAIL statement.

Display Statements

The display operating statements 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 constructing a program, the information produced by display statements enables the user to detect errors and omissions. When executing a program, the user can employ display statements to observe the effect of the execution on one or more variables and/or statements.

A display statement can produce a vast 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 their execution (see "Interrupting Execution").

The service provided by each of the display statements can be made a permanent part of the user's program because each service is also available in the form of a system subroutine (see "Program Called Services").

LIST Statement

General Forms
LIST LIST(h) LIST(h_1, h_2)

The LIST statement causes a listing of the statements in the user's program. The number of statements listed is determined by the arguments specified by the user. The listing is printed at the terminal with each statement prefaced by a line number.

The table below gives each form of the LIST statement and its corresponding range (i.e., the region of the program it services).

FORM	RANGE
LIST	Causes a listing of the entire user's program.
LIST(h)	The region listed extends from the statement identified by *h* up to and including the end of the user's program. If *h* is specified as 0, the entire program is listed.
LIST(h_1,h_2)	The region listed is bounded by statements h_1 and h_2. If h_1 is specified as 0, the region extends from the beginning of the program up to and including statement h_2. If h_1 is equal to h_2, only one statement is listed.

An example of each statement follows:

```
LIST
LIST(99+3)
LIST(107., 107.)
```

Note that the second example contains an adjusted statement number. In this case, the listing begins with the third statement following statement 99. Only references by statement numbers may be adjusted in this manner. The third example would cause statement 107. alone to be listed.

COPY Statement

General Forms
COPY COPY(h) COPY(h_1, h_2)

The COPY statement causes a listing of program statements to be printed at the terminal. It produces the same type of listing as does the LIST statement, except that line numbers are not included as part of the listing. The COPY statement is specified in the same manner as the LIST statement (see "LIST Statement").

PDUMP Statement

General Forms
PDUMP PDUMP(h)

The PDUMP statement produces an alphabetical listing of all variables in the specified region of the user's program. Each variable is accompanied by its current value or an indication that it has not yet been "set" (see "Concept of 'Set' and 'Used' Variables"). Array variables are alphabetically listed following the listing of simple variables. However, in the case of an array, only those variables whose values are not zero are printed, with the exception of the first and last variable in the array (these two variables are always printed regardless of their value).

When variables that belong to a *common* storage area (see "COMMON Statement") come under the effect of a PDUMP statement, the following should be remembered:

1. Simple variables belonging to a *common* storage area always behave as "set."
2. Array variables belonging to a *common* storage area behave as follows:
   a. An array variable that has not been "set" by the program in which the PDUMP appears will have only its first and last elements printed.
   b. An array variable that has been "set" by the program in which the PDUMP appears will have all of its elements printed.

The PDUMP statement has two forms:

1. The PDUMP statement with no arguments specifies that all the variables in the entire program are to be listed.

2. The PDUMP(h) statement specifies that all the variables appearing in statement *h* are to be listed. If an array variable is contained in statement *h*, only the last "set" element of that array is listed. Thus, a PDUMP in such a case would be meaningful only when used directly after the execution of statement *h*. If *h* is specified as 0, the statement is equivalent to PDUMP with no arguments.

Some examples of the PDUMP statement are:

```
PDUMP
PDUMP(25)
PDUMP(30+5)
PDUMP(121.)
```

Note that the third example causes the PDUMP to service the fifth statement following the statement numbered 30.

Status Word PDUMP: The first line of output resulting from the execution of a PDUMP statement is prefaced by the status word PDUMP.

QDUMP Statement

General Forms
QDUMP QDUMP(h)

The QDUMP statement produces an alphabetical listing of all variables in the specified region of the user's program that have changed in value since the beginning of the program or since the last execution of a PDUMP or QDUMP statement. Each listed variable is accompanied by its current value. Any array variables are alphabetically listed after all simple variables have been listed. However, in the case of an array only the last "set" element (see "Concept of 'Set' and 'Used' Variables") in the array is listed. If more elements are needed, either a PDUMP or a PRINT 0 should be used.

Note that changes made by a subprogram to array variables belonging to a *common* storage area are not detected by a QDUMP in the calling program.

The QDUMP statement is specified in the same manner as the PDUMP statement (see "PDUMP Statement").

Status Word QDUMP: The first line of output resulting from the execution of a QDUMP statement is prefaced by the status word QDUMP.

INDEX Statement

General Forms
INDEX INDEX(n) INDEX(v)

The INDEX statement produces a listing of statement numbers and variables used in a program. Each statement number and variable is accompanied by the line number(s) of the statement(s) in which each appears.

There are three forms of the INDEX statement:

1. The INDEX statement with no arguments produces a listing with the following information:
   a. All statement numbers in the user's program are listed in numerical sequence. Accompanying each statement number is the line number of the statement which that particular statement number identifies and the line numbers of all other statements that refer to that particular statement number. A minus sign preceding a line number signifies that the statement number is referred to by the statement corresponding to that line number; a plus sign preceding a line number signifies that the statement number identifies the statement corresponding to that line number. If a statement number has not been used as an identifier and has not been referred to elsewhere in the program, it is preceded by an asterisk.
   b. All variables in the user's program are listed alphabetically. Each variable is accompanied by at least one line number indicating the statement(s) in which that particular variable has appeared. A minus sign preceding a line number signifies that the variable has been referred to by the statement corresponding to the line number; a plus sign preceding a line number signifies that the variable has been defined by the statement corresponding to the line number; when no sign precedes a line number, it signifies that the variable has been declared in the statement corresponding to the line number. If a variable has not been defined and has not been referred to, then that variable is preceded by an asterisk. Note that in the case of a statement such as X=X+1, where X is both defined and referred to in the same statement, an INDEX statement would signify X only as being defined.
2. The INDEX(n) statement (where *n* is a statement number) produces a listing like the one described in item 1, part *a* above, but only for statement number *n*. If *n* is specified as 0, this statement is equivalent to an INDEX statement with no arguments.
3. The INDEX(v) statement (where *v* is any variable appearing in the program) produces a listing like the one described in item 1, part *b* above, but only for the particular variable specified (i.e., *v*).

Some examples of the INDEX statement are:

```
INDEX
INDEX(17)
INDEX(VARIB)
```

Status Word INDEX: The first line of output resulting from an INDEX statement is prefaced by the status word INDEX.

CHECK Statement

General Form
CHECK

The CHECK statement produces a listing (in the same format as that produced by an INDEX statement) of all those variables and statement numbers that would have been listed with an accompanying asterisk if an INDEX statement had been used (see "INDEX Statement").

Status Word CHECK: The first line of output resulting from a CHECK statement is prefaced by the status word CHECK.

AUDIT Statement

General Form
AUDIT

The AUDIT statement produces a listing of all the regions in the user's program that have not been exe-

cuted. Each region is indicated by the line numbers of the statements which bound it on either side. Also included is an alphabetical listing of all the variables that have both not been "set" and not been "used" in the program.

Status Word AUDIT: The first line of output resulting from an AUDIT statement is prefaced by the status word AUDIT.

Program Called Services

The services provided by each of the operating statements listed below can be made a permanent part of the user's program. This is made possible by the fact that these services also function as system subroutines. Thus, by using the name of an operating statement as the operand of a CALL statement (see "Rules for Calling Subroutine Subprograms"), a particular service can be retained as part of the active program. Any acceptable form of the following statements can be used:

AUDIT	INDEX	STEP
CHECK	LIST	STEPX
CLEAR	PDUMP	TRAIL
COPY	QDUMP	TRAILX
EDIT	RESET	TRAP
GUARD	SNAP	TRAPX
GUARDX	SNAPX	

The following example illustrates the difference between using a service as an operating statement (PROGRAM A) and using it as a system subroutine (PROGRAM B).

```
   PROGRAM A                      PROGRAM B
   .                              .
   .                              .
   .                              .
10 first executable statement  10 first executable statement
   .                              .
   .                              .
   .                              .
20 IF (X-Y)40, 30, 40          20 IF (X-Y)40, 30, 40
30 CONTINUE                    30 CONTINUE
   SNAP(10, 50)                   CALL SNAP(10, 50)
   .                              .
   .                              .
   .                              .
40 CONTINUE                    40 CONTINUE
   .                              .
   .                              .
   .                              .
50 END                         50 END
   START(0)                       START(0)
```

In PROGRAM A, the "snapping" service is in effect from the moment statement 10 is executed and it remains in effect for the entire program. However, in PROGRAM B, the "snapping" service takes effect only when the associated CALL statement is executed. Since statements 10 through 30 have been executed by the time the service takes effect, they will come under the effect of the "snapping" service only if there is a subsequent transfer of control back to the beginning of the program. Otherwise, only statements 30+2 through 50 are serviced by the SNAP.

Terminal Commands

Introductory Information

The terminal commands are considered to be part of the QUIKTRAN language although their functions are radically different from the other parts of the language (i.e., program and operating statements). Whereas the other parts of the QUIKTRAN language aid the user in constructing a working program, the terminal commands provide the user with an assortment of "housekeeping" services.

Through terminal commands, the user can communicate with the operator of the exchange device, detect certain machine malfunctions, request a new page for printer output, initialize and terminate conversational operations, etc.

The terminal commands in this section are discussed only with regard to their use in conversational operations. Since some of these commands can also be used in batch operations, such use is described in "Basic Information for Batch Processing."

Terminal commands are also unique with respect to their construction and the way they are entered by the user:

1. Each terminal command must be prefaced by a semicolon in order to be recognized by the system. Thus, the semicolon can be considered to be the first character of a terminal command.
2. Terminal commands must be typed by the user beginning in column 1.

The seven terminal commands discussed herein are:

;USER ;CONSOLE ;EXIT ;FINISH	Initialization and Termination
;ECHO ;EJECT ;SEND	Miscellaneous "Housekeeping"

Initialization and Termination Commands

;USER Command

General Form
;USER (identification code)

The ;USER command communicates the user's identification code to the system. *This command must be entered as the very first act in a terminal operation.*

Each user is assigned an identification code by the computing center. Each identification code must consist of six characters. The first two characters are called

the "user group code" and the last four characters, the "user code."

The "user group code" must be two alphabetic characters. The first character may range from A through Z, and the second from A through E. Combinations ZD and ZE are not allowed. Thus, there are 128 valid "user group codes." This code must always be present.

The "user code" must be four characters, each of which may be any alphameric character. Since this code is used along with the "user group code" to control the retrieval of programs in the user's library, it too must always be present.

If the ;USER command is not entered, or if it is entered but not accompanied by an identification code, all enteries by the user are rejected by the system.

An example of the use of this command is:

```
;USER (ZANAME)
```

NOTE: An identification code may be shared by two or more terminals. If two users are simultaneously sharing the same identification code, the possibility exists that:

1. They may concurrently try to execute a call to the same subprogram, or
2. One user may try to execute a call to a subprogram that is currently being executed by the other user.

The type of situation described above is categorized as a "recursive" call to a subprogram. A solution is to wait then try again within a few moments.

;CONSOLE Command

General Form
;CONSOLE

The ;CONSOLE command sets up the terminal for conversational operations. If the terminal has been operating in the conversational manner, the command is rejected (since it is superfluous) and conversational operations continue; otherwise, the system sends back a message indicating that conversational operations can begin.

;EXIT Command

General Form
;EXIT

The ;EXIT command causes the user's identification code to be deactivated. Once the user's identification code has been deactivated, any subsequent CALL, SAVE, LOAD, or PURGE statements are rejected because they no longer have meaning. The user may continue working on his active program but if he desires to use one of the above-mentioned operating statements, he must issue another ;USER command.

The following example shows how a user could transfer a program from his library (identification code AB1234) to another user's library (identification code SE5678).

```
;USER (AB1234)
;CONSOLE
        COMMAND
        LOAD (EXAMP)
;EXIT
;USER (SE5678)
        SAVE
;EXIT
;USER (AB1234)
    .
    .
    .
    END
```

;FINISH Command

General Form
;FINISH

The ;FINISH command indicates to the system that the user wishes to terminate operations for a particular session. Use of this command can be considered as the normal procedure for terminating conversational operations.

Miscellaneous "Housekeeping" Commands

;ECHO Command

General Form
;ECHO (test pattern)

The function of the ;ECHO command can perhaps be best explained by an example. Assume that a user has been trying to enter a program statement and the system continually rejects it although no error is apparent. The question arises that perhaps the system is not reading the statement correctly. To answer this question, the user enters an ;ECHO command with an appropriate "test pattern." (A "test pattern" can consist of anything the user desires to type.) The system reads the command and "echoes" (i.e., prints out exactly that which it has read) the "test pattern" on the very next line in alignment with the original "test pattern." If the original and the "echo" do not match, a machine malfunction is quite probable; if they do match, then either the system is functioning properly and the user is in error, or the original "test pattern" was not very significant and a new "test pattern" should be tried.

To carry the example a step further, assume that the user has been trying to enter the statement, FORMAT (E12.4/F10.4), without success. Suspecting a malfunction, he enters the following command:

;ECHO (FORMAT(E12.4/F10.4) IS IT O. K.)

The system returns the following on the next line:

ECHO (FORMAS(E12.4/F10.4)

Evidently the letter "T" is being read by the system as an "S", thus causing the statement to be rejected.

NOTES:

1. The first right parenthesis terminates a "test pattern" and, consequently, an ;ECHO command.
2. Some nonalphameric characters are not "echoed" exactly as they are read (see Figure 5 and Appendix C).

;EJECT Command

General Form
;EJECT

The ;EJECT command causes the terminal printer to skip to a new page. However, before using this command, the user should make certain that the pages are properly aligned. In other words, when the user begins operating at the terminal, the printer should be adjusted so that typing begins at the top of the page.

;SEND Command

General Form
;SEND (message)

The ;SEND command allows the user to transmit a message to the operator of the exchange device at the computing center. The message may consist of any information that the user wants to transmit. This command may be used at any time after terminal operations have been initiated. If a ;SEND command is rejected by the system, it is suggested that the user make another attempt to transmit his message within the next few minutes. (A rejection indicates that the system cannot accept this message until messages previously sent by other users have been received by the operator.)

Batch processing is a feature of the QUIKTRAN System that allows the user to send entire FORTRAN, COBOL, MAP, etc., programs to the computing center at any time for off-line processing. In addition, batch processing permits the user to request that one or more of the FORTRAN programs he has debugged and saved in the QUIKTRAN disk library be compiled or compiled and executed off line. By allowing debugged programs to be executed off line, batch processing frees the terminal for the construction, testing, debugging, execution, and saving of new FORTRAN programs.

When batch input is received at the computer center, it is temporarily stored until it can be compiled and/or processed. (Each installation will decide for itself as to how often it will process the batch input it has received, e.g., once a day, etc.) Batch output, i.e., processed batch input, is temporarily stored at the computing center until the user requests the output at his terminal.

Batch processing, then, extends the capabilities of the QUIKTRAN System by allowing the user to send jobs to the computing center for processing by the 7040/7044 operating system. The features of batch processing are:

1. The user can send FORTRAN programs for compilation or compilation and execution.
2. If a user has previously saved a FORTRAN program while in the conversational mode, he can request that it be compiled or compiled and executed, via batch processing, without having to retransmit it through his terminal.
3. The user can send non-FORTRAN programs, i.e., COBOL, MAP, etc., for compilation and/or processing.
4. The user can request that his batch output be sent to any user group in the system or that it be printed out at the computing center and sent back to him by mail.

Preparing to Enter the Batch Input Mode

When a user begins operations at his terminal by entering a ;USER command, he receives a message from the system as to what type of QUIKTRAN is available. The QUIKTRAN System has two levels of operation: full QUIKTRAN operation (conversational and batch) and batch only operation. The message "FULL QUIKTRAN SERVICE AVAILABLE" means that he can immediately begin conversational or batch operation. The message "QUIKTRAN SERVICE FOR BATCH ONLY" means that he can start batch operation only, not conversational. If the system status changes while the user is operating his terminal, he will be notified by additional messages.

Before the user begins batch operations, he must select the program that is to be processed off line. He must either (1) have previously saved the FORTRAN program at the computing center while in the conversational mode or (2) be ready to send the program via his terminal. (Note that the maximum length of batch input cannot exceed eighty characters and that binary cards cannot be transmitted from a terminal.)

In addition, the user must decide whether he wants the results of his program typed out at a terminal or printed at the computing center and returned to him by mail. The user can have an output listing of his program printed either at a terminal or at the computing center. If he requests punched output, however, this can only be done at the computing center.

Entering the Batch Input Mode

The batch input mode is initiated when the user enters the ;INPUT command. This command gives an estimate of how many cards are in his program and indicates to which user group the output of this batch is to be sent.

Actual entry of the ;INPUT command depends on how the terminal has been used prior to putting it in the batch mode. For example:

1. If the user has identified himself with a ;USER command but has not yet entered a command such as ;CONSOLE, he can simply enter the ;INPUT command and begin batch operation. For example:

```
;USER (ttuuuu)        user identifying himself
;INPUT (xxx, yy)      directly entering batch mode
```

2. If the terminal has been used for conversational operations since it was signed on, the user must first finish the current operation with a ;FINISH command before entering the ;INPUT command and beginning batch operations. For example:

```
;USER (ttuuuu)        user identifying himself
;CONSOLE               enters conversational mode
(operates terminal in conversational mode)
;FINISH                terminates conversational mode
;INPUT (xxx, yy)       enters batch mode
```

IBSYS Control Cards Used in Batch Input Jobs

The batch input control cards must be included with the user's program. These control cards must be used

whether the user's program is being entered directly from the terminal or is retrieved from the user's library. Even though the system will accept the user's program without these control cards, the program will not be compiled and/or processed without the control cards.

