IRIS

USER REFERENCE MANUAL

educational data systems

Educational Data Systems

INTERACTIVE REAL-TIME INFORMATION SYSTEM

(IRIS)

USER

REFERENCE MANUAL

This manual is intended for all users of an IRIS system. Included are procedures for logging on and off a terminal, using the available languages and processors, entering, saving, and modifying programs, and using data files and other system facilities.

Operating procedures for BASIC and other languages on the system are included, but the reader should refer to a specific programming manual for information about how to write and debug a program. The Business BASIC Programming Manual (EDS 1016) provides detailed descriptions of all statements, functions, and commands in the BASIC language, including all extensions and special features in EDS Business BASIC.

Disclaimer: Every attempt has been made to make this manual complete, accurate, and up to date. However, there is no warranty, express or implied, as to the accuracy of the information contained herein. This revision reflects the IRIS system as released in March, 1974.

EDS 1017-4

TABLE OF CONTENTS

1. HOW TO LOG ON

To log on to the IRIS system, turn the terminal's power switch to the LINE position, and press the ESC (or ALT MODE) key. On a CRT or other terminal with no ESC key, try CRTL [or CTRL SHIFT K. The computer may print a welcome message. In any case it should ask ACCOUNT ID? Type your assigned account ID, and press the RETURN key. To maintain the secrecy of your account ID, it will not be echoed (printed) as it is typed; nevertheless, the computer is receiving everything typed and will log on the port if a valid account ID is entered. An invalid account ID will be rejected, and your account ID will again be requested. See the system manager if an account ID has not been assigned.

If all of the account's allotted CPU time or connect time has been used, the log-on will be rejected with an appropriate message. Otherwise, the port will be logged on, and the port number, account number (group and user), and the date and time will be printed. The account's status (remaining CPU time and connect time allotments and disc block usage and assignment) will also be printed. The account status may be followed by log-on messages from the system manager. The account status print out may be terminated by pressing the ESC key, but the log-on messages cannot be suppressed. The computer will then print a # symbol, indicating that the port is in control mode and ready for use.

1.1 Unusual Log On Responses

If the computer rings the bell instead of asking for your account ID, it is because it did not recognize the ESC. Try pressing the ESC key again. If the bell rings again it is probably due to incorrect parity from the terminal's keyboard. Press CTRL P (hold down the CTRL key and press the P key) to tell the computer not to check parity (a % symbol will be printed), and then press the ESC key again.

If the computer responds in some manner other than as described above, it is likely that another user left the terminal without logging off. In this case it is best to log off the terminal (see "How to Log Off" in this manual) and then log on with the proper account, since files created on another user's account cannot be protected against access by the user to whom that account is assigned. Also, such files may not be accessible later from the user's own account, or may even be on a disc cartridge that has been removed from the system.

If the computer does not respond at all when the ESC key is pressed, or if the terminal chatters when turned on line, the time-sharing system may have been shut down fōr use in stand-alone mode or for maintenance, or your terminal may not have a proper connection to the computer. If the terminal has a speed select switch, it may be set wrong. Most CRT terminals have such a switch, usually labeled BAUD RATE. Try pressing ESC, or CTRL P and then ESC, with the switch at each setting.

1.2 How to Change BAUD Rate

Once logged on, the speed of a CRT or other terminal with switchable baud rate may be changed by the following procedure:

a) The terminal must be in system command mode as indicated by a # symbol printed by the computer. If not in system command mode, press CTRL C (hold down the CTRL key and press the C key) and wait for the # symbol.

b) Type the system command

```
BAUD b
```

where b is some standard baud rate (110, 150, 300, 600, 1200, 2400, 4800, or 9600).

c) If the computer prints NO SUCH PROCESSOR it is because the system does not have a speed-selectable multiplexer, and you cannot change speed; otherwise...

d) Change the speed select switch on the terminal to the same baud rate specified in the BAUD command.

e) Press the ESC key. A # symbol should be printed to indicate a successful speed change.

```
***********************
*                     *
*      CAUTION        *
*                     *
***********************
```

Do not change the port's baud rate to a speed at which your terminal will not operate. It may be necessary for the system manager to reset the speed to a usable rate; meanwhile your account will be charged for connect time.

2. HOW TO USE BUSINESS BASIC

This section gives instructions on how to enter, run, list, modify, save and delete a BASIC language program, and to use the desk calculator mode for quick calculations. A prior knowledge of the BASIC language is assumed. If you are not already familiar with programming and program debugging techniques, you should refer to manuals and books on the BASIC language itself. For detailed information on the various statements, functions, and commands in the BASIC language, including all extensions and special features in EDS Business BASIC, refer to the Business BASIC Programming Manual (EDS 1016).

2.1 How to Enter a BASIC Program

To enter a BASIC program you must first log on to the system (see Section 1 of this manual) and then select BASIC as the language to be used. To select any language or to perform any other system command, your terminal must be in control mode as indicated by a # symbol printed by the computer. If not already in control mode, press CTRL C (hold down the key labeled CTRL and press the C key) and wait for the # to be printed. In some cases it may be necessary to press CTRL C a second time. Now type the command

```
BASIC
```

and press the RETURN key. This selects Business BASIC as your programming language. It will not be necessary to do so again unless you leave BASIC with a CTRL C to save or delete a program. If it is desired to clear out a program and begin entering a new program, type the command

```
NEW
```

and press the RETURN key. This will completely eliminate the current program so that the new one can be entered without conflict.

To enter your BASIC program, type each statement, starting with a line number and ending with the RETURN key. If the program statements are not entered in order of increasing line numbers, they will automatically be sorted into that order as they are entered. If an error is detected in one of your statements, an error message such as

```
ERROR #6
```

will be printed. To determine the type of error, refer to the BASIC Error Numbers in Appendix 1 of this manual, or type the word

```
HELP
```

and press the RETURN key to request a message to be printed describing the type of error.

2.2 How to RUN a BASIC Program

After entering your program, type the command

```
RUN
```

and press the RETURN key. Your program will be executed starting with the smallest line number. All variables are initially assumed to be zero, all user functions are assumed undefined, and all arrays and strings assumed dimensionless until a statement is encountered to provide the necessary information. A run may be aborted at any time by pressing the ESC key.

If an error is detected in your program, an error message such as

```
ERROR #25 AT 320
```

will be printed. This example indicates an error in statement 320. To determine the type of error, refer to the BASIC Error Numbers in Appendix 1, or type the word HELP and press RETURN to get a type-out of the meaning of the latest error. A command of the form

```
HELP 30
```

may be given to get a message on an earlier error (in this case, error #30).

Some types of errors are serious enough that the program is stopped when the error is detected, and the word READY is printed following the error message. Other types of errors allow the program to continue with the next statement in sequence, thus allowing several errors to be detected in a single run.

A run may also be started at any desired statement in the program by giving a command such as

```
450 RUN
```

which will start executing the program at the line number specified. In this case, the variables, user-defined functions, arrays, and strings remain as they were at the time the last run was stopped. This form of the RUN command is illegal after entering a new program until at least one run has been started with RUN (without a line number) to initialize all variables, etc. It is also illegal after modifying a program if it was stopped within a subroutine or a FOR-NEXT loop.

2.3 How to LIST a Program or Punch a Program Tape

To obtain a listing of your current program as it stands, give the command

```
LIST
```

and press RETURN. Your entire program will be listed in standard form. Statements within a FOR-NEXT loop are indented for easy identification of loops and nesting. The listing may also be started at any point in the program by giving a command such as

```
180 LIST
```

to start listing, in this example, at line 180 of the program. If there is no statement with the given line number then the listing will start with the next higher numbered line. A segment of a program may be listed by giving a command such as

```
270 LIST 600
```

which, in this example, will cause lines 270 through 600, inclusive, to be listed. Also, the ESC key may be pressed at any time to terminate listing. A BASIC program may also be listed on the line printer by giving the command

```
DUMP $LPT
```