The batch mode control cards should be placed in the card reader along with the cards of the program to be processed. Enter the ;INPUT command either at the keyboard or at the card reader, as shown in the examples below. The following examples show the required control cards and commands for several batch operations.

1. When a FORTRAN program, entered at the terminal is to be compiled but not executed, and the output is assigned to user group AA.

```
1                          16
;INPUT (120, AA)           (estimated card input is 120, output is assigned to user group AA)
$JOB                       (user puts name of program here)
$IBJOB                     NOGO, NODECK
                           (compile program only and do not put compiled program on punched cards because punched card output cannot be received at a terminal)
$IBFTC
(FORTRAN program on punched cards or $RECOM card inserted here)
$IBSYS
;FINISH
```

2. When a FORTRAN program, entered at the terminal, is to be compiled and executed, and the output is to be printed at the computing center and returned to the user by mail.

```
1                          16
;INPUT (120, 00)           (estimated card input is 120, output is printed at the computing center and mailed back to user)
$JOB                       (user puts name of program here)
$IBJOB                     NODECK
                           (do not put compiled program on punched cards)
$IBFTC
(FORTRAN program on punched cards or $RECOM card inserted here)
$ENTRY                     (causes execution of compiled program)
$IBSYS
;FINISH
```

3. When the user wishes to select a program previously saved and debugged in the conversational mode, he inserts a $RECOM card in place of the FORTRAN program on punched cards, using the same control cards as shown in either of the two previous illustrations.

4. When the user wishes to compile and/or process non-FORTRAN programs, entered at his terminal, he should refer to the publication *IBM 7040/7044 Operating System (16/32K): Programmer's Guide*, Form C28-6318, for information regarding control card formats and options. Any such control cards used in batch operations must be preceded by a ;INPUT command and followed by a ;FINISH command.

5. If the user wishes to make use of other control card options when compiling or compiling and executing his FORTRAN programs, he should again refer to the publication *IBM 7040/7044 Operating System (16/32K): Programmer's Guide*, Form C28-6318.

Leaving Batch Input Mode

The batch mode is normally terminated by placing a ;FINISH command card at the end of the punched card deck. Examples of this are shown in the section "Batch Mode Control Cards." If the user wishes to enter the conversational mode after leaving the batch mode, he enters the ;CONSOLE command.

Batch Output

The user has the option of having batch output printed either at his terminal or at the computing center and mailed to him. This selection is made by means of the ;INPUT command, as described in "Entering the Batch Input Mode."

Batch Output Sent to Terminal

Whenever a user identifies himself via the ;USER command, the system will notify him when and if it has batch output for his user group. The user then has the following options:

1. He can request that the batch output of one program be typed out. To do this, he immediately types in the ;OUTPUT command specifying the batch number that the system assigned to the batch input upon entry and user group to which this batch output is assigned. For example, when a user in user group AA has been notified that there is batch output for his user group, he can request that a specific batch output, e.g., AA1234, be printed by typing the command

;OUTPUT (AA1234, AA)

2. He can request that all the batch output assigned to his user group be typed out (multiple batch output). To do this, he immediately types in the ;OUTPUT command replacing the batch number with six zeros and specifying the user group to which this batch output is assigned. For example, when a user in user group AA has been notified that there is batch output for his user group, he can request that all the output be printed by typing the command

;OUTPUT (000000, AA)

3. He can immediately type in a ;INPUT command or a ;CONSOLE command, in order to enter new batch input or put his terminal in the conversational mode. When he finishes either operation with a ;FINISH command, he can immediately begin a new conversational or batch operation. However, if he waits more than three PROCEED lights before entering a new command, he will then start to receive all batch output assigned to his user group (multiple batch output). Note that if the conversational processor is not in operation, the ;CONSOLE command will be rejected.

4. He can automatically receive all the batch output assigned to his user group (multiple batch output) by allowing his terminal to remain idle for three PROCEED lights.

Automatic Batch Output

When a user has been notified by the QUIKTRAN System that batch output is ready for his user group, and if he does not start some operation, e.g., enter a ;CONSOLE command, before the PROCEED light has turned off for the third time, the user will start to receive all the batch output assigned to his user group (multiple batch output). This action will occur at the end of any operation begun after the user has been notified that batch output is waiting.

Batch Output Printed at Computing Center

The user can request that his batch output be printed at the computing center and returned to him by mail. This request is made in the ;INPUT command and is described in "Entering Batch Input Mode." The types of batch output that can be printed at the computing center are described in "Control Cards Used in Batch Input Jobs."

Reassigning Batch Output

The user has the option, after a batch input has been entered (but not processed), of changing the user group to which it was assigned. He also has the option, when batch output is ready (but not yet printed at the terminal), of reassigning the output to another user group.

Reassigning Batch Output Before Processing

The user can reassign batch output after the batch input has been entered but is not yet processed. The reassignment can be from one terminal to another, from a terminal to the computer center, or from the computing center to a terminal. To do this, the user types the ;ROUTE command specifying the batch number that the System assigned when the input was entered and the old and new destination codes.

When the reassigned batch output is ready and one of the users in the newly specified group identifies himself with a ;USER command, he will be notified that his group has batch output.

If the user tries to reassign batch output during the time the input is being processed, he will receive a reject message.

Reassigning Batch Output After Processing

If the user has been notified that batch output assigned to his user group is ready but he wishes to reassign the output to another user group, he must type the ;ROUTE command specifying the batch number assigned when the QUIKTRAN System received the input, and the old and new user group assignments.

When one of the users in the newly specified group identifies himself with a ;USER command, he will be notified that his group has batch output.

Canceling Batch Jobs

The user has the ability, when entering batch input, to cancel the batch input he has just sent. He also has the ability, when receiving batch output, to stop the current batch output operation and to cancel the remainder of the current output that would otherwise be sent to him when the user is receiving multiple batch output, he can cancel the remainder of the current batch output but those batches that follow are not automatically cancelled.

Canceling Batch Input

If the user wishes to discontinue his batch input to cancel the part of this batch input that has just been sent, he should set the RDR1 switch to OFF if the card reader is being used. The KEYBOARD switch must be set to SEND. When the PROCEED light goes on, he should then type the ;CANCEL command. If the system has any batch output for the user's group at this time, it will be automatically sent if the user permits more than three PROCEED lights before starting a new batch or conversational operation.

If the user wishes to cancel a complete batch input that was previously sent (but one that has not yet been processed), he must first finish his present operation with a ;FINISH command and then type the ;PURGE command, specifying the batch number assigned when the system received the input and the destination code. If the System cannot accept the ;PURGE command because the specified batch input has been sent to the 7040/7044 for processing, it will send back a reject message to the user.

Canceling Batch Output

If the user is receiving batch output and does not want to receive the rest of this particular output, he should set the KEYBOARD switch to SEND. The PROCEED light will go on when the printer reaches the top of the next page. An EOT is transmitted by the user and, when the system responds with a READY message and another PROCEED light, he enters the ;CANCEL command. The ;CANCEL command cancels only the remainder of the current batch output. It does not cancel succeeding batch output if the user is receiving multiple batch output.

If there is no further batch output, the user can select any conversational or batch command for the next operation. However, if there is more batch output and this is a multiple batch output operation, the next batch output will begin immediately. After one page has been printed, i.e., when the PROCEED light goes on, the user can cancel it. The multiple batch output will continue until all the batch output assigned to this user group has been printed or canceled.

The user also has the option of canceling a specific batch output before it is printed (after it has been processed and is ready to be transmitted to the user). The ;PURGE command is used, specifying the batch number assigned when the system received the input and the user's group code. If the user has just received notification of batch output, he can enter the ;PURGE command immediately to cancel a specific batch. If he is in the middle of performing some operation, however, when he decides to use the ;PURGE command, he must first finish his present operation with a ;FINISH command before entering the ;PURGE command. If, after being notified of batch output, the user waits more than three PROCEED lights before starting a new batch or conversational operation, the system will automatically start to send him all the batch output assigned to his user group.

Note that the user can interrupt a multiple batch output to start a conversational operation but not a new batch operation. For example, a ;PURGE command cannot be used *during* a multiple batch output but it can be used before a multiple batch output to purge a specific batch. During a multiple batch output, then, the user can interrupt it to cancel the current batch output (but not the remainder of the batches in the output) or he can interrupt it to enter conversational operation. He cannot interrupt a multiple batch output to begin such operations as ;PURGE or ;INPUT.

Interrupting and Resuming Batch Input and Output

The user has the option of interrupting batch input and output operations so that he can enter the conversational mode (;CONSOLE command). The system will automatically interrupt the user's operation if the user accidentally disconnects his terminal from the computing center while he is sending batch input or receiving batch output. The batch operation is suspended until the terminal is again connected and a ;RESUME command is given.

Interrupting Batch Input

If the user wishes to interrupt batch input, he should set the RDR1 switch to OFF if the card reader is being used. The KEYBOARD switch must be set to SEND. He can now type a ;CONSOLE command in order to start conversational operation. (Note that the ;CONSOLE command can be used only when the full QUIKTRAN System is in operation. Also note that the user cannot begin a second batch operation after interrupting a previous batch operation.) At the end of the conversational operation, the user enters a ;FINISH command and then types the ;RESUME command specifying the batch number assigned when the system received the input. The user then restarts the card reader operation to continue his batch input.

Note that while a ;CANCEL command can be entered after interrupting batch input to delete the interrupted job, a ;PURGE command cannot. A ;PURGE command can only be used after the current batch operation has been finished by a ;FINISH command.

Interrupting Batch Output

If the user wishes to interrupt batch output during the time it is being received at his terminal, he sets the KEYBOARD switch to SEND. When the printer reaches the beginning of the next page, the PROCEED light will go on. The user should then transmit an EOT. After the system prints a READY message, the user can then type a command in order to start another operation, e.g., ;CONSOLE. (Note that the ;CONSOLE command can be used only when the full QUIKTRAN System is operating. Also note that the user cannot begin a second batch operation after interrupting a previous batch operation.) At the end of the conversational operation, the user enters a ;FINISH command and then types the ;RESUME command specifying the batch number of the interrupted output. The system then resumes typing batch output where it left off when interrupted.

If the batch number in the ;RESUME command is incorrect, the entire batch output will be returned to the stack of available output and must be requested again with a ;OUTPUT command, using the correct batch number. The batch output received by this new ;OUTPUT command will start from the beginning of the batch.

Therefore, if the user has interrupted batch output and now wants to receive all of this particular batch output from the beginning, he should enter the command ;RESUME (xxxxxx), followed by the command ;OUTPUT (ttnnnn, yy) where ttnnnn is the correct batch number.

Note that while a ;CANCEL command can be used after interrupting batch output, a ;PURGE command cannot. A ;PURGE command can be used only before the current batch output operation has been started.

Messages Sent from the Computing Center

Under certain conditions, the computing center will send a message to the terminal instead of the expected batch output. These conditions are:

1. When the number of lines of output exceeds the maximum number of lines that can be sent back to the remotely located terminal. The batch output can be printed only at the computing center.
2. When the number of lines of output exceeds the maximum number of lines that have been allocated for batch output storage at the computing center. The first part of the output will be replaced by the excess lines and is thereby destroyed. The latter part of the batch output can be printed at the computing center.
3. When the batch output is printed at the computing center because it could not be made available to the terminal at the time it was processed.
4. When a saved program, selected by a $RECOM card for batch processing, cannot be recomposed and processed.
5. When a saved program, selected by a $RECOM card for batch processing, is not found in the user's library.
6. When the program name was missing on the $RECOM card.

Batch Mode Commands

The following commands are used before, during, and after batch operations.

Commands for Entering and Leaving the Batch Mode

;INPUT Command

General Form
;INPUT (xxx, yy) where: xxx is the three digit maximum estimate by the user of the number of cards in this batch input (001-999) yy is the user group to which the output of this batch is assigned a user group code of 00 indicates that the output of this batch is to be printed at the central computer.

The ;INPUT command is used when the user wishes to send batch input to the central computer for compiling or compiling and/or processing. If the user has been operating in the ;CONSOLE mode, he must end this operating with a ;FINISH command before he can enter the ;INPUT command. When the ;INPUT command is entered (via keyboard or punched card), the system assigns a batch number to this particular batch input and prints it at the terminal. The batch number consists of six characters ttnnnn, tt being the user's group code and nnnn being four digits. If the system cannot accept batch input at this time, it sends a reject message to the user.

If the user wishes to have the output of his program printed at the computing center and mailed back to him, yy should be 00. If the user's estimated card count is too large for the system to accept at this time, it will send back a reject message. If the actual card input exceeds the estimated card count, the system will continue to accept input for as long as it has space. If it runs out of space, the entire batch is rejected and discarded.

;OUTPUT Command

General Form
;OUTPUT (ttnnnn, yy) where: ttnnnn is the batch number which the system assigned when the user entered the batch input. A batch number equal to zeros indicates that all batch output for this user group is to be printed at the terminal. yy is the user group code to which the batch output has been assigned.

The ;OUTPUT command is used when the user, after being notified that batch output is waiting, requests that some or all of it be printed at his terminal. If the user ignores the batch output notification or only requests a specific batch output and, after receiving it, begins another operation (e.g., ;CONSOLE) as soon as he ends the new operation with a ;FINISH command, he can immediately begin a new conversational or batch operation. However, if he waits more than three PROCEED lights before entering a new command, the system will automatically begin to print the remainder of the batch output assigned to his user group whether he wants it or not. The canceling procedure may be followed at this time.

;RESUME Command

General Form
;RESUME (ttnnnn) where: ttnnnn is the batch number that the system assigned when the user entered the batch input.

The ;RESUME command is used to resume batch input or output operations after the user previously inter-

rupted the batch operations to enter the conversational mode. A ;FINISH command must be used at the end of the conversational operation before the ;RESUME command can be entered. Details on how to interrupt batch operations are given in "Interrupting and Resuming Batch Input and Output."

If the user has interrupted batch output and if he wishes to receive all of this particular batch output from the beginning, he should, at the proper time, enter the command ;RESUME (xxxxxx), followed by the command ;OUTPUT (ttnnnn,yy), where ttnnnn is the correct batch number.

;FINISH Command

General Form
;FINISH

The ;FINISH command is used to end an operation begun by a ;INPUT or ;CONSOLE command.

Commands for Changing Batch Operations

;CANCEL Command

General Form
;CANCEL

The ;CANCEL command is used to cancel the current batch input or output operation being performed. All of the current batch input that has just been entered by the user will be canceled by the ;CANCEL command. Note that the ;CANCEL command can cancel only those batch inputs that are currently in the process of being entered, not those that have been fully entered and followed by a ;FINISH command (see ";PURGE Command"). If the user is receiving batch output and does not wish to receive more of the current batch, he sets the KEYBOARD switch to SEND, and waits for the PROCEED light to go on (when the printer reaches the beginning of the next page), transmits an EOT, and then types a ;CANCEL command. The remainder of the current batch output will be canceled.

;PURGE Command

General Form
;PURGE (ttnnnn, yy) where: ttnnnn is the batch number that the system assigned when the user entered the batch input (tt being the user's group code and nnnn being four digits) yy is the user group code to which the batch output has been assigned

The PURGE command is used to cancel a previously submitted (but unprocessed) batch input or a waiting batch output.

;ROUTE Command

General Form
;ROUTE (ttnnnn, yy, rr) where: ttnnnn is the batch number which the system assigned when the user entered the batch input (tt being the user's group code and nnnn being four digits) yy is the user group to which the output was previously assigned rr is the user group to which the output is now assigned.

The ;ROUTE command is used to specify a new user group destination in place of the one previously assigned. This command can be used after batch input has been entered (but not yet processed). It can also be used when batch output is ready but has not yet been printed at the terminal.

Before processing, batch output can be reassigned from one remote terminal to another, from a remote terminal to the computing center, or from the computing center to a remote terminal.

After processing, batch output can only be reassigned from one remote terminal to another, provided that the output has not yet been printed at a remote terminal. Batch output, at this time, cannot be reassigned from the computing center to a remote terminal or from a remote terminal to the computing center.

Batch Mode Control Card

$RECOM Control Card

The format of this card is:

1	16	22	29
$RECOM	prog name	,user ident	

prog name must be the same as the one under which the program was saved

user ident is the user's six character identification

The $RECOM control card is used when the user wishes to compile or compile and execute a program via batch processing that he has previously saved during conversational operation (see "SAVE Statement"). The $RECOM control card is used during batch ;INPUT operations (see "Control Cards Used in Batch Input Jobs"). It is not necessary to use the user identification code assigned to this program unless it is different from the one used in the current ;USER command. Note that if the user identification is used, a comma in column 22 must precede it.

All FORTRAN input/output statements in the user's program should have valid FORTRAN input and output unit numbers (see the publication *IBM 7040/7044 Operating System (16/32K): FORTRAN IV Language*, Form C28-6329). All input must be supplied to the computing center before execution as part of the batch input job.

This section consists principally of a listing produced at a terminal during a QUIKTRAN session. Its purpose is to demonstrate the response of the system to all the statements and commands available to a user, in a setting as near as possible to a natural one for QUIKTRAN. Owing to the listing's didactic purpose, the ration of messages and comments to QUIKTRAN statements is relatively high, as is the ratio of operating to program statements.

How to Read the Listings

The output listing from a QUIKTRAN terminal is a record of all communications between the user and the system. The user transmits his input to the system from the keyboard (or card reader), and at the same time the corresponding characters are typed by the printer. The system, in turn, transmits all its output to the printer. Consequently, the origin of a particular segment of the listing is not always obvious. Distinguishing between the two sources presents no difficulty, however, to anyone who can follow the process by which user and system together produced the output shown.

The beginning of the listing is shown in Figure 5. The first thing a user must do is set up the terminal in accordance with instructions given in "Equipment." Then, as soon as the user has established a connection between the computing center and his terminal, the system responds with:

```
000     READY
```

```
000   READY ;USER[MEJDOE]

          FULL QUIKTRAN SERVICE AVAILABLE
000   READY ;CONSOLE AND THIS LINE WILL NOT HAVE AN EOB...
000   RJECT KB TIMEOUT
000   READY ;C ONSOLE
000   RJECT REQUEST
000   READY ;CONSOLE
101. -READY CV
101. -READY CV*****************************************************************  1  **
101. -READY CV       STATUS WORD "RJECT" - MEANS USER'S ENTRY ON PRECEDING LINE
101. -READY CV         WAS NOT ACCEPTED BY THE SYSTEM
101. -READY CV       STATUS WORD "READY" - MEANS SYSTEM IS INVITING INPUT FROM
101. -READY CV         THE TERMINAL.
101. -READY CV
101. -READY CV*****************************************************************  2  **
101. -READY CV       "CV" IN COLS. 1 AND 2, PRINTS THIS LINE, IN THIS LISTING, AT
101. -READY CV       THIS TIME. (THE SYSTEM NEVER RECEIVES IT AS INPUT)
101. -READY CV       THIS DECK USES "CV" TO POINT OUT QUIKTRAN STATEMENTS AND
101. -READY CV         RESPONSES.
101. -READY CV
101. -READY CV       ";USER(      )" IDENTIFIES ALL INPUT FROM THIS TERMINAL AS
101. -READY CV       YOURS; IDENTIFICATION EFFECTIVE UNTIL ";EXIT".
101. -READY CV
101. -READY CV       ";CONSOLE" STARTS  CONVERSATIONAL  OPERATIONS BETWEEN USER
101. -READY CV       AND THE SYSTEM.
101. -READY CV
101. -READY CV*****************************************************************  3  **
101. -READY CV       NOW, THE SYSTEM IMMEDIATELY PRINTS BACK THE PARENTHESIZED
101. -READY CV         CHARACTERS, EXACTLY AS IT READ THEM.
101. -READY CV
101. -READY ;ECHO(THEQUICKBROWNFOXJUMPSOVERLAZYDOG0123456789,='/-$+..>;$(:"*[?!<)
           ECHO(THEQUICKBROWNFOXJUMPSOVERLAZYDOG0123456789,='/-$+..=;:(:"*(?!+)
101. -READY ;ECHO(:>[<)
           ECHO(:=(+)
101. -READY ;ECHO(Δ)
101. -RJECT ILLEG CHAR
101. -READY ;ECHO(*)
101. -RJECT ILLEG CHAR
101. -READY ;ECHO(\)
101. -RJECT ILLEG CHAR
101. -READY CV
101. -READY CV*****************************************************************  4  **
101. -READY CV       TRANSMIT THE ENCLOSED MESSAGE TO THE OPERATOR AT THE CENTER
101. -READY CV
101. -READY ;SEND (WE ARE GIVING A DEMONSTRATION HERE)
101. -READY CV
101. -READY CV*****************************************************************  5  **
101. -READY CV       SKIP THE PRINTER CARRIAGE TO THE TOP OF THE NEXT PAGE.
101. -READY CV
101. -READY ;EJECT
```

Figure 5.

The print element is now positioned to the right of READY. When the PROCEED light comes on, the keyboard unlocks. It is now the user's turn. (*Whenever* it is the user's turn to type, the print element is at the point where it is now.)

The PROCEED light stays on all during the time that the user can communicate with the system. The light will go off when (1) the user transmits an "end-of-line" (EOB) signal (not shown in the listing because it is not a character that prints), or (2) approximately 15 seconds have elapsed without any activity at the keyboard. Anything the user has entered meanwhile will have been typed on the rest of the line, beginning where the system left off. When the PROCEED light goes off, the keyboard locks, and the print element moves to the beginning of a new line. The system now examines what the user transmitted. When the system has concluded its examination, it again starts the printer.

(On the second line of Figure 5, the computing center has interspersed a message between the user's EOB and the system's response. It is always possible for an operator to send such a message; at sign-on time it is customary.)

The message 000 READY is an example of the way the system always begins its turn, i.e., with a line number and a status message. The status message is not always the word READY. Figure 5 shows some messages beginning with another status word, RJECT. When the status word is READY, it is the entire message. When the status word is RJECT, the balance of the message tells why the user's statement was rejected. The system uses as much of the line beyond the status word as it may need. It may take up more than one line. But no matter how many lines may be used, for no matter how many messages, the system will always end up returning to the beginning of a new line, and then typing a number followed by READY. The PROCEED light comes on again; it is again the user's turn.

Note that if the user makes his entries by way of the card reader, the sequence described remains the same with one exception. If the KEYBOARD switch is OFF, the keyboard will remain locked and the PROCEED light will stay off even though the system is reading a card. There is, of course, no possibility of exceeding the 15 seconds' time out; with the reader, it is always EOB which ends the user's turn.

What has been described is the essence of the conversational process. It partitions every listing into two fields: system and user. Always, the first part of the line is reserved to the system, which prints line numbers and status information. This is the "control field" and takes up the first twelve print positions. What follows the control field on a given line will have originated with either the system or the user, depending strictly on what is printed in that field. If READY, ALTER, or I (see below) is there, all that follows was entered by the user. In all other cases, it was printed by the system.

Housekeeping

In any programming environment, certain steps are necessary that are ancillary to the main business of computing. The terminal commands perform such duties in QUIKTRAN. In Figure 5, the user's first entry establishes the identification MEJDOE as pertaining to all information coming from his terminal. (This identification remains in force until ;EXIT is given. See Figure 11.) The statement ;CONSOLE requests conversational interaction between the user and QUIKTRAN. (The statement ;FINISH ends the conversational operations.)

It is necessary that the semicolon symbol (;) of these statements occupy the very first position made available for the user. This position is "column 1," matching the FORTRAN statement coding form and the IBM punched card. (The characters are printed from a read-in card into the corresponding "columns" of the listing.)

For the terminal commands to work, the characters must be entered closed up, one in each column, without the "embedded blanks" that are tolerated in all other QUIKTRAN statements.

The rest of the commands are also shown in Figure 5. Observe that the user can employ ;ECHO ( ) to verify the accuracy of transmission from the terminal, by character and column. The statement ;EJECT will move the paper a length of one page. Note that to the system, the top of the first page is wherever the user positions the paper for printing, before he begins. At any time, the user can turn the roller by hand, changing the relation of the actual top to the "top" the system is counting from. (The user cannot control spacing from the keyboard or reader, because RETURN and LINE FEED are illegal characters in QUIKTRAN.)

Like the terminal commands, the comment code CV also must start in column 1. When the user puts an EOB at the end of a CV line, the system does not examine the line in the way it would others. Indeed, it behaves as though the line had never existed (except for keeping track of the number of lines on a page). In other words, after a CV line, the system invariably responds with the same status word that invited the preceding entry (i.e., the CV). The CV code has been used unsparingly in the illustrations, in order to flag statements for the attention of the student. (It may, in fact, help to imagine the listing's appearance with none of the CV's. The listing without these would be somewhat more typical.)

Immediate Execution

QUIKTRAN is responsive on a statement-by-statement basis. This means that the system can execute any valid arithmetic statement, if the user so desires. Unless the user is building a program (as in Figures 7 through 11), the system during conversational operations does just that, automatically.

After the user has entered ;CONSOLE (see Figure 5), the system changes the line number from 000 to 101, and puts a minus sign (−) before the status word. This tells the user that he has established a conversational relationship between his terminal and computing center. (The conditions under which the system will increment the line numbers are illustrated in Figure 7. In Figures 5 and 6, the line number does not change.)

The user now has two capabilities:

1. He can execute arithmetic assignment statements consisting of known constants and reserved functions; or
2. He can activate a program.

Figure 6 shows the user exercising the first of these capabilities.

In Figure 6, the user is not building a program. Therefore, every time the system receives a valid arithmetic statement, it: (1) evaluates the expression; (2) goes to the line below the user's entry and types an equals sign (=) after the line number (instead of a minus), signifying *output* as the result of execution by the system; (3) omits the status word for that line; (4) prints out the value of the expression; and (5) discards the statement and related variables. The value is printed out in decimal notation, the exact form of which can be changed at the option of the user (see below). The system then returns to the READY condition, inviting more input. The FORTRAN arithmetic statement, employed by itself in this fashion, can thus make of the user's terminal a powerful symbolic calculator.

Several features assist the user in these pursuits. Note first that the statements entered for execution follow the placement rules for FORTRAN coding: column 6 is blank and the statement may begin in any column thereafter (if it ends before the end of the line). Most users set a tab stop for column 7. (If the user tabs to the beginning of his entry, *wherever* he

```
101. -READY        X = 2. + 2.
101. =            X= 0.40000000E 01
101. -READY CV
101. -READY        ANSW = SQRT( SQRT( 625.0))
101. =            ANSW= 0.50000000E 01
101. -READY CV
101. -READY                    ROOT = SQRT(4720.)
101. =            ROOT= 0.68702256E 02
101. -READY CV
101. -READY            ROOT2 = 4720.**0.5
101. =            ROOT2= 0.68702256E 02
101. -READY CV
101. -READY        ROOT3 ENDS BY HITTING CANCEL KEY-
101. -CANCL PREV LINE
101. -READY CV
101. -READY CV*****************************************************************  6  **
101. -READY CV        STATUS WORD "CANCL" - PRECEDING LINE TREATED LIKE A
101. -READY CV          "CV" LINE PER REQUEST FROM KEYBOARD.
101. -READY CV
101. -READY        X = SQRT[100.000]
101. =            X= 0.10000000E 02
101. -READY CV
101. -READY ;ECHO[AND THEN I BACKSPACE OVER THE WORDS I JUST TYPED, AND STRIKE OVER WITH NEW WORDS]
            ECHO(AND THEN I BACKSPACE OVER THE WORDS I JUST TYPED, AND STRIKE OVER WITH NEW WORDS)
101. -READY CV
101. -READY ;ECHO[ONE CAN STRIKE OVER WITH BLANKS   ]
            ECHO(ONE CAN STRI                       )
101. -READY ;CV
101. -RJECT REQUEST
101. -READY CV
101. -READY        Z = 57289. - 29,756.00
101. -RJECT  STATEMENT NOT IN LANGUAGE
101. -READY CV
101. -READY        X = 3 + 9.
101. -RJECT  MIXED MODE
101. -READY CV
101. -READY        ROOT = SQRT(SQRT(625.)
101. -RJECT  PARENTHESES NOT IN BALANCE
101. -READY CV
101. -READY        Z = 2.**3
101. =            Z= 0.80000000E 01
101. -READY CV
101. -READY        E =  20. * ATAN ( 20 ./4)) - 4./2. * ALOG(4.**2+20.**2)
101. -RJECT  PARENTHESES NOT IN BALANCE
101. -READY         E = 20. * ATAN ( 20./4.)  - 4./2. * ALOG(4.**2+20.**2)
101. =            E= 0.15406645E 02
101. -READY CV
101. -READY        VAL = 1./COS(50.)+ALOG(ABS(SIN(50./2.)/COS(50./2.)))
101. =            VAL=-0.97715002E 00
101. -READY ;EJECT
```

Figure 6.

tabs to is regarded as column 7 by the system.) Although the terminal prints each character as it is entered by the user, the user is free to change his mind until with EOB he signals that his message is complete. If the user omits EOB, control will pass after about 15 empty seconds back to the system, which will soon present him with a fresh line. He can defer that event by working the SHIFT key, or hasten it by pressing the CANCEL key, then pressing EOB. In either case, nothing is left of the line, except the impression made on the paper at the time it was being entered.

Another way for the user to avoid entering a defective statement is to backspace and correct characters (before EOB). He must remember that backspacing over characters deletes them and that spacing over them again, accordingly, leaves blanks. In any event, the user need never enter a line he knows to be in error.

As for lines that are entered by the user in spite of errors they may contain, the examination performed by the system rejects any defectively composed statement. Figure 6 shows several incorrect statements and the system's diagnostic messages rejecting them. In practice, this is nearly equivalent to debugging the statement. Note that the READY condition *permits* but does not *oblige* the user to do something about his defective statement. As far as the system is concerned, the statement it just rejected has never existed (like the CV line).

Command Mode, Program Mode, and Automatic Mode

QUIKTRAN, of course, does not confine execution to the single statement. The user can enter any number of FORTRAN statements one-by-one, for serial execution later. The system, in that case, keeps track of the statements. Any such sequence of statements is, in QUIKTRAN, a program.

Figure 7A shows a user entering a short program (the program itself is of no consequence). The user begins by entering a statement that contains the word PROGRAM and the name XX1. The system, in its turn, (1) adds one

```
101. -READY CV*****************************************************************  7  **
101. -READY CV       TO GET INTO PROGRAM MODE, EITHER (1) LOAD A ROUTINE FROM
101. -READY CV       YOUR LIBRARY OR (2) BEGIN A NEW PROGRAM; FOR EXAMPLE, .....
101. -READY CV
101. -READY       PROGRAM XX1
102. +READY CV
102. +READY CV*****************************************************************  8  **
102. +READY CV       PLACE AN IMAGE OF THE ACTIVE PROGRAM IN THIS USER'S LIBRARY.
102. +READY CV
102. +READY       SAVE
102. +READY CV
102. +READY CV*****************************************************************  9  **
102. +READY CV       INITIATE EXECUTION FROM ( ). (A "0" SPECIFIES THE BEGINNING
102. +READY CV       OF THE PROGRAM.)
102. +READY CV
102. +READY       START (0)
101. =HALT END OF PROGRAM ENCOUNTERED DURING EXECUTION
102. +READY CV
102. +READY CV**************************************************************** 10  **
102. +READY CV       STATUS WORD "HALT" -  - LACK OF ANY FURTHER STATEMENTS IN
102. +READY CV         YOUR PROGRAM STOPPED EXECUTION OF IT.
102. +READY CV
102. +READY CV**************************************************************** 11  **
102. +READY CV       NOW DESTROY THE ACTIVE IMAGE OF THE PROGRAM, IN ORDER TO
102. +READY CV         EITHER (1) ENTER STATEMENTS FOR IMMEDIATE EXECUTION, OR
102. +READY CV       (2) ACTIVATE ANOTHER PROGRAM.
102. +READY CV
102. +READY       COMMAND
101. -READY CV
101. -READY CV**************************************************************** 12  **
101. -READY CV       USER AGAIN CHANGES FROM COMMAND TO PROGRAM MODE, THIS TIME BY
101. -READY CV       ACTIVATING A PROGRAM ALREADY IN HIS LIBRARY.
101. -READY CV
101. -READY       LOAD (XX1)
102. +READY CV
102. +READY CV**************************************************************** 13  **
102. +READY CV       PRODUCE A LISTING OF THE ACTIVE PROGRAM
102. +READY CV
102. +READY       LIST
101. =      CF    PROGRAM XX1
102. +READY CV
102. +READY CV**************************************************************** 14  **
102. +READY CV       A "C" IN COL. 1 AND A BLANK IN COL. 2 INCLUDE THE LINE IN
102. +READY CV       EVERY LISTING OF THE PROGRAM WITHOUT AFFECTING EXECUTION.
102. +READY CV
102. +READY C  ................GENERATOR NO. 1
103. +READY       X=0.
104. +READY     2   X = X + 1
104. +RJECT  MIXED MODE
104. +READY     2 X=X+1.
105. +READY       A = X
106. +READY       Y=X
107. +READY     2 A = A - 1.
107. +RJECT  STATEMENT 2 HAS BEEN PREVIOUSLY DEFINED
107. +READY     3 A = A-1.
108. +READY       IF (A-0.) 4,4,5
109. +READY     5 Y=Y+A
110. +READY       GO TO 4
111. +READY     4 PRINT 101,X,Y
```

Figure 7A.

```
112. +READY   101 FORMAT (2E15.2)
113. +READY       IF (5. - X) 6,6,2
114. +READY     6 STOP 333
115. +READY       END
116. +READY       START (0)
111. =0101         0.10E 01       0.10E 01
111. =0101         0.20E 01       0.30E 01
111. =0101         0.30E 01       0.50E 01
111. =0101         0.40E 01       0.70E 01
111. =0101         0.50E 01       0.90E 01
114. =STOP  333
116. +READY CV
116. +READY CV*************************************************************** 15  **
116. +READY CV       "0101" IN STATUS-WORD POSITION.  AN OUTPUT LINE IS ABOUT TO BE
116. +READY CV       PRINTED UNDER CONTROL OF FORMAT STATEMENT NO. 101.
116. +READY CV
116. +READY CV       STATUS WORD "STOP" - THAT STATEMENT, ARRIVED AT IN THE COURSE
116. +READY CV       OF EXECUTING USER'S PROGRAM, STOPPED EXECUTION. LINE NUMBER TO
116. +READY CV       LEFT OF "STOP" IS LINE NUMBER OF STOP STATEMENT IN PROGRAM.
116. +READY CV       NUMBER FOLLOWING, IF ANY, IS USER-SUPPLIED STOP CODE.
116. +READY CV
116. +READY CV*************************************************************** 16  **
116. +READY CV       PRINT OUT DESTINATION WHENEVER BRANCH IS TAKEN.
116. +READY CV
116. +READY       TRAP
116. +READY       START (0)
108. =TRAP  TRANSFER TO 4 (111.)
111. =0101         0.10E 01       0.10E 01
113. =TRAP  TRANSFER TO 2 (104.)
108. =TRAP  TRANSFER TO 5 (109.)
110. =TRAP  TRANSFER TO 4 (111.)
111. =0101         0.20E 01       0.30E 01
113. =TRAP  TRANSFER TO 2 (104.)
108. =TRAP  TRANSFER TO 5 (109.)
109. =BREAK EXECUTION INTERRUPTED BY TERMINAL USER-LAST EXECUTED  108.
116. +READY CV
116. +READY CV*************************************************************** 17  **
116. +READY CV       STATUS WORDS-  "TRAP" IDENTIFIES USER-ORDERED BRANCH TRACE.
116. +READY CV                      "BREAK" ALWAYS MEANS SIGNAL FROM USER STOPPED
116. +READY CV                        EXECUTION.
116. +READY CV
116. +READY CV       (USER DISCOVERS THAT THERE IS NO LOOP BACK TO 3)
116. +READY CV
116. +READY CV*************************************************************** 18  **
116. +READY CV       DELETE THE SPECIFIED REGION (IN THIS CASE, A SINGLE LINE).
116. +READY CV
116. +READY       ALTER(110.,110.)
110. +ALTER CV
110. +ALTER CV*************************************************************** 19  **
110. +ALTER CV       STATUS WORD "ALTER" MEANS THAT SYSTEM IS OFFERING A LINE,
110. +ALTER CV       ON WHICH USER MAY ENTER STATEMENT FOR INCORPORATION INTO
110. +ALTER CV         ACTIVE PROGRAM.  LINE NUMBER INDICATES PLACE IN PROGRAM.
110. +ALTER CV
110. +ALTER       GO TO 3
110.1+ALTER CV
110.1+ALTER CV*************************************************************** 20  **
110.1+ALTER CV       END ALTER SEQUENCE.
110.1+ALTER CV
110.1+ALTER       ALTERX
```

Figure 7B.

to the line number (which becomes 102.), and (2) prints a plus sign (+) after the line number instead of a minus. As the + signifies, a program is now active at the terminal.