Line numbers may be similarly used with the TAPE and DUMP commands (see below) for output of a selected part of the program. If a terminating line number is used with the DUMP command, it must be given <u>after</u> the Filename; e.g.

```
300 DUMP $LPT 1750
```

To punch a tape of your program, type the command

```
TAPE
```

but do not press the RETURN key until after pressing the ON button on your tape punch. About six inches of blank tape will be punched, and the complete program will then be listed and punched, followed by blank tape for trailer.

If there is a high speed paper tape punch on the system, it may be used to punch a list tape of a program. To do so, type the command

```
DUMP $PTP
```

and press the RETURN key. The tape, which will be punched with leader and trailer, may be loaded later as described in Section 2.10.

2.4 How to Modify or RENUMBER a Program

Any statement in a program may be modified by typing the statement in the new form desired, using the same line number. Any statement entered will replace a previously entered statement with the same line number.

To insert a statement between two statements in the program, use any line number that is between the line numbers of those statements.

To delete a statement from a program, type only the line number, and press the RETURN key.

A program may be renumbered by giving the command RENUMBER which will cause the line numbers of the program to be changed to 10, 20, 30, . . ., and will cause all references in the program (such as in GOTO and GOSUB statements) to be adjusted so that they still point to the same statements as before. A more general form of the renumber command is

```
x RENUMBER y
```

where x and y are decimal numbers. This will cause the program to be renumbered using line numbers starting with x and increasing in steps equal to y. If y is omitted, a value of ten is assumed. If x is omitted, a value equal to y is assumed. If a statement references a line number that does not exist then it will be changed to reference the next higher line number or to zero if there is no higher line number, and an error message will be printed giving the old line number of the statement where the error occurred.

2.5 How to SAVE a Program

Once a program has been entered it may be desirable to save a copy of it in a passive file. Such a copy may be recalled at a later time to be used and/or modified (see "How to Use a Saved Program"). Other users may be allowed to use, copy, and modify the saved program, or it may be selectively protected against access by others. Also, if desired, other users may be charged for using your program.

To save a program, your terminal must be in control mode as indicated by a # printed by the computer. If it is not in control mode, press CTRL C and wait for the # symbol. Now enter the command

```
SAVE Filename
```

where Filename is any legal file identifier (see Filename in the Glossary).

In most cases, the computer will print SAVED!! indicating that your program has been saved under the Filename given. However, several types of errors are possible as follows:

EMPTY FILE	You don't have a program to save.
COPY PROTECTED	You are using someone else's program and he has prohibited copying it.
ILLEGAL FILENAME	The identifier given is not a valid Filename as defined in the Glossary.
NO "!" OR WRONG ACCOUNT	The Filename given is already in use to identify a file of the same type, on the same Logical Unit, and on the same user account (the system will not allow the existing file to be inadvertantly replaced, but it may be intentionally replaced as described in Section 2.7) or the Filename is in use to identify another user's file on the same Logical Unit. Another user's file may not be replaced by the SAVE command.
TYPES DON'T MATCH	The Filename given is already in use on the same Logical Unit to identify a different type of file on your account. Only files of the same type may be replaced by the SAVE command.
FILE BEING CHANGED	The Filename given is already in use to identify a file which is currently being built, replaced, or modified by another user. Such a file can not be replaced.
LOGICAL UNIT NEEDS n MORE BLOCKS	The Logical Unit is full or nearly full and needs n more blocks to save your program. The system manager should be notified of this condition. See note below.
ACCOUNT NEEDS n MORE BLOCKS	Your account does not have enough blocks allotted to save this program as well as others you have saved. You may be able to save this one by first deleting others you have previously saved. See note below.

Note: It is necessary to have enough blocks available to build the new copy of your program even if you are replacing an old file by the same Filename. If your new version cannot be saved due to a shortage of disc blocks, you may be able to overcome this problem by first deleting the old file with a KILL command.