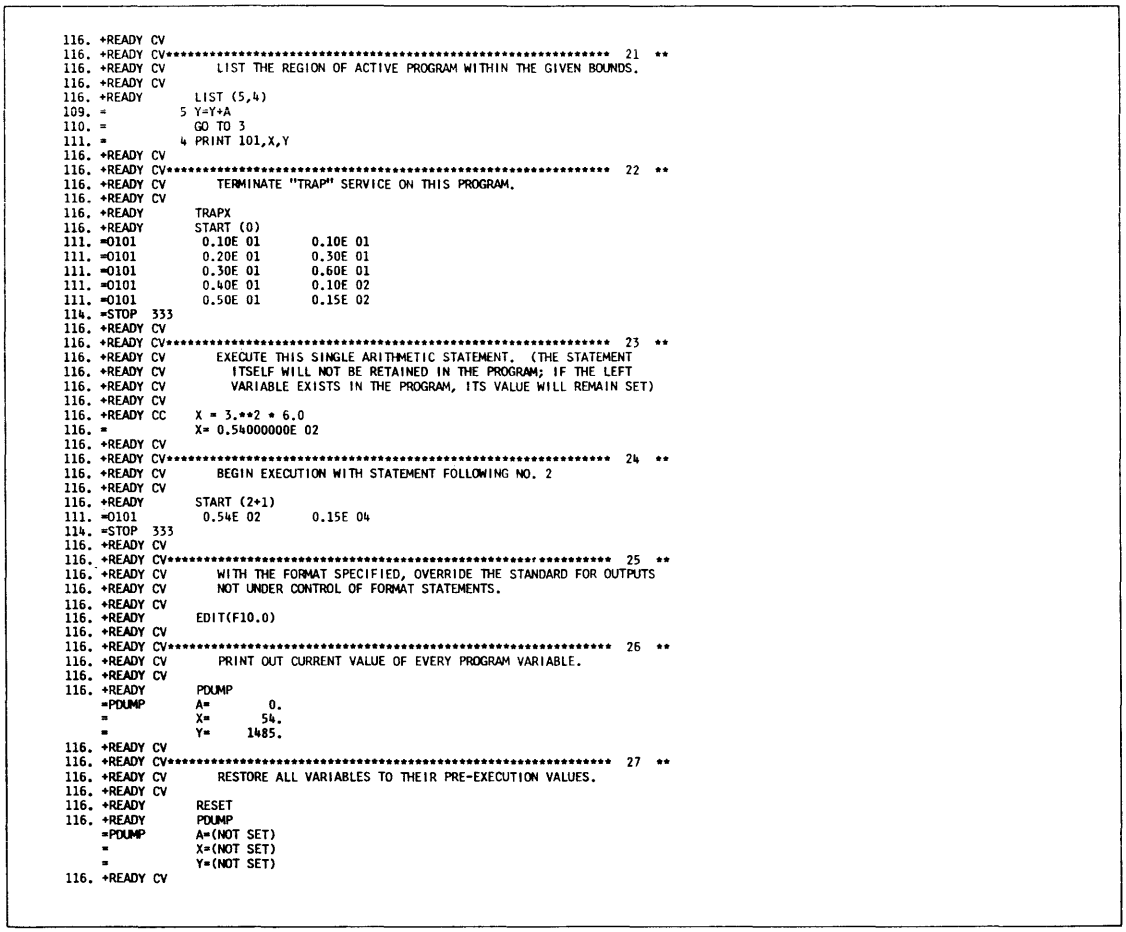
The system will, from now on, keep track of the statement on line 101. and any subsequent program statements entered by the user, regarding them as a single program. Assigning 101. to the first statement, the system supplies a line number to every statement in the user's program. Note that (1) the system always supplies this number automatically in the control field; (2) the user does have discretionary powers over the interval of increment and can renumber if he wants (see below); and (3) the line number is *not* a label by which his program can refer to the statement: for that the user puts his own statement number in columns 1 through 5.

Note in the example, that the statement which made the program active becomes the program's first statement. Note also that one statement is all it takes to make up a program. In Figure 7, when the system is offering line 102. to the user, the active program consists solely of the statement PROGRAM XX1.

For the period that a program is "active" the system considers that all entries (except housekeeping entries) refer to that program. For example, if the user enters SAVE (see Figure 7), the system copies an image of the active program into the user's library, where it stays under the name XX1. The system then repeats 102. in the control field. Program XX1 is *still* active.

Or if the user enters START(0), the system will attempt to execute the program. With the only statement in the program not an executable one, the system will not get very far this time. It is important, however, that the reader realize that the attempt will be made. The system will then repeat 102. +READY in the control field. Program XX1 is still active.

Or if the user enters another program statement, the system (after inspecting for composition) examines the statement for consistency with the rest of the program. If the system accepts the statement, the system will consider it to be the next statement in order in the pro-



```
116. +READY CV
116. +READY CV*************************************************************** 21  **
116. +READY CV       LIST THE REGION OF ACTIVE PROGRAM WITHIN THE GIVEN BOUNDS.
116. +READY CV
116. +READY       LIST (5,4)
109. =          5 Y=Y+A
110. =            GO TO 3
111. =          4 PRINT 101,X,Y
116. +READY CV
116. +READY CV*************************************************************** 22  **
116. +READY CV       TERMINATE "TRAP" SERVICE ON THIS PROGRAM.
116. +READY CV
116. +READY       TRAPX
116. +READY       START (0)
111. =0101         0.10E 01       0.10E 01
111. =0101         0.20E 01       0.30E 01
111. =0101         0.30E 01       0.60E 01
111. =0101         0.40E 01       0.10E 02
111. =0101         0.50E 01       0.15E 02
114. =STOP  333
116. +READY CV
116. +READY CV*************************************************************** 23  **
116. +READY CV       EXECUTE THIS SINGLE ARITHMETIC STATEMENT.  (THE STATEMENT
116. +READY CV         ITSELF WILL NOT BE RETAINED IN THE PROGRAM; IF THE LEFT
116. +READY CV         VARIABLE EXISTS IN THE PROGRAM, ITS VALUE WILL REMAIN SET)
116. +READY CV
116. +READY CC    X = 3.**2 * 6.0
116. =            X= 0.54000000E 02
116. +READY CV
116. +READY CV*************************************************************** 24  **
116. +READY CV       BEGIN EXECUTION WITH STATEMENT FOLLOWING NO. 2
116. +READY CV
116. +READY       START (2+1)
111. =0101         0.54E 02       0.15E 04
114. =STOP  333
116. +READY CV
116. +READY CV*************************************************************** 25  **
116. +READY CV       WITH THE FORMAT SPECIFIED, OVERRIDE THE STANDARD FOR OUTPUTS
116. +READY CV       NOT UNDER CONTROL OF FORMAT STATEMENTS.
116. +READY CV
116. +READY       EDIT(F10.0)
116. +READY CV
116. +READY CV*************************************************************** 26  **
116. +READY CV       PRINT OUT CURRENT VALUE OF EVERY PROGRAM VARIABLE.
116. +READY CV
116. +READY       PDUMP
     =PDUMP       A=         0.
     =            X=        54.
     =            Y=      1485.
116. +READY CV
116. +READY CV*************************************************************** 27  **
116. +READY CV       RESTORE ALL VARIABLES TO THEIR PRE-EXECUTION VALUES.
116. +READY CV
116. +READY       RESET
116. +READY       PDUMP
     =PDUMP       A=(NOT SET)
     =            X=(NOT SET)
     =            Y=(NOT SET)
116. +READY CV
```

Figure 7C.

gram. When this happens it is obvious to the user, for (besides not rejecting the statement) the system offers line 103. for the next entry, leaving behind number 102. which is now assigned to the newly accepted statement.

When no program is active at the terminal, the terminal is in "command mode." When a program is active ("program mode"), it is a *particular* program, and no statement entered can implicitly refer to any other program. To change from one active program to another, it is always necessary to return first to command mode (using the COMMAND statement). The statement COMMAND wipes out the "active image" of the active program. Therefore, the user should first copy it into his library (with SAVE) if he would ever make it active again (otherwise he must re-enter it, statement by statement). Only the most recent SAVE has abiding effect, replacing that program (if any) which has the same name as the active program. Accordingly, judicious use of SAVE from time to time in constructing a long program will protect the user from the loss that would otherwise result from an inadvertant COMMAND.

The statement for making active a program previously placed in the user's library is LOAD ( ), naming the program. Note that LOAD ( ) does not remove the program from the user's library any more than SAVE obliterates the active image. Both statements merely copy from one place to another. The LOAD ( ) command is effective only in the command mode, but has the same effect as PROGRAM _____ in changing from command to program mode. The statement PROGRAM _____ becomes the program's first statement; LOAD ( ), does not. There is no difference between an active program image made so by loading and one which has been brought to the same point by building on a PROGRAM statement.

The salient difference, as should now be clear, between program mode (+) and command mode (−) is that in program mode statements are kept as part of a savable and executable program whereas in command mode statements are discarded (after immediate ex-

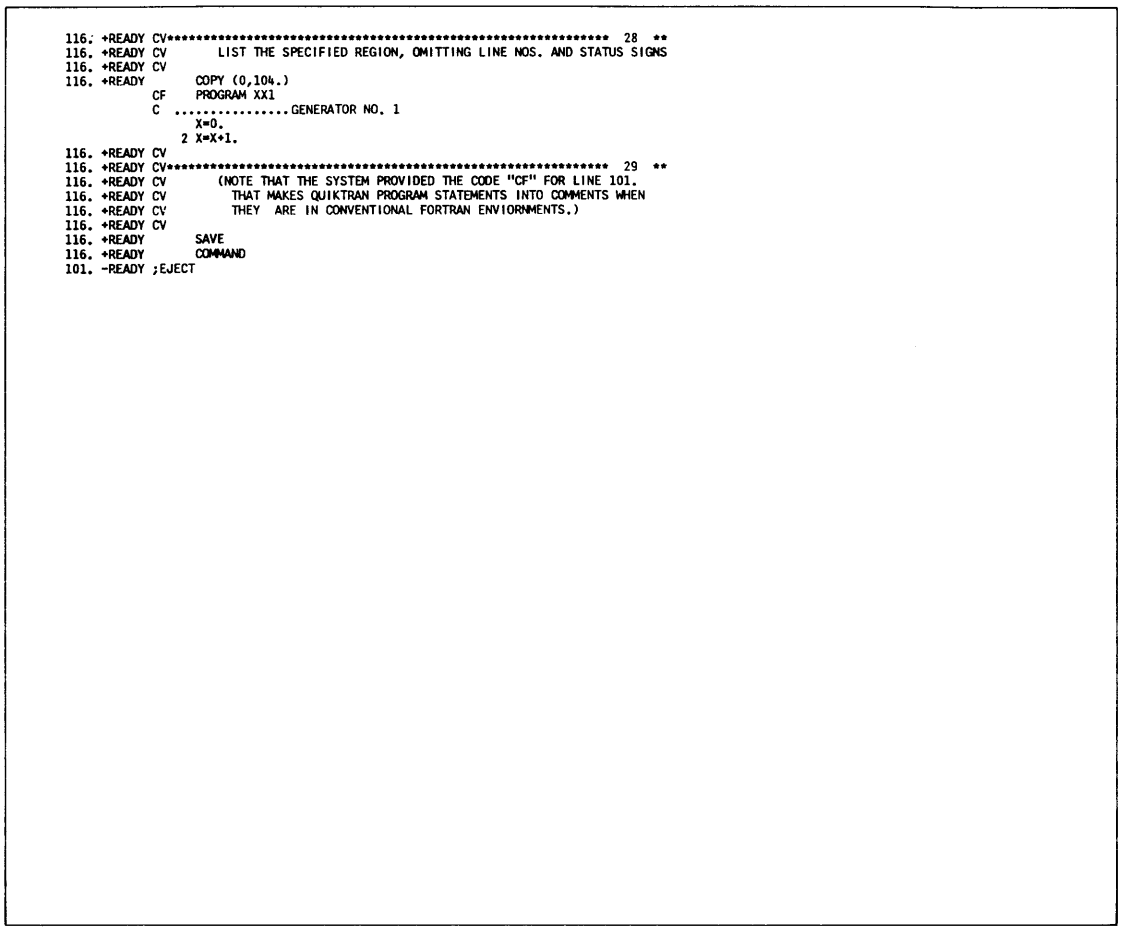


Figure 7D.

ecution, of course). None of the capability of command mode is denied the user in program mode, however.

The third QUIKTRAN mode is "automatic," meaning that the system is in execution (=). After execution, the user is returned to whichever mode he was in before. The user may, therefore, find it convenient to think of the automatic mode as a special way of operating while in either of the other two modes. For instance when a program has just been executed, that program is just as active as it was before – it is still possible to add statements to it, to execute all over again, etc.

Building a Program

Figures 7 through 11 show programs being entered, executed, and otherwise operated upon. It is not the purpose of this part of the Guide to describe in detail all QUIKTRAN statements. These statements are listed alphabetically at the end of this section, and their locations in the illustrations are indexed. It is desirable, however, to orient the prospective user in program-mode operations.

The basic sequence of events in the program mode is the same as that in command mode. The inspection procedure now takes more steps. A statement may now suffer rejection even if free of defects in construction, for it must also avoid being inconsistent with the program already active (see, e.g., Figure 7A, line 107). Also, the system now incorporates every acceptable program statement into the active program. This incorporation replaces the immediate execution that took place in command mode.

The user may, thus, enter statement after statement of the FORTRAN type, building toward the completion of the program he has in mind. To the system, however, there is always a "program," consisting of whatever statements have thus far been accepted into it. Because the system responds on a statement-by-statement basis, there are some differences from the way FORTRAN statements are conventionally treated. Declaratives (like INTEGER, REAL, DIMENSION) must precede executable statements. Also, there is no need to mark the completion of a program with END. How-

ever, QUIKTRAN accepts END (for compatibility with FORTRAN compilers), and will permit no program statements to come after an END statement. The system will also print out a message when the user enters END if it detects DO-loop errors (see Figure 8A, line 117.).

Everything the user enters into the system will be interpreted by the system as in some way relating to the active program. (The only exceptions are of a housekeeping nature, like the terminal commands.) There are principally four kinds of actions the user can initiate: (1) acceptance of a statement as part of the active program; (2) copying of the active program into the user's library; (3) execution of the program itself (or any part of it), or of various service routines, in either case temporarily interrupting the building of a program; and (4) attaching to the active program some qualification that will condition future performance. Some examples of item (3) are: START, LIST, PDUMP, CS and CHECK. Note that after any kind of execution, the system returns the user to that point at which he departed the active program. Examples of item (4) are SNAP and DELTA ( ). Note that the system registers its acceptance of the task thus imposed on it by returning to the + READY condition.

In building a program, the serial entry of sequential program statements has already been discussed. The nonconsecutive entry of sequential statements is made possible by use of the ALTER ( ) statement (Figure 7B). The system responds to an ALTER ( ) statement by deleting the region of the program specified and offering the first of the specified line numbers to the user, who may use that line as though it were offered in the normal sequence. The system will continue to offer lines sequentially until either the system or the user terminates the alter sequence. Instead of specifying a region of the program for deletion, the user may specify a new position lying between existing statements, in which case it is that line number which begins the alter sequence (Figure 10). If at the end of an alter sequence statements have been entered which make the program logically unworkable, the system will so inform the user.

```
101. -READY       PROGRAM XX2
102. +READY C                        .......GENERATOR NO. 2
103. +READY       EQUIVALENCE M,N
103. +RJECT  ELEMENT IN FIELD 'M,N' IS NOT AN OPERATOR
103. +READY       EQUIVALENCE (M,N)
104. +READY       INTEGER A
105. +READY       J=0
106. +READY     1 M=J
107. +READY       J=J+1
108. +READY       K=J
109. +READY       IF (J-1) 4,4,2
110. +READY     2 DO  6 A = 1,N
111. +READY     3 K = K + (J - A)
112. +READY CV
112. +READY CV*****************************************************************  30  **
112. +READY CV       (NOTE THAT THE FOLLOWING STATEMENT WILL MERELY PRINT
112. +READY CV         WHEN EXECUTED BY QUIKTRAN.)
112. +READY CV
112. +READY       4 PUNCH 101,J,K
113. +READY   101 FORMAT (2I10)
114. +READY       GO TO (1,1,1,1,1,1,1,1,1,5),J
115. +READY     5 PAUSE 555
116. +READY       STOP 666
117. +READY       END
117. +ERROR  DO 110. REFERENCES UNDEFINED LABEL 6
117. +ERROR  STATEMENT 4 (112.) REFERENCED FROM OUTSIDE RANGE OF DO BY 109.
118. +READY CV
118. +READY CV*****************************************************************  31  **
118. +READY CV       STATUS WORD "ERROR" - THE SYSTEM HAS DETECTED EITHER THAT
118. +READY CV         PARTS OF YOUR PROGRAM ARE INCONSISTENT OR THAT YOUR
118. +READY CV         PROGRAM IS INCOMPLETE.
118. +READY CV
118. +READY       ALTER (110.,110.)
110. +ALTER     2 DO 3  A = 1,N
110.1+ALTER       ALTERX
118. +READY       START (0)
112. =0101           1         1
112. =0101           2         3
112. =0101           3         6
112. =0101           4        10
112. =0101           5        15
112. =0101           6        21
112. =0101           7        28
112. =0101           8        36
112. =0101           9        45
112. =0101          10        55
115. =PAUSE 555
118. +READY CV
118. +READY CV*****************************************************************  32  **
118. +READY CV       STATUS WORD "PAUSE" - EXECUTION INTERRUPTED BY YOUR
118. +READY CV         INSTRUCTION IN PROGRAM STEP  AT LINE NUMBER SHOWN.
118. +READY CV
118. +READY       START
116. =STOP  666
118. +READY       RESET
118. +READY       SAVE
118. +READY       COMMAND
101. -READY       LOAD (XX2)
118. +READY       LIST (101.,104.)
```

Figure 8A.

```
101. =      CF    PROGRAM XX2
102. =      C                          .......GENERATOR NO. 2
103. =            EQUIVALENCE (M,N)
104. =            INTEGER A
118. +READY CV
118. +READY CV************************************************************ 32.5 **
118. +READY CV       COPY THE ACTIVE PROGRAM IMAGE INTO THIS USER'S LIBRARY -
118. +READY CV         UNDER THE NAME SPECIFIED.
118. +READY CV
118. +READY       SAVE(FACTL 2)
118. +READY CV       (THE SAME PROGRAM IS NOW IN THE LIBRARY TWICE...UNDER TWO
118. +READY CV       DIFFERENT NAMES.)
118. +READY       COMMAND
101. -READY CV
101. -READY CV************************************************************ 33  **
101. -READY CV       REMOVE THE NAMED PROGRAM FROM THIS USER'S LIBRARY.
101. -READY CV
101. -READY       PURGE (XX2)
101. -READY       LOAD(XX2)
101. -ERROR  LOAD REQUEST NOT FULFILLED (PROGRAM NOT IN LIBRARY)
101. -READY CV
101. -READY CV       (PROGRAM XX2 NOW UNLOADABLE BY THAT NAME.)
101. -READY CV
101. -READY       LOAD(FACTL2)
118. +READY       LIST (101.,104.)
101. =      CF    PROGRAM XX2
102. =      C                          .......GENERATOR NO. 2
103. =            EQUIVALENCE (M,N)
104. =            INTEGER A
118. +READY CV       (NOTE THAT THE PROGRAM STATEMENT DEFINING THE BEGINNING OF
118. +READY CV       "FACTL2" REMAINS "PROGRAM XX2".)
118. +READY       COMMAND
101. -READY       PROGRAM FACTL3
102. +READY C                .....GENERATOR NO. 3
103. +READY       A=10.
104. +READY       X=1.
105. +READY       Y=1.
106. +READY    10 PRINT 101,X,Y
107. +READY       X=X+1.
108. +READY       Y=Y+X
109. +READY       IF(A-X) 20,10,10
110. +READY    20 STOP
111. +READY       END
112. +READY       START (0)
106. =XEQER  STATEMENT NUMBER USED DOES NOT EXIST IN PROGRAM
112. +READY CV
112. +READY CV************************************************************ 34  **
112. +READY CV       STATUS WORD "XEQER" - ATTEMPT TO ACCOMPLISH THE IMPOSSIBLE
112. +READY CV         ENDED EXECUTION OF YOUR PROGRAM.
112. +READY CV
112. +READY CV
112. +READY CV************************************************************ 35  **
112. +READY CV       LIST UNEXECUTED STATEMENTS, UNSET VARIABLES.
112. +READY CV
112. +READY       AUDIT
      AUDIT         106. /111.  NOT EXECUTED
      AUDIT        A       NOT USED
      AUDIT        X       NOT USED
      AUDIT        Y       NOT USED
```

Figure 8B.

The SAVE statement copies the program into the user's library under the name used in the program-defining statement. The form SAVE ( ), giving an alias, puts it in under the alias (see Figure 8). The only limit to the number of different times the program image is saved (each time under a different name) is the capacity of the library. Should the user wish, for any reason, to remove a program already in the library, he may do so with a PURGE ( ) statement. This is the only operating statement which, entered in the program mode, does not necessarily refer to the active program. It must always explicitly name the program to be removed.

The user can interrupt the building of a program by calling for any of several kinds of execution. The START ( ) statement executes the program itself (in the specified range). The CS code executes the last arithmetic or input/output statement entered by the user.

Other statements cause the execution of special routines that print out information. The statements INDEX and CHECK compile cross-reference information about the program variables and labels. The AUDIT statement searches the program for unset variables and unexecuted statements. The statements PDUMP and QDUMP display values; LIST, COPY, and NUMBER cause the active image to be displayed at the terminal. Note that NUMBER does this in conjunction with the service of renumbering the program.

But always after interruption for any kind of execution, whether of the program itself or of a service, the user is returned to program mode where he had left off. Note that if any portion of the program has been executed, in response to START ( ) or the process codes (like CS), any variables that have been set thereby will continue to have the values they acquired. *The user must think of his program as having the form and content imparted to it by whatever has affected it during its periods of activity at the terminal.* If he wants his variables to behave as "unset" after they have been affected by execution, he uses RESET.

```
112. +READY CV************************************************************** 36  **
112. +READY CV       STATUS WORD "AUDIT" - PRECEDES EVERY LINE OF OUTPUT CAUSED
112. +READY CV       BY YOUR REQUEST FOR "AUDIT".
112. +READY CV        (AUDIT SHOWS LINES 106. THROUGH  111. NOT EXECUTED.)
112. +READY CV
112. +READY CV
112. +READY CV************************************************************** 37  **
112. +READY CV       LIST REFERENCES TO UNLOCATABLE STATEMENT NOS. AND UNDEFINED
112. +READY CV       VARIABLES... ALSO TO TERMS DEFINED BUT NEVER REFERENCED.
112. +READY CV
112. +READY       CHECK
      CHECK *   101       -106.
      CHECK *FACT!3       +101.
112. +READY CV
112. +READY CV************************************************************** 38  **
112. +READY CV       THE OUTPUT ABOVE SAYS THAT LINE NO. 106 CONTAINS A REFERENCE
112. +READY CV       TO A NONEXISTENT STATEMENT NUMBERED 101.
112. +READY CV       STATUS WORD "CHECK" PRECEDES EVERY LINE OF OUTPUT CAUSED BY
112. +READY CV         YOUR REQUEST FOR A "CHECK"
112. +READY CV
112. +READY CV
112. +READY CV
112. +READY CV************************************************************** 39  **
112. +READY CV       USER WANTS TO INSERT A LINE 106.4.
112. +READY CV
112. +READY       ALTER (106.4)
106.4+ALTER   101 FORMAT (2F10.0)
106.5+ALTER       ALTERX
112. +READY       START (0)
106. =0101          1.        1.
106. =0101          2.        3.
106. =0101          3.        6.
106. =0101          4.       10.
106. =0101          5.       15.
106. =0101          6.       21.
106. =0101          7.       28.
106. =0101          8.       36.
106. =0101          9.       45.
106. =0101         10.       55.
110. =STOP
112. +READY       COMMAND
101. -READY ;EJECT
```

Figure 8C.

The CSS code offers a capability to the user that resembles the immediate execution of command mode. The CSS code results in all subsequent arithmetic and input/output statements receiving immediate execution as well as being accepted into the program. (The CX and CXX codes perform analogous functions, but without the printout.) In all cases, of course, variables remain set to the values they have acquired.

The CC code makes possible an operation almost exactly like command-mode execution. It signals the system *not* to retain the statement that follows it on the same line, but to execute it nonetheless. Note that if the statement sets a variable that appears in the active program, that variable retains the acquired value even though the statement containing it is discarded. In all cases, a subsequently given RESET will revoke the effect execution may have had on any program variable.

The statements that cause the system to take note of a change it must keep track of (rather than execution it must perform immediately) are of two kinds: those controlling the form of certain system activities and those invoking certain testing services.

In the first class are DELTA ( ), EDIT ( ), and AUXOP ( ). The first two perform editorial services on the line increments and on the format of programmed system output, respectively. The AUXOP ( ) statement names a second device, in addition to the printer, to receive all output. In all three cases, these services stay with the program when it is copied into the library and for as many times as the program may again be made active. Only the statements revoking them can terminate their effect.

In this connection, note that there are two different ways that EDIT ( ) may work, depending on whether it is entered in the command mode or in the program mode. The EDIT ( ) statement in command mode does not affect formats in program mode, but will continue to govern output at the terminal whenever operations revert to command mode. The EDIT ( ) statement in program mode governs all unprogrammed formats for executions relating to the active program, during the current and any future periods of activity.

The five test statements (GUARD, SNAP, STEP, TRAIL, and TRAP) as well as XEQER (FINISH) cause the system to remember to watch for certain conditions during any future executions of the active program (again, during this or future periods of being active at the terminal), and to report the occurrence of these conditions to the user should they be detected. Corresponding to each such statement is one that will revoke the service it provides. The CLEAR statement revokes all test services in effect on a program, and resets variables as well.

Input and Output

In the routines shown in Figures 7 and 8, the variables were all set by arithmetic assignment statements in the program. Figure 9 illustrates a program containing a READ statement. By following what happens in execution as a result of the READ statement, the user should begin to get a clear picture of QUIKTRAN input and output.

Refer to line 119. There the user enters a SAVE statement and then starts execution of his program. The first thing the system types in the control field on the next line is neither 119. nor 120., but the number of line 106., which is the READ statement's line. The system follows the line number with an = sign. The = sign shows (as it has in all preceding examples) that the system is delivering output as the result of execution. The significance of the line number is that the output is being caused by the execution of that line, viz., of the READ statement. (Note that not all *system* output is *program* output.)

The output, however, consists solely of the characters in the status-word position, viz., 1101. *When the system output consists of a message in the status-word position beginning with "I," it means that the system is requesting input from the terminal.* This is only logical, because the system is in process of executing a statement (the READ) which requires input, and this is the way the system gets it. The procedure takes two steps: first, inviting the user to enter input; and second, reading the input the user enters. After an "I" status

```
101. -READY       PROGRAM  SORT
102. +READY C
103. +READY C        READS "I" NUMBERS INTO AN ARRAY "A" AND SORTS THE ARRAY.
104. +READY       DIMENSION A (50)
105. +READY       I = 0
106. +READY       READ 101, I, (A(J), J=1, I)
107. +READY   101 FORMAT(I5/(12F6.1))
108. +READY       K = I - 1
109. +READY       DO 15 M = 1,K
110. +READY       L = M + 1
111. +READY       DO 15  N = L,I
112. +READY       TEMP = (A(M)
112. +RJECT  PARENTHESES NOT IN BALANCE
112. +READY       TEMP = (A(M))
113. +READY       A(M) = AMIN1(A(M),A(N))
114. +READY    15 A(N) = AMAX1 (TEMP,A(N))
115. +READY       PRINT 102,A
116. +READY   102 FORMAT (///24H THE ARRAY ORDER NOW IS-/(11F12.1))
117. +READY       STOP 77
118. +READY       END
119. +READY       SAVE
119. +READY       START(0)
106. =1101     48
106. =    2 -032.10034.5-067.8-090.0-087.60054.3-021.00078.10001.20030.5-040.60020.9
106. =    3  -36.2-000.4-001.2-000.9000000-099.90099.90088.8-077.70066.6-014.70095.2
106. =    4 0054.3 -21.00056.70089.0-012.3 -46.7 089.1 -23.4 -56.80089.20098.70065.4
106. =    5 -030.2-060.70059.40027.2-036.50088.1-094.20026.70018.8-016.90071.2  47.0
115. =0102
115. =    2
115. =    3
115. =    4  THE ARRAY ORDER NOW IS-
115. =    5       -99.9       -94.2       -90.0       -87.6       -77.7       -67.8       -60.7       -56.8       -46.7       -40.6
115. =OVFLO       -36.5
115. =    6       -36.2       -32.1       -30.2       -23.4       -21.0       -21.0       -16.9       -14.7       -12.3        -1.2
115. =OVFLO        -0.9
115. =    7        -0.4         0.          1.2        18.8        20.9        26.7        27.2        30.5        34.5        47.0
115. =OVFLO        54.3
115. =    8        54.3        56.7        59.4        65.4        66.6        71.2        78.1        88.1        88.8        89.0
115. =OVFLO        89.1
115. =    9        89.2        95.2        98.7        99.9         0.          0.
117. =STOP  77
119. +READY CV************************************************************  40  **
119. +READY CV       STATUS WORD "I---" - SYSTEM REQUESTS INPUT SO THAT IT MAY
119. +READY CV         EXECUTE A READ STATEMENT UNDER FORMAT STATEMENT "---"
119. +READY CV
119. +READY CV       STATUS WORD "OVFLO" - THIS LINE IS THE REMAINDER OF THE
119. +READY CV         LINE CALLED FOR IN YOUR FORMAT SPECIFICATION, AND STARTED
119. +READY CV         ABOVE.
119. +READY CV
119. +READY       CUMMAND
101. -READY ;EJECT
```

Figure 9.

message, then, although the system is still in control (being in automatic mode), the rest of the line originates with the user. (To recapitulate, the three status messages that are interfaces between system and user are: READY, ALTER, and I - - -.)

The reader will, therefore, interpret the "48" in the first execution line in Figure 9 as having been typed by the user. The "101" part of 1101 tells the user that the system will interpret the input record he enters in accordance with the field specification contained in statement 101 (which is, of course, a FORMAT statement).

The requirements for the first line of input have been satisfied ("I5" in statement 101). But the FORMAT statement specifying the input-field layout for the READ statement shows that there should be more to read on the next line. Accordingly, execution of line 106. is incomplete and the system again invites input. This time, it does not repeat the 1101. Instead, it prints only "2," signifying the second line of input required in order to execute line 106. The user then proceeds, in accordance with the 101 FORMAT statement, to complete the input fields called for. When the user has done this, the print element rests at the beginning of a new line, just under the "1" in 106. = .

The system does not type anything at this time. The program being executed has now to perform the computations that do the sorting. While these computations are being carried out, the system is silent; the print element does not move. If the KEYBOARD switch has been left ON, the PROCEED light will come on at intervals. This is not an invitation to transmit input. (Note that the print element is not in the appropriate position for input.) But the user is being offered an opportunity to interrupt execution with an EOT from the keyboard. He can interrupt only when the PROCEED light is on. If he does interrupt (with an EOT), the system will not resume execution; it will, instead, print the status work BREAK, and a message informing the user what statement was being executed (see Figure 7B). It will then return the user to program mode, offering the line number it was offering when the user started execution. The statement START is sufficient to resume execution of a program thus interrupted (at the point where it had left off). If, during the pause in execution when the PROCEED light is on, the user wishes not to interrupt, he can speed the return to execution by sending EOB. If he does not want even the opportunity to interrupt, he should turn the KEYBOARD switch OFF until he does.

(Note that execution sets "I" to zero before it is set to 48 by the READ statement. This fulfills a technical requirement of the system, viz., that when a variable is used in the same input statement that sets it, that variable must have been set before the input statement is executed. The value to which it is pre-set is of no consequence.)

Output in Figure 9 produces a control field that is analagous in structure to that produced by the request for input. Observe that the statement controlling output format calls for longer lines than the system can print. The system does not suppress the line or its superfluous portion, but prints the line as far as it can before printing the remainder on the next line, which is labeled OVFLO.

Figure 9 also completes the examples of the way the system uses line numbers. An inspection of all figures will show that line numbers always refer to the status message. [Even the following are not exceptions to this: (1) command-mode execution (Figure 6), which is a special case of the line number indicating which statement is being executed; (2) the status word READY, which, in either mode, shows what line number is ready for the next program statement; and (3) the status word RJECT, which, of course, gives the number of the line on which the user attempted to enter the rejected statement.]

Command-Type Operations in Program Mode

In Figure 10, the user begins by finding the value H of an expression he enters in command mode. In this case, what the user has done is to copy a formula — a well-known one in electronics — at the keyboard, substituting his selected quantities for the variables of the formula. In the process of finding H, nothing, of course, is saved. Should the user want to find H for other quantities of the variables, he would have to key in the statement once again with the new values — a requirement he can avoid by using program mode.