If any error message is printed, then your program has not been saved. However, SAVE does not affect your active file, so another attempt to save your program may be made. When a program is successfully saved, your account is charged for the number of blocks required to save it.

The simple form of the SAVE command given above allows only a user on your account or a higher privilege user to run, modify, or copy your program, but no charge will be made if such a user does access your program. The general form of the SAVE command

```
SAVE      <pp>      $ddd.cc      lu/Filename
```

allows you to grant access to your program by other users at your privilege level and lower, to charge all others for its use, and/or to specify a Logical Unit other than the one assigned to your account. In this case pp represents a two digit number specifying the desired protection. The first digit gives protection against users at lower privilege levels, and the second digit gives protection against users on other accounts at your own privilege level. It is not possible to protect against higher privilege users. A file may be protected against other users of your own account only by use of a password (see Filename in the glossary). Each digit indicates protection as follows:

p	protection
0	None
1	Copy protect. Prohibits others from listing your program, punching a tape of it, or saving it under a different Filename or on a different Logical Unit.
2	Write protect. Prohibits others from deleting the saved copy of your program or changing its attributes.
4	Read protect. Prohibits others from using your program.

The types of protection may be combined by adding the values given for each desired type (see example below). If the program was previously saved then the current protection will remain in effect unless it is respecified at this time.

The dollar sign indicates that the amount ddd.cc (dollars and cents) is to be charged to the account of any other user who gains access to your program. The cost given will be truncated to the nearest ten cents. If the program was previously saved, then the cost specified at that time will remain in effect unless respecified. Both the protection and the cost are optional in the SAVE command. Additional error messages are possible if protection or cost is specified:

INVALID PROTECTION	The protection given does not adhere to the form shown in the example.
ILLEGAL COST	Characters other than digits and one period were given following the dollar sign, or a cost greater than $999.90 was specified.

The following example shows a typical SAVE command:

```
SAVE   <72>    $2.58    4/CASHFLOW3
```

This command saves your program under the Filename CASHFLOW3 on Logical Unit number four and protects it against any access by lower privilege users. Other users at your own privilege level will be allowed to use it, list it, or save their own copy of it, but only you or other users on your account will be able to delete it or to modify and resave it under the same name. Any user accessing it (other than on your own account) will be charged $2.50 for the use of your program, and a record of charges made will be accumulated in your program's file header so that the charges can be forwarded to you.

A program can also be saved in source form in a data file formatted as a single 75-character string by entering the BASIC command

```
DUMP Filename
```

where Filename is any legal file identifier.

2.6 How to Use a Saved Program

To issue a command to use a saved program, your terminal must ʟ in the control mode as indicated by a # symbol printed by the computer. If not already in control mode, press CTRL C first. A BASIC program may then be accessed by entering the command

```
BASIC Filename
```

where Filename is the identifier given to the program when it was saved, including the Logical Unit number if different from that assigned to your account. The program file will be copied into your active file, and may be listed, modified, or run as desired. However, if Filename is another user's program, it may be read protected or copy protected, and there may be a charge for the use of the program.

If the program is read protected, the command above will be rejected with an appropriate error message. If it is copy protected, you will not be able to list, modify, or resave the program. Section 5.1 of this manual tells how to get a library listing of available programs and the charges, if any, for their use.

A saved program may also be run directly from control mode by giving the command

```
RUN Filename
```

or simply

```
Filename
```

which will bring a copy of the program Filename into your active file (if it is not read protected) and immediately begin running the program.

If the program was saved as a data file, it may be accessed by giving the command

```
LOAD Filename
```

where Filename is the name assigned when the program was saved by use of a DUMP command.

2.7 How to Modify a Saved Program

A saved program may be modified by obtaining a copy of the program (see Section 2.6), modifying the program in your active file (see Section 2.4), and then resaving it. It may be resaved under the same Filename by giving the simple command

```
SAVE
```

and the system will print the Filename. In this case, the modified version will replace the originally saved version.