The user therefore enters PROGRAM HENRY and on line 102. enters the formula itself as a statement in the now active program. As long as the program is active, he need only set its two variables to the chosen quantities (with CC) and then execute. The START (0) statement would do, but here CS is more efficient. He may repeat this procedure as many times as he cares to.

In the example, the user then saves the program, reloads it, lists it, and decides to add to it. His additions create a program which: (1) *reads* the values into the formula, (2) does the computation, (3) prints H and the variables, and (4) repeats these steps indefinitely. When this version is active, the user's terminal can be made to function as though it were a desk calculator with memory, programmed to compute H for every pair of values X and D fed in.

(Note that at line 101.5 the user requests an alter sequence after line 102. The system, instead of presenting him with line 102.1 in an alter sequence, gives him 103., in the normal sequence. The NOTE informs the user that because there are no statements in the

program following line 102., the alter sequence is not necessary.)

A Sample Problem

In Figure 11, a QUIKTRAN user builds a program for solving an equation. As the listing shows, the user first enters, on lines 104. through 111., the statements that should execute his basic algorithm, without any input or output as yet. He adds an END (line 112.) to make the program FORTRAN-compatible. Then he requests the SNAP service.

There is one variable in this program which has to be set by input, and that variable is DELX. The user now sets it to 0.5. After a false START, the user initiates execution. Although there are no output statements in the program, the user can see execution proceeding because the SNAP service supplies a line of output every time a variable is newly set (identifying the statement being executed in the line-number column). After executing 109., the system tries to execute 110. But the function used there has not been defined, so the system returns the user to line 113. of the program with an appropriate message.

```
101. -READY CV       (THE FOLLOWING LINES SHOW HOW A QUIKTRAN USER  PERFORMS A
101. -READY CV       COMMAND - TYPE OPERATION REPEATEDLY ON VARYING DATA, BY USING
101. -READY CV       THE PROGRAM MODE.)
101. -READY       H = 2.E-9  *50. * (ALOG(2.*50./10.)-1. + 10./50.)
101. =            H= 0.15025851E-06
101. -READY       PROGRAM HENRY
102. +READY         H  =  2.E  -  9 *  X * (ALOG (2. * X/D ) -  1.0 + D/X)
103. +READY CC    X = 50.
103. =            X= 0.50000000E 02
103. +READY CC    D = 10.
103. =            D= 0.10000000E 02
103. +READY CV
103. +READY CV************************************************************  41  **
103. +READY CV       EXECUTE THE PRECEDING ARITHMETIC OR I/O STATEMENT.
103. +READY CV
103. +READY CS
102. =            H= 0.15025851E-06
103. +READY       SAVE
103. +READY       COMMAND
101. -READY       LOAD (HENRY)
103. +READY       LIST
101. =      CF    PROGRAM HENRY
102. =            H=2.E-9*X*(ALOG(2.*X/D)-1.0+D/X)
103. +READY       ALTER(101.1)
101.1+ALTER C        THE INDUCTANCE (HENRY) OVER A DISTANCE (D) BETWEEN 2 WIRES
101.2+ALTER C          OF EQUAL LENGTH.
101.3+ALTER CV
101.3+ALTER CV************************************************************  42  **
101.3+ALTER CV       (FOLLOWING INSTRUCTION WILL HAVE SAME EFFECT AS  1 CONTINUE)
101.3+ALTER CV
101.3+ALTER     1 BACKSPACE 3
101.4+ALTER CV
101.4+ALTER CV************************************************************  43  **
101.4+ALTER CV       READ ACCORDING TO THE STANDARD FOR NON-FORMATTED INPUT.
101.4+ALTER CV
101.4+ALTER       READ 0,X,D
101.5+ALTER       ALTER(102.1)
101.5+NOTE   MODE RESET - ALTER LINE NUMBER 102.1 HAS REACHED UPPER BOUND
103. +READY CV
103. +READY CV************************************************************  44  **
103. +READY CV       PRINT ACCORDING TO THE STANDARD FOR NON-FORMATTED OUTPUT.
103. +READY CV
103. +READY       PRINT 0,X,D,H
104. +READY       GO TO 1
105. +READY       SAVE
105. +READY       START (0)
101.4=1 00  50./10.
103. =0 00     0.50000000E 02   0.10000000E 02   0.15025851E-06
101.4=1 00  60./20.
103. =0 00     0.60000000E 02   0.20000000E 02   0.13501114E-06
101.4=1 00  70./30.
103. =0 00     0.70000000E 02   0.30000000E 02   0.13566231E-06
101.4=1 00  90./60.
103. =0 00     0.90000000E 02   0.60000000E 02   0.13775021E-06
101.4=1 00
101.4=BREAK EXECUTION INTERRUPTED BY TERMINAL USER-LAST EXECUTED  101.4
105. +READY       COMMAND
101. -READY ;EJECT
```

Figure 10.

In the second group of lines numbered 113., the user guards execution of the statement that uses the function. This time, the system does not try to execute the guarded region but instead prints out a message saying that it has encountered a guard condition at line 110. The first time this happens, the user makes the error of entering START, which forces the system to try to execute the statement at 110. This proves impossible because the function is not yet available, so on subsequent occasions, the user supplies (with a CC) a value for the variable Y which 110. sets, and then starts execution one line past 110. The user has effectively substituted his own action for execution of the missing subprogram.

The SNAP outputs show values for X that satisfy the user. He therefore requests an alter sequence beginning with 106.5, and puts statements in his program that will print out the values of both X and Y every time they are newly computed. He closes the alter sequence, and removes the GUARD and SNAP services.

Now the user changes from program SAMPLE to FUNCTION COMPY (Figure 11B), first making sure to SAVE program SAMPLE. The function COMPY is the function called for in line 110. of the main program and the user needs to enter the subprogram defining it. (He starts out using integer symbols for the arguments in the FUNCTION statement. Since the values to be passed to the main program are real, he arranges, using the REAL statement, to have the COMPY variables so regarded.)

After defining the function, making sure to include a RETURN statement, the user inserts values for the variables (with CC lines) in order to test this subprogram. Here, CS would not do for finding COMPY (line 105.), for L must first be computed (104.). So the user gives START (0). He gets an error message when the system tries to execute the RETURN at 107. to the active program, for it is the function that *is* the active program. To find out what value the function has, the user takes a QDUMP.

```
101. -READY CV      (IN THE FOLLOWING LINES, A QUIKTRAN USER BUILDS A PROGRAM.  HE
101. -READY CV        (WRITES A MAIN ROUTINE AND TWO SUB-PROGRAMS.)
101. -READY         PROGRAM SAMPLE
102. +READY C       .......SOLUTION AND PLOT OF A DIFFERENTIAL EQUATION
103. +READY C
104. +READY         X = 0
105. +READY         Y = 1.0
106. +READY         DO 4 I = 1,500,2
107. +READY         IF (X - 1.) 3,2
107. +RJECT  IF(EXPRESSION)N1,N2,N3 IS THE ONLY VALID FORM
107. +READY         IF (X - 1.) 3,2,2
108. +READY       2 STOP 77
109. +READY       3 X = X + DELX
110. +READY         Y = COMPY(X,Y,DELX)
110. +NOTE    THE FOLLOWING ARE FUNCTIONS COMPY
111. +READY CV
111. +READY CV********************************************************************  45  **
111. +READY CV         STATUS WORD "NOTE" BRINGS TO YOUR ATTENTION INFORMATION YOU
111. +READY CV           MIGHT REQUIRE TO AVOID ERROR.
111. +READY CV
111. +READY       4 CONTINUE
112. +READY         END
113. +READY CV
113. +READY CV********************************************************************  46  **
113. +READY CV         EVERY TIME A VARIABLE IS SET, PRINT IT OUT WITH ITS VALUE.
113. +READY CV
113. +READY         SNAP
113. +READY CC      DELX = 00.5000
113. =              DELX= 0.50000000E 00
113. +READY         START
113. +ERROR  START USED TO RESUME AUTOMATIC MODE - NOT AFTER EXECUTING STOP, END, OR LAST STATEMENT IN PROGRAM
113. +READY         START (0)
104. =SNAP          X= 0.
105. =SNAP          Y= 0.10000000E 01
109. =SNAP          X= 0.50000000E 00
113. +ERROR  SUBPROGRAM COMPY IS NOT IN YOUR LIBRARY
113. +ERROR  RETURNING TO MAIN PROGRAM
113. +READY CV
113. +READY CV********************************************************************  47  **
113. +READY CV         STATUS WORD "SNAP" PRECEDES EVERY LINE OF OUTPUT CAUSED BY
113. +READY CV           THE "SNAP" SERVICE YOU REQUESTED.
113. +READY CV
113. +READY CV********************************************************************  48  **
113. +READY CV         RETURN TO USER CONTROL OF EXECUTION OF SPECIFIED REGION (BY
113. +READY CV         PREVENTING AUTOMATIC EXECUTION THEREOF).
113. +READY         GUARD (110.,110.)
113. +READY         START (0)
104. =SNAP          X= 0.
105. =SNAP          Y= 0.10000000E 01
109. =SNAP          X= 0.50000000E 00
110. =GUARD GUARDED STATEMENT ENCOUNTERED DURING EXECUTION
113. +READY CV
113. +READY CV********************************************************************  49  **
113. +READY CV         THE STATUS WORD "GUARD" -- EXECUTION IN THIS REGION IS
113. +READY CV         "SINGLE-CYCLE". IF YOU WANT EXECUTION OF THIS STATEMENT, USE
113. +READY           START
113. +ERROR  SUBPROGRAM COMPY IS NOT IN YOUR LIBRARY
113. +ERROR  RETURNING TO MAIN PROGRAM
113. +READY         START (0)
```

Figure 11A.

```
104. =SNAP        X= 0.
105. =SNAP        Y= 0.10000000E 01
109. =SNAP        X= 0.50000000E 00
110. =GUARD GUARDED STATEMENT ENCOUNTERED DURING EXECUTION
113. +READY CC    Y = X*Y*DELX
113. =            Y= 0.25000000E-00
113. +READY       START (4)
109. =SNAP        X= 0.10000000E 01
110. =GUARD GUARDED STATEMENT ENCOUNTERED DURING EXECUTION
113. +READY CC    Y = X*Y*DELX
113. =            Y= 0.12500000E-00
113. +READY       START (4)
108. =STOP  77
113. +READY       ALTER (106.5)
106.5+ALTER       PRINT 101,X,Y
106.6+ALTER   101 FORMAT (2X,F7.4,F8.5)
106.7+ALTER       ALTERX
113. +READY CV
113. +READY CV*************************************************************** 50  **
113. +READY CV       REMOVE EFFECT OF "GUARD".
113. +READY CV
113. +READY       GUARDX
113. +READY CV
113. +READY CV*************************************************************** 51  **
113. +READY CV       DISCONTINUE SNAP SERVICE.
113. +READY CV
113. +READY       SNAPX
113. +READY       SAVE
113. +READY       COMMAND
101. -READY CV
101. -READY CV*************************************************************** 52  **
101. -READY CV       GO INTO PROGRAM MODE, MAKING THE NAMED ROUTINE ACTIVE.
101. -READY CV       (THE FIRST TWO WAYS TO DO THIS WERE "PROGRAM" AND "LOAD(  )")
101. -READY CV
101. -READY       FUNCTION COMPY(I,J,K)
102. +READY C                                  A SUBPROGRAM WHICH COMPUTES THE NEW
103. +READY C                                       Y IN PROGRAM 'SAMPLE'.
104. +READY       REAL I,J,K,L
105. +READY       L = I*J*K
106. +READY       COMPY = L + J
107. +READY       RETURN
108. +READY       END
109. +READY CC    I = .1
109. =            I= 0.10000000E-00
109. +READY CC    J = .2
109. =            J= 0.20000000E-00
109. +READY CC       K = .3
109. =            K= 0.30000000E-00
109. +READY       EDIT(F8.2)
109. +READY       START (0)
107. =XEQER  RETURN CANNOT BE EXECUTED IF NOT A SUBPROGRAM
109. +READY CV
109. +READY CV*************************************************************** 53  **
109. +READY CV       PRINT OUT CURRENT VALUES OF WHATEVER VARIABLES HAVE CHANGED
109. +READY CV       SINCE LAST DUMP.
109. +READY CV
109. +READY       QDUMP
     =QDUMP       COMPY=    0.21
     =            I=    0.10
```

Figure 11B.

The dump shows values for L and COMPY that satisfy the user. He saves his subprogram, and reloads the main routine. Note that the system re-enters program mode with the line number (113.) it had arrived at when SAMPLE was last active. There has been no logical break in the activity of program SAMPLE.

The user changes the value of DELX from 0.5 to 0.25 (DELX has been set to 0.5 since the execution of the previous CC). Execution shows that the PRINT statement on line 106.5 is delivering output in accordance with FORMAT statement 101. The second iteration of 0101 demonstrates that the function is being set.

Now the user constructs the subprogram (Figure 11C) that will plot a graph of the changing value of Y in SAMPLE. After doing so in lines 101. through 112., the user supplies values for the two variables which the subroutine must receive from the main program. He also requests a printout of K whenever line 108. is executed. Execution discloses that X is always equal to 0.0 and K is equal to 1, because the array ARRAY in the subroutine is empty.

Execution stops when the system runs out of program to execute (see HALT). The system, of course, again offers 113. to the user, who this time enters a READ 0 statement. The READ 0 statement immediately becomes the last (i.e., line 113.) statement in the program. Accordingly, when on his next turn the user enter CS, the system executes the READ 0 statement. Execution takes two steps, externally. In the first step, the system, with 113. =I 00, requests input. The "I 00" means that the input record is to be interpreted as nonformatted (i.e., "self formatting," using separator characters). In the second step, the system reads whatever the user enters in response to I 00, setting the array. Now, with START (0), the user shows that his plot routine is working. (The letter A represents the character placed in ZPLOT to be printed out in the graph. The user may observe that this character is a + sign in the output, but a 1.0 in the statement that sets A. The reason for this is that the 101 FORMAT statement calls for whatever is in the Kth position of ZPLOT to be printed under "A" conversion. The 1.0 is stored in an internal form that is always + under that conversion.)

```
      =             J=    0.20
      =             K=    0.30
      =             L=    0.01
  109. +READY       SAVE
  109. +READY       COMMAND
  101. -READY       LOAD (SAMPLE)
  113. +READY CC    DELX = .25
  113. =            DELX= 0.25000000E-00
  113. +READY       START (0)
106.5=0101     0.     1.00000
106.5=0101     0.2500 1.06250
106.5=0101     0.5000 1.19531
106.5=0101     0.7500 1.41943
106.5=0101     1.0000 1.77429
  108. =STOP  77
  113. +READY       COMMAND
  101. -READY CV
  101. -READY CV************************************************************  54  **
  101. -READY CV    GO INTO PROGRAM MODE, MAKING THE NAMED ROUTINE ACTIVE.
  101. -READY CV       (THE OTHER THREE WAYS TO DO THIS ARE WITH (1) PROGRAM,
  101. -READY CV       (2) LOAD (NAME),  (3) FUNCTION .)
  101. -READY CV
  101. -READY       SUBROUTINE PLOT (A,J)
  102. +READY C ....A SUBPROGRAM FOR PLOTTING " J" VALUES FROM A COMMON ARRAY
  103. +READY C
  104. +READY       DIMENSION ARRAY(500),ZPLOT(100)
  105. +READY       COMMON ARRAY
  106. +READY       DO 2 I=1,J,2
  107. +READY       X=ARRAY(I)
  108. +READY      0K=1.+(ARRAY(I+1)-ARRAY(2))/(ARRAY(J+1)
  108. +READY      4-ARRAY(2))*50.
  109. +READY       DO 1 L=1,K
  110. +READY     1 ZPLOT(K)=A
  111. +READY     2 PRINT 101,X,(ZPLOT(K),L=1,K)
  112. +READY   101 FORMAT(F7.4,100A1)
  113. +READY CC    A=1.
  113. =            A= 0.10000000E 01
  113. +READY CC    J=5
  113. =            J=           5
  113. +READY CV
  113. +READY CV************************************************************  55  **
  113. +READY CV       PRINT OUT VALUE OF ( ) EVERY TIME IT IS SET DURING EXECUTION.
  113. +READY CV
  113. +READY       SNAP(K)
  113. +READY       START (0)
  108. =SNAP        K=           1
  111. =0101   0.    +
  108. =SNAP        K=           1
  111. =0101   0.    +
  108. =SNAP        K=           1
  111. =0101   0.    +
  112. =HALT END OF PROGRAM ENCOUNTERED DURING EXECUTION
  113. +READY CV
  113. +READY CV************************************************************  56  **
  113. +READY CV       STATUS WORD "HALT" - EXECUTION ENDED BECAUSE YOUR PROGRAM DID.
  113. +READY CV
  113. +READY       READ 0,(ARRAY(I),I=1,10)
  114. +READY CS
  113. =I 00  .1/.2/.3/.4/.5/.6/.7/.8/.9/1.0
```

Figure 11C.

```
114. +READY       START (0)
108. =SNAP        K=           1
111. =0101   0.1000+
108. =SNAP        K=          26
111. =0101   0.3000++++++++++++++++++++++++++++
108. =SNAP        K=          51
111. =0101   0.5000++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
113. =I 00  99999999999999999999999999999999999999999999999999999999999999999999999999
113. =XEQER  ILLEGAL CONSTANT IN FIELD STARTING '999999'
114. +READY       ALTER (113.,113.)
113. +ALTER       RETURN
113. +NOTE   MODE RESET - ALTER LINE NUMBER 113.1 HAS REACHED UPPER BOUND
114. +READY       END
115. +READY CV
115. +READY CV*************************************************************  57  **
115. +READY CV       RESET ALL VARIABLES AND DISCONTINUE ALL SERVICES.
115. +READY CV
115. +READY       CLEAR
115. +READY       SAVE
115. +READY       LOAD (SAMPLE)
115. +ERROR  ILLEGAL STATEMENT IN PROGRAM MODE
115. +READY       COMMAND
101. -READY       LOAD (SAMPLE)
113. +READY       ALTER (103.1)
103.1+ALTER       DIMENSION TABLE (500)
103.2+ALTER       COMMON TABLE
103.3+ALTER       ALTER (106.3)
106.3+ALTER       TABLE (I) = X
106.4+ALTER       TABLE (I + 1) = Y
106.4+NOTE   MODE RESET - ALTER LINE NUMBER 106.5 HAS REACHED UPPER BOUND
113. +READY       ALTER (107.1)
107.1+ALTER    2 CALL PLOT (CHAR, I)
107.1+NOTE   PLOT IS A SUBROUTINE NAME
107.1+RJECT  STATEMENT 2 HAS BEEN PREVIOUSLY DEFINED
107.1+ALTER       ALTER (108.,108.)
108. +ALTER    2 CALL PLOT (CHAR,I)
108. +NOTE   PLOT IS A SUBROUTINE NAME
108.1+ALTER       STOP 77
108.2+ALTER       ALTERX
113. +READY CV
113. +READY CV*************************************************************  58  **
113. +READY CV       LIST A "CROSS-REFERENCE SYMBOL TABLE" FOR THE PROGRAM.
113. +READY CV
113. +READY       INDEX
      INDEX       2      +108.  -107.
      INDEX       3      +109.  -107.
      INDEX       4      +111.  -106.
      INDEX     101      +106.6 -106.5
      INDEX *CHAR        -108.
      INDEX *COMPY       -110.
      INDEX *DELX        -109.  -110.
      INDEX  I           +106.  -106.3 -106.4 -108.
      INDEX *PLOT        +108.
      INDEX *SAMPLE      +101.
      INDEX  TABLE        103.1  103.2 +106.3 +106.4
      INDEX  X           +104.  -106.3 -106.5 -107.  +109.  -110.
      INDEX  Y           +105.  -106.4 -106.5 +110.
113. +READY CV
113. +READY CV*************************************************************  59  **
```

Figure 11D.

The second time execution arrives at the READ statement, and the system requests input, the user feeds in an illegally high number, causing an XEQER and a return to program mode, this time offering line 114. The user, with ALTER (113., 113.), deletes the READ 0 statement with which he filled up the subroutine's array ARRAY. In its place, he inserts RETURN, which makes the subroutine usable by a calling program. (Observe the system reminding the user that he has overstepped the alter sequence.) The user puts END after RETURN, and before saving the subroutine, CLEARS it of whatever it has acquired by way of services and assignment values.

With three programs in his library, the user now makes the main routine SAMPLE active again (see Figure 11D, at 103.1 + ALTER). In order to make subroutine PLOT usable, the user provides declaratives setting up an array TABLE and putting it in COMMON with ARRAY. The statement COMMON TABLE makes it possible to pass values between TABLE (in SAMPLE) and ARRAY (in PLOT). He programs (at 106.3) the placement of successive values of X and Y in TABLE, from which ARRAY in PLOT will be filled. Then he provides for branching to PLOT when SAMPLE has completed execution (i.e., just before the STOP in SAMPLE). (He makes a mistake, calling for 107.1 to be supplied him. The system informs the user that, in that case, he would have two statements numbered 2. The user follows with entires that, in effect, enter the CALL PLOT statement and move the "2" from the STOP 77 to CALL PLOT.) After terminating the alter sequence, the user calls for a complete cross-reference symbol table of his program (INDEX).

Now (on the line following the INDEX), the user enters a value for the character, CHAR, which SAMPLE passes to PLOT. The other variable in this program that requires setting from input quantities is already set (DELX = .5). This is the first time the user has tried execution when all three of his routines should be working, so he institutes a TRAIL. The TRAIL service shows transfers to COMPY before each computation of X and Y. The interspersed program output shows the results of that computation. The user may, therefore, infer that the return to the main routine was also made. The last TRAIL message shows a transfer to the PLOT subroutine after the last pair of values has been printed out.

Having found that all parts of his program are working, the user then calls for the renumbering (Figure 11E) of his program, after first calling for a new increment, with DELTA (10., 1.0), of 10 between line numbers and 1 between lines in the alter sequences. At the end of the resulting printout, the user then enters an alter sequence in which he adds some input and output statements to his program. The added statements (beginning with 262.) will result in: a message to the user reading "ENTER DELX(F4.7) AND PLOT CHARACTER"; a request from the system for input to DELX and CHAR; the printout of a title ("SOLUTION OF A DIFFERENTIAL EQUATION") and headings for the columns message to the user reading "ENTER DELX(F4.7) AND PLOT CHARACTERS"; a request from the system for input to of X and Y. He next executes the now completed program. The results of his latest additions can be seen in the first four lines of execution in Figure 11F.

At the completion of execution, he positions the paper at the top of a new page and enters the statement AUXOP (PUNCH) followed by COPY. The effect of AUXOP (PUNCH) is that for every line of printed output, a card is punched. Thus, as COPY is executed, the user acquires a punched deck of program SAMPLE.

When the system has completed listing and punching his program, the user signs off with ;EXIT. Now, no user code identifies the information flow to and from his terminal. Observe the ERROR message that follows the user's attempt to LOAD (SAMPLE).

Last, the user sends a ;FINISH statement, which ends this conversational QUIKTRAN session, since the system will no longer either invite or scan conversational input from the terminal. There is still a data connection, but, to begin conversational operations again, the user would have to re-establish a ;CONSOLE connection.

```
113. +READY CV       STATUS WORD "INDEX" PRECEDES EVERY LINE OF OUTPUT CAUSED
113. +READY CV         BY YOUR REQUEST FOR AN INDEX.
113. +READY CV         AN "INDEX" IS A LIST OF EVERY NAME OR LABEL IN THE ACTIVE
113. +READY CV         PROGRAM, GIVING THE FOLLOWING INFORMATION FOR EACH NAME OR
113. +READY CV         LABEL - (1) ALL THE LINES, MARKED BY "-", CONTAINING REFER-
113. +READY CV         ENCES TO IT; (2) ALL THE LINES, MARKED BY "+", WHERE IT IS
113. +READY CV         THE LEFT-MOST ELEMENT; (3) AND THE LINES, UNMARKED, WHERE
113. +READY CV         DECLARATIVES MAKE REFERENCE TO IT. IF THE ITEM IS STARRED
113. +READY CV         WITH AN ASTERISK (*) IT HAS BEEN REFERENCED BUT NOT DEFINED
113. +READY CV
113. +READY CC     CHAR = 1.
113. =           CHAR= 0.10000000E 01
113. +READY CV
113. +READY CV*************************************************************  60  **
113. +READY CV       APPLY "TRAP"SERVICE TO SUBPROGRAM LINKS
113. +READY CV
113. +READY       TRAIL
113. +READY       START (0)
106.5=0101     0.     1.00000
110. =TRAIL CONTROL TRANSFERED FROM SAMPLE TO COMPY
106.5=0101     0.5000 1.25000
110. =TRAIL CONTROL TRANSFERED FROM SAMPLE TO COMPY
106.5=0101     1.0000 1.87500
108. =TRAIL CONTROL TRANSFERED FROM SAMPLE TO PLOT
111. =0101   0.    +
111. =0101   0.5000+++++++++++++++
111. =0101   1.0000++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
108.1=STOP  77
113. +READY CV
113. +READY CV*************************************************************  61  **
113. +READY CV       STATUS WORD "TRAIL"  PRECEDES EVERY MESSAGE PUT OUT IN
113. +READY CV         RESPONSE TO REQUEST FOR TRAIL.
113. +READY CV
113. +READY CV*************************************************************  62  **
113. +READY CV       STOP THE "TRAIL" SERVICE.
113. +READY CV
113. +READY       TRAILX
113. +READY       DELTA (10.,1.0)
113. +READY       NUMBER(201.)
201. =      CF    PROGRAM SAMPLE
211. =      C     .......SOLUTION AND PLOT OF A DIFFERENTIAL EQUATION
221. =      C
231. =            DIMENSION TABLE(500)
241. =            COMMON TABLE
251. =            X=0
261. =            Y=1.0
271. =            DO 4 I=1,500,2
281. =            TABLE(I)=X
291. =            TABLE(I+1)=Y
301. =            PRINT 101,X,Y
311. =       101 FORMAT(2X,F7.4,F8.5)
321. =            IF(X-1.)3,2,2
331. =          2 CALL PLOT(CHAR,I)
341. =            STOP 77
351. =          3 X=X+DELX
361. =            Y=COMPY(X,Y,DELX)
371. =          4 CONTINUE
381. =            END
391. +READY       ALTER(262.)
```

Figure 11E.

```
262. +ALTER   102 FORMAT (43H ENTER DELX(F7.4) AND PLOT CHARACTER       )
263. +ALTER       PRINT 102
264. +ALTER       READ 103, DELX, CHAR
265. +ALTER   103 FORMAT (1X,F7.4,A1)
266. +ALTER   104 FORMAT (43H SOLUTION OF A DIFFERENTIAL EQUATION       )
267. +ALTER       PRINT 104
268. +ALTER   105 FORMAT (5X,1HX,7X,1HY)
269. +ALTER       PRINT 105
270. +ALTER       ALTERX
391. +READY       SAVE
391. +READY       START (0)
263. =0102   ENTER DELX(F7.4) AND PLOT CHARACTER
264. =1103   00.1000=
267. =0104   SOLUTION OF A DIFFERENTIAL EQUATION
269. =0105       X       Y
301. =0101     0.     1.00000
301. =0101     0.1000 1.01000
301. =0101     0.2000 1.03020
301. =0101     0.3000 1.06111
301. =0101     0.4000 1.10355
301. =0101     0.5000 1.15873
301. =0101     0.6000 1.22825
301. =0101     0.7000 1.31423
301. =0101     0.8000 1.41937
301. =0101     0.9000 1.54711
301. =0101     1.0000 1.70182
301. =0101     1.1000 1.88902
111. =0101   0.    =
111. =0101   0.1000=
111. =0101   0.2000==
111. =0101   0.3000===
111. =0101   0.4000======
111. =0101   0.5000=========
111. =0101   0.6000============
111. =0101   0.7000================
111. =0101   0.8000=====================
111. =0101   0.9000============================
111. =0101   1.0000=======================================
111. =0101   1.1000=================================================
341. =STOP  77
391. +READY ;EJECT
```

Figure 11F.

```
391. +READY CV
391. +READY CV************************************************************  63  **
391. +READY CV       FOR EVERY LINE OF PRINTED OUTPUT, PUNCH ONE CARD.
391. +READY CV
391. +READY       AUXOP(PUNCH)
391. +READY       COPY
           CF     PROGRAM SAMPLE
           C      .......SOLUTION AND PLOT OF A DIFFERENTIAL EQUATION
           C
                        DIMENSION TABLE(500)
                  COMMON TABLE
                  X=0
                  Y=1.0
                102 FORMAT(43H ENTER DELX(F7.4) AND PLOT CHARACTER       )
                  PRINT 102
                  READ 103,DELX,CHAR
                103 FORMAT(1X,F7.4,A1)
                104 FORMAT(43H SOLUTION OF A DIFFERENTIAL EQUATION       )
                  PRINT 104
                105 FORMAT(5X,1HX,7X,1HY)
                  PRINT 105
                  DO 4 I=1,500,2
                  TABLE(I)=X
                  TABLE(I+1)=Y
                  PRINT 101,X,Y
                101 FORMAT(2X,F7.4,F8.5)
                  IF(X-1.)3,2,2
                  2 CALL PLOT(CHAR,I)
                  STOP 77
                  3 X=X+DELX
                  Y=COMPY(X,Y,DELX)
                  4 CONTINUE
                  END
391. +READY       AUXOP
391. +READY CV
391. +READY CV
391. +READY CV************************************************************  64  **
391. +READY CV       SIGN THIS USER OFF (WITHOUT AFFECTING TERMINAL OPERATION).
391. +READY CV
391. +READY ;EXIT
391. +READY       COMMAND
101. -READY       LOAD (SAMPLE)
101. -ERROR  USER IDENTIFICATION REQUIRED TO LOCATE USER LIBRARY FOR LOAD, SAVE, AND SUBROUTINE AND FUNCTION CALLS
101. -READY CV
101. -READY CV************************************************************  65  **
101. -READY CV       TERMINATE CONVERSATIONAL RESPONSES FROM QUIKTRAN. (TERMINAL
101. -READY CV       WILL REMAIN USABLE UNTIL DISCONNECT.)
101. -READY CV
101. -READY ;FINISH

000   READY
```

Figure 11G.

Batch Processing

Figure 5 showed the user entering ;CONSOLE and thereby getting into conversational operation with the system. This was directly after the system had accepted his ;USER ( ) command. At that time he had two other options besides ;CONSOLE. These options were ;INPUT ( ) and ;OUTPUT ( ), either of which would have placed him in batch mode. Figure 12 shows him choosing the former.

When, as in Figure 12, the system is in batch mode, it is capable of receiving statements in batches, saving them up, and operating on them as a unit, i.e., not conversationally, but in the more conventional way. This means that the user will not control the execution of his program directly, as he does in conversational operation. Such control as is necessary will be exercised by the system, and the program will be executed at another time.

Interaction between the user and the system in batch mode is still "conversational," but the user can elicit activity from the system only with the terminal commands. In other words, operations are conversational for housekeeping only. (All terminal commands illustrated in Figure 6, for example, still work in batch mode.) Otherwise the system does not respond to each statement entered, except to detect the end of its transmission, put it away, and signal (with the usual READY) that it is ready for the next line.

At the beginning of Figure 12, conditions at the terminal are the same as those at the end of Figure 11, after the user transmitted ;FINISH. These are also the conditions after the system receives ;USER ( ), when the user first signs on. The system will be idle until it receives one of the following three commands:

;CONSOLE
;INPUT ( )
;OUTPUT ( )

The system will be in one of two states at this time: either it will be capable of responding to the ;CONSOLE request for interaction or it will be capable only of batch processing; a message at sign-on time will show which. If the message indicates that "full QUIKTRAN service" is available, the ;CONSOLE request will be honored. Batch requests will also be honored but, batches received will not be processed during the period of full service. If the message indicates "batch only" a ;CONSOLE command will be rejected.

In Figure 12, the user signifies his intention to transmit a job to the computing center by entering ;INPUT ( ). The parenthesized data is the estimated number of cards to be read for transmission and the destination of output from job. The latter is either the user's two-character code or, as in the figure, the number 00 signifying that the output will be printed at the computing center and not a terminal. The system responds with 001 READY and the next card read is considered to be the first of a series that the command ;FINISH will terminate (ending) with the card preceding ;FINISH). The system responds (1) with a message giving the job an identifying designation containing the user's code and (2) with 001 READY. Note that, in batch mode, the lines are numbered sequentially beginning with 001 instead of 101, and there is no status sign (+ or −), as in the conversational modes.

The job, transmitted to the computing center as shown at the beginning of Figure 12, consists of a program written in FORTRAN and control cards which make it recognizable to the System Monitor (IBSYS) of the Operating System. See the section "Control Cards Used in Batch Input Jobs" for instructions for setting up a FORTRAN job for batch processing. Any job that is executable under the System Monitor, i.e., FORTRAN, COBOL, assembly language, etc., can be transmitted to the system in this manner. For the program shown, the control cards request compilation into machine code without execution. (They also specify that no deck be punched.) The terminal command ;FINISH signals the system that the preceding line of input was the last of the batch input. Accordingly the system (1) tells the user if there is any output waiting for him and (2) informs him that it is again idle and READY. In Figure 12, there is no output available yet, so there is no message. Again the user has three options and again he chooses ;INPUT. The control cards for the program shown specify compilation and execution without a listing. When the requested execution has been completed, the terminal user will be notified by a message that output is available for him (see Figure 13).

During the transmission of this batch, the user avails himself of the full QUIKTRAN service to build a program segment conversationally. At the end of the conversational session, the user tries to cause the output of the batch first transmitted to be diverted from the computing center to his terminal. His ;ROUTE command is, however, rejected because the command immediately preceding it (his ;FINISH command) returned him to the batch mode, which he had been in before he entered the conversational session. Whenever a user has interrupted a batch input or output transmission to enter a conversational operation, completion of the conversational operation causes the system to revert to the batch mode in readiness to carry on the batch transmission. However, the transmission will not continue unless specifically directed to by the command ;RESUME ( ), which gives the identification of the transmission to be resumed, as shown in Figure 12. The figure also shows a successful transmission of the ;ROUTE ( ) command (after ;FINISH of the last input batch).

```
000   READY ;INPUT(015,00)
            BATCH JOB ME0001
001   READY $JOB              GENERATOR NO. 3
002   READY $IBJOB MEJDOE   NOGO,NOMAP
003   READY $IBFTC MEJDOE   LIST,NODECK
004   READY          A=20.
005   READY          X=1.
006   READY          Y=1.
007   READY       10 PRINT 101,X,Y
008   READY          X=X+1.
009   READY          Y=Y+X
010   READY          IF(A-X)20,10,10
011   READY      101 FORMAT(2F10.0)
012   READY       20 STOP
013   READY          END
014   READY $IBSYS
015   READY ;FINISH

000   READY ;INPUT(015,ME)
            BATCH JOB ME0002
001   READY $JOB                GENERATOR NO. 2
002   READY $IBJOB MEJDOE   GO,NOMAP
003   READY $IBFTC MEJDOE   NOLIST,NODECK
004   READY          J=0
005   READY        1 M=J
006   READY          J=J+1
007   READY          K=J
008   READY          IF(J-1)4,4,2
009   READY        2 DO 3 L=1,M
010   READY ;CONSOLE
            INTERRUPT ME0002
101. -READY          PROGRAM INTEGER
101. -RJECT   NAME INTEGE IS LONGER THAN 6 CHARACTERS
101. -READY          PROGRAM INTGR
102. +READY          J = 0
103. +READY        1 PRINT 101,J
104. +READY          1 FORMAT (I3)
104. +RJECT   STATEMENT 1 HAS BEEN PREVIOUSLY DEFINED
104. +READY      101 FORMAT (I3)
105. +READY          J = J+1
106. +READY          GO TO 1
107. +READY          END
108. +READY ;FINISH

010   READY ;ROUTE(ME0001,00,ME)
010   RJECT ILLEG REQ
010   READY ;RESUME(ME0002)

010   READY        3 K=K+(J-L)
011   READY        4 PRINT 101,J,K
012   READY      101 FORMAT (2I10)
013   READY          GO TO (1,1,1,1,1,1,1,1,1,5),J
014   READY        5 STOP 666
015   READY          END
016   READY $ENTRY
017   READY $IBSYS
018   READY ;FINISH

000   READY ;ROUTE(ME0001,00,ME)
```

Figure 12A

```
000   READY ;INPUT(015,ME)
            BATCH JOB ME0003
001   READY $JOB                SORT
002   READY $IBJOB MEJDOE   GO,NOMAP
003   READY $IBFTC MEJDOE   NOLIST,NODECK
004   READY $RECOM          SORT
005   READY $ENTRY
006   READY      10
007   READY -032,10034,5-067,8-090,0-087,60054,3-021,00078,10001,20030
008   READY $IBSYS

            QUIKTRAN SERVICE FOR BATCH ONLY
009   READY ;FINISH

000   READY ;OUTPUT[ME0002,ME]
000   RJECT JOB UNAVAL
000   READY ;OUTPUT[ME0001,ME]
000   RJECT JOB UNAVAL
000   READY ;OUTPUT[ME0003,ME]
000   RJECT JOB UNAVAL
000   READY
```

Figure 12B

QUIKTRAN also affords the batch-mode capability of directing the compilation or compilation and execution of a program contained in the user's own library. Thus, a user can write and debug a program, put it in his library (with SAVE), and then, in batch mode, direct that it be compiled and executed. This has the advantage of not taking up terminal time, as execution in the program mode does; it is the natural way to execute a long program or one with more than a small amount of data to work on. As the figure shows, the user transmits the same control statements he would use in sending any job for the Operating System; where the program deck would come, the user substitutes one card, the $RECOM card. This is the only Operating System control card the user needs to know that is not in the general Operating System vocabulary. It tells the system that the input batch is to be found at a given location in the user's library, instead of at the terminal.

At any time that a ;ROUTE ( ) command is valid, so is a ;PURGE ( ) command. As Figure 13 shows, ;PURGE ( ) names a job, the results of which are not to be put out at the terminal. If the computer has not yet processed the job, nothing will be saved. Otherwise, results will be on file, although inaccessible to a terminal command.

When output is ready, the user is signalled by a message to that effect after his next ;FINISH or ;USER ( ) command, and at no other time. If the user (1) does not transmit ;INPUT ( ) or ;CONSOLE, or some other terminal command, like ;PURGE ( ) or (2) does not request a specific output, the system will, after the lapse of about a minute, commence to transmit to the user all the output, job after job, that has been produced for that user group identification. Once this process is underway, the user is not helpless to alter it. He may interrupt it by transmitting EOT whenever the system reaches the top of a page in the output listing. The interruption may be temporary, as an interruption to do conversational operation (;CONSOLE), or it may be permanent. To make it permanent, the user enters the statement ;CANCEL, which terminates the current transmission of output and causes the system to immediately begin transmission of output of the next job, if there is one.

```
$**** BATCH ME0001 (ME)  USER MEJDOE  0008  0027 0106  0107
$JOB            GENERATOR NO. 3
$IBJOB MEJDOE  NOGO,NOMAP
11555  UNITS   U00   U02   U03   000   000   000
$IBFTC MEJDOE  LIST,NODECK
```

```
GENERATOR NO. 3                          FORTRAN SOURCE LIST                    07/19/65      PAGE  1
      ISN      SOURCE STATEMENT

        0 $IBFTC MEJDOE  LIST,NODECK
        1       A=20.
        2       X=1.
        3       Y=1.
        4    10 PRINT 101,X,Y
        5       X=X+1.
        6       Y=Y+X
        7       IF(A-X)20,10,10
       10   101 FORMAT(2F10.0)
       11    20 STOP
       12       END
```

```
          INTERRUPT ME0001
000   READY ;CANCEL
```

```
$**** BATCH ME0002 (ME)  USER MEJDOE  0010  0027 0107  0111
$JOB              GENERATOR NO. 2
$IBJOB MEJDOE  GO,NOMAP
11555  UNITS   U00   U02   U03   000   U04   000
$IBFTC MEJDOE  NOLIST,NODECK
```

```
GENERATOR NO. 2                          FORTRAN SOURCE LIST                    07/19/65      PAGE  1
    ISN      SOURCE STATEMENT

      0 $IBFTC MEJDOE  NOLIST,NODECK
      1       J=0
      2     1 M=J
      3       J=J+1
      4       K=J
      5       IF(J-1)4,4,2
      6     2 DO 3 L=1,M
      7     3 K=K+(J-L)
     11     4 PRINT 101,J,K
     12   101 FORMAT (2I10)
     13       GO TO (1,1,1,1,1,1,1,1,1,5),J
     14     5 STOP 666
     15       END
```

```
GENERATOR NO. 2                        IBMAP ASSEMBLY MEJDOE                    07/19/65      PAGE  2

  NO MESSAGES FOR ABOVE ASSEMBLY
```

```
       GENERATOR NO. 2                        IBLDR -- JOB   MEJDOE                    07/19/65      PAGE  3

OBJECT PROGRAM IS BEING ENTERED INTO STORAGE.
       1         1
       2         3
       3         6
       4        10
       5        15
       6        21
       7        28
       8        36
       9        45
      10        55
```

```
       GENERATOR NO. 2                                                                PAGE  4

$IBSYS
 END BATCH ME0002
```

Figure 13A

```
$**** BATCH ME0003 (ME)  USER MEJDOE  0016  0027 0109  0117
$JOB              SORT
$IBJOB MEJDOE  GO,NOMAP
11555  UNITS   000   002   003   000   004   000
$IBFTC MEJDOE  NOLIST,NODECK
```

```
SORT                                FORTRAN SOURCE LIST                    07/19/65      PAGE  1
    ISN       SOURCE STATEMENT

      0 $IBFTC MEJDOE  NOLIST,NODECK
        CF     PROGRAM SORT
        C         READS 'I' NUMBERS INTO AN ARRAY 'A' AND SORTS THE ARRAY
        C
      1        DIMENSION A(50)
      2        I=0
      3        READ 101,I,(A(J),J=1,I)
     11    101 FORMAT(I5/(12F6.1))
     12        K=I-1
     13        DO 15 M=1,K
     14        L=M+1
     15        DO 15 N=L,I
     16        TEMP=A(M)
     17        A(M)=AMIN1(A(M),A(N))
     20     15 A(N)=AMAX1(TEMP,A(N))
     23        PRINT 102,A
     24    102 FORMAT(//24H THE ARRAY ORDER NOW IS:/(11F12.1))
     25        STOP 77
     26        END
```

```
SORT                                IBMAP ASSEMBLY MEJDOE                  07/19/65      PAGE  2

  NO MESSAGES FOR ABOVE ASSEMBLY
```

```
       SORT                                 IBLDR -- JOB   MEJDOE                  07/19/65       PAGE  3

  OBJECT PROGRAM IS BEING ENTERED INTO STORAGE.

 THE ARRAY ORDER NOW IS:
      -90.0       -87.6       -67.8       -32.1       -21.0         1.2        34.5        54.3        78.1       300.0
        0.0         0.0         0.0         0.0         0.0         0.0         0.0         0.0         0.0         0.0
        0.0         0.0         0.0         0.0         0.0         0.0         0.0         0.0         0.0         0.0
        0.0         0.0         0.0         0.0         0.0         0.0         0.0         0.0         0.0         0.0
        0.0         0.0         0.0         0.0         0.0         0.0
```

```
       SORT                                                                                    PAGE  4

  $IBSYS
   END BATCH ME0003
  000   READY
```

Figure 13B

The QUIKTRAN Vocabulary

The numbers following entries refer to locations in the sample listing (Figures 5 through 11).

SYSTEM OUTPUT	COMMENT NO.	USER INPUT	COMMENT NO.
+ALTER	19	ALTER ( )	18, 39
AUDIT	36	ALTERX	20
		AUDIT	35
		AUXOP ( )	63
		AUXOP	–
=BREAK	17		
−CANCL	6	C	14
+CANCL	–	CALL _____	58
		CC	23
CF	29		
		CHECK	37
CHECK	38	CLEAR	57
		COMMAND	11
		;CONSOLE	2
		COPY	28
		CS	41
		CSS	–
		CSS*	–
		CV	2
		CX	–
		CXX	–
		DELTA ( )	62
		DELTA	–
+ERROR	31	;ECHO ( )	3
		EDIT ( )	25
		EDIT	–
		;EJECT	5
		END	31
		;EXIT ( )	64
		;FINISH	65
		FUNCTION	52
=GUARD	49	GUARD	48
=HALT	10, 56		
=I − − −	40		
INDEX	59		
		GUARDX	50
		INDEX	58
		LIST	13, 21
		LOAD ( )	12
		NUMBER	62
+NOTE	45		
=O − − −	15		
=OVFLO	40		
=PAUSE	32	PDUMP	26
		PRINT 0	44
=PDUMP	27	PROGRAM _____	7
		PUNCH	30
		PURGE ( )	33
=QDUMP	53	QDUMP	53
−READY	1	READ 0	43
+READY	10	RESET	27
−RJECT	1		
+RJECT	45		
=SNAP	47	SAVE	8
		SAVE ( )	32.5
		;SEND ( )	4
=STOP	15	SNAP	46, 55
		SNAPX	51
		START	32
		START ( )	9, 24
		STEP	–
		STEPX	–
		SUBROUTINE ___ ( )	54
=TRAIL	61	TRAIL	60
=TRAP	17	TRAILX	62
		TRAP	16
		TRAPX	22
		;USER ( )	2
		WRITE	–
=XEQER	34	XEQER (FINISH)	–
		XEQER	–

Appendix A: Comparison With Fortran IV

Serious attention has been paid to maintaining compatibility between the QUIKTRAN System and the IBM 7040/7044 FORTRAN IV processors. Programs written in the language described in this publication are acceptable without change to the 7040/7044 FORTRAN IV processor. Conversely, FORTRAN IV programs are acceptable to the QUIKTRAN System with the limitations given below.

The following limitations apply to FORTRAN IV programs to make them acceptable to the QUIKTRAN 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 comments may appear anywhere in the program.
3. As happens 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. Logical statements and relational operators are not permitted.
   d. Restrictions on the use of continuation cards.
   e. No magnetic tape input or output. However, the 1052 is used for statements that require tape units.
   f. Real constants up to eight digits, with magnitude within the range of 10^{-38} to 10^{38} or zero.
   g. Integer constants up to ten digits.
   h. The maximum size of an input/output record is 120 characters, except card records, which are limited to 80 characters.
   i. Arrays cannot be arguments of functions and subroutines, but must be passed through COMMON.
   j. A statement number must not exceed 199.
   k. The number of constants, variables, arrays, or functions must be less than 190 and the number of operators in a statement is limited to 55.
   l. Restrictions on the form of the EQUIVALENCE statement.
   m. Library function names and built-in function names are reserved.
   n. Declarative statements must precede the first executable statement. If a DIMENSION statement is used, it must appear as the first declarative statement.
   o. No dimension information may appear in type-defining statements.
   p. No DATA statements are allowed.
   q. The READ(m) and WRITE(m) statements may not be used.
   r. Built-in functions will not accept more than two arguments.
   s. In a function subprogram, the name of that function may not appear in a type-defining statement.
   t. No assigned GO TO statements are allowed.
   u. No ASSIGN statements are allowed.
   v. Alphameric data is not allowed as an argument in a function or subroutine call.
   w. The BACKSPACE, REWIND, and END FILE statements do not cause any operation to take place.
   x. Machine indicator tests are not available.
   y. Input/output scale factors are not permitted.

Appendix B: Reserved Names

The user may not use a reserved name as the name of a variable, array, subprogram, etc. He may use it only for the purpose for which it is intended.

All names of library functions, built-in functions, and those operating statements that function as system subroutines (i.e., program-called services) are reserved names. The following lists contain all of the reserved names in alphabetical order under their proper categories:

BUILT-IN FUNCTIONS

ABS	DIM	MAX0
AINT	FLOAT	MAX1
AMAX0	IABS	MIN0
AMAX1	IDIM	MIN1
AMIN0	IFIX	MOD
AMIN1	INT	SIGN
AMOD	ISIGN	

LIBRARY FUNCTIONS

ALOG	COS
ALOG10	EXP
ARCOS	SIN
ARSIN	SQRT
ATAN	TANH
ATAN2	

PROGRAM CALLED SERVICES

AUDIT	INDEX	STEP
CHECK	LIST	STEPX
CLEAR	PDUMP	TRAIL
COPY	QDUMP	TRAILX
EDIT	RESET	TRAP
GUARD	SNAP	TRAPX
GUARDX	SNAPX	

Appendix C: Special Characters

The following chart gives the special characters that appear on a 1052 keyboard having the H character set.

CHARACTER	CARD CODE	REMARKS
:	8-5	valid
'	8-4	valid
>	8-6	valid, but stored as =
=	8-3	valid
<	12-8-6	valid, but stored as +
+	12	valid
[	12-8-5	valid, but stored as (
(	0-8-4	valid
]	11-8-5	valid, but stored as)
)	12-8-4	valid
,	0-8-1	valid
/	0-1	valid
-	11	valid
$	11-8-3	valid
.	12-8-3	valid
;	11-8-6	valid
b	8-2	valid, but is stored as :
"	8-1	valid
*	11-8-4	valid
?	12-8-2	valid
!	11-8-2	valid
|||	0-8-7	invalid
△	11-8-7	invalid
\	0-8-6	invalid
EOB	0-6-9	valid, but not stored

Appendix D: 1050 Terminal Input Error Messages

The following error messages can be sent to a 1050 terminal because the 7740 finds an error in input. In all cases, the input message is discarded by the 7740. Error messages generated by the 7040/7044 are self-explanatory and are therefore not listed.

MESSAGE	EXPLANATION
RJECT ILLEG CHAR	The input contained an invalid character.
RJECT MSG LENGTH	The input exceeded the maximum number of characters – 120 for conversational, 80 for batch.
CANCL PREV LINE	The input was cancelled by the terminal user.
RJECT KB TIMEOUT	The user at the terminal keyboard allowed too much time to elapse between the entering of two characters.
RJECT XMT ERROR	A transmission error occurred so that the input message was not received correctly at the 7740.
RJECT ENDMSG EOT	The input message was invalid because the end of message was an EOT instead of an EOB.
RJECT ILLEGAL REQ	The request made in a terminal commands is invalid because: a. the command name is invalid b. a parameter is invalid c. the punctuation is incorrect d. the terminal is not in the operating mode to accept this command e. this command requires a ;USER command which has not been previously entered
RJECT ANY ;INPUT	A batch input command cannot be accepted at this time because the disk is unavailable (recovery, end of run, or disk error).
RJECT NO DISK SP	No disk space is available for this batch input.
RJECT ANY ;OUTPUT	A batch output command cannot be accepted at this time because the disk is unavailable (end of run or disk error).
RJECT JOB UNAVAL	No batch job with the number specified in the command is presently in the 7740.
RJECT JOB NUMBER	The batch job number specified in the ;RESUME command does not correspond to the job number of the current batch job.
RJECT DISK ERROR	The command cannot be accepted at this time because of disk error.
NOTE NOT A JOB	A ;FINISH command has been given following a batch input request, but no batch input has been received at the 7740. The batch input operation is ended by the ;FINISH command.

Index

COMMENT SHEET

IBM 7040/7044 QUIKTRAN USER'S GUIDE

FORM C28-6800-2

FROM

NAME _____ OFFICE/DEPT NO. _____

CITY/STATE _____ DATE _____

To make this manual more useful to you, we want your comments: what additional information should be included in the manual; what description or figure could be clarified; what subject requires more explanation; what presentation is particularly helpful to you; and so forth.

FOLD FOLD

CUT ALONG LINE

FOLD FOLD

How do you rate this manual: Excellent _____ Good _____ Fair _____ Poor _____

Suggestion from IBM Employees giving specific solutions intended for award considerations should be submitted through the IBM Suggestion Plan.

NO POSTAGE NECESSARY IF MAILED IN U.S.A.

FOLD ON TWO LINES, STAPLE, AND MAIL

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 33504
NEW YORK, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY
IBM CORPORATION
1271 AVENUE OF THE AMERICAS
NEW YORK, N. Y. 10020

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS
DEPARTMENT D39

FOLD FOLD

CUT ALONG LINE

STAPLE STAPLE

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