If a new program is written and it is desired to replace an old program by the same name, enter the command in the form

```
SAVE Filename!
```

The exclamation mark following the Filename indicates that the user is aware that he may already have a file by the same name and he wishes it to be replaced. Only a file on the same account may be replaced, however, even with this command. The protection and cost may also be specified as discussed in Section 2.5. The modified version may also be saved under a new Filename as explained in Section 2.5. However, it will not be possible to modify or resave another user's program if it is copy protected or to resave it under the same Filename on the same Logical Unit.

2.8 How to KILL a Saved Program

A saved program may be deleted by giving the system command

```
KILL Filename
```

where Filename is the identifier of the saved program. Your terminal must be in control mode as indicated by a # symbol printed by the computer. It may be necessary to first press CTRL C to return to control mode.

If Filename is your own program, it will be deleted, the disc blocks will be returned to the general pool, and your account will be credited for the reduced disc block usage. However, if Filename is another user's program, it will be deleted only if it is not write protected. When another user's program is deleted, that user's account is credited for the reduced disc block usage.

Several files may be deleted with a single command. For example,

```
KILL  TIMER, D$36, 4/MAX, BJ22, 1/JJ, $PTP
```

will attempt to delete all six files listed, and the response ALL DELETED will be printed if all are successfully deleted. However, in this example, D$36 is illegal as a Filename. Also, let us assume that there are no such files as TIMER or JJ, and that BJ22 is write protected. In this case, the messages

```
? NOT  FOUND:  1  5
? ILLEGAL NAME:  2
? WRITE PROTECTED:  4
? SYSTEM FILE:  6
```

will be printed. The numbers refer to the sequence in which the Filenames were given; i.e., 1 refers to TIMER, 2 to D$36, etc. In this example, only the third file (MAX on Logical Unit number four) has been deleted. If a Logical Unit number is not specified then the user's assigned Logical Unit is assumed. It may be that TIMER or JJ exist on a different Logical Unit.

2.9. How to Use Calculator Mode

Most BASIC statements may be executed immediately upon entry by typing the statement without a line number. The exceptions are the data file statements and any transfer of control statement such as GOTO, GOSUB, RETURN, NEXT, or ON.

The most useful statement in calculator mode is PRINT. For this reason, the semi-colon is accepted as an abbreviation for the word PRINT at the beginning of a statement. To evaluate any expression, type the word PRINT (or a semi-colon) followed by the expression, then press RETURN. The computer will evaluate the expression and print an equal sign followed by the calculated value. Values may also be assigned to variables and these values used in other calculations. For example, try statements such as

```
PRINT  2/3
LET  A=5
PRINT  A, A↑2, 3*A
;SQR(A+1)
;LOG(A/2+3↑4*1.42)-INT(A/3.2)
B=A+7*3
```

Note in the last example that the word LET is not required in an assignment statement. It is also possible to dimension arrays, enter, print, and manipulate matrices, etc., without writing a program.

A READ statement may be executed from the keyboard if one or more DATA statements have been entered with line numbers. However, a RESTOR statement must first be executed to set the pointer to the first data element.

2.10 How to Load a BASIC Program from Tape

A BASIC program may be available on paper tape, either punched as a list tape (see Section 2.3) or punched off-line (Teletype on LOCAL). If punched off-line there must be a LINE FEED after (not before) each RETURN code on the tape or the first character of each line may be lost. To load such a program tape, follow these steps:

1. Select BASIC language (see Section 2.1).

2. Type NEW and press the RETURN key to delete any old program (unless it is desired to merge the tape with the existing program).

3. Load the tape into the Teletype reader as follows:

    a. Set the tape reader switch to FREE.

    b. Open the plastic tape gate by pressing the knob at the right side of the gate to the right.

    c. Load the tape with the leader toward the front and the program tape hanging behind the reader.

    d. Position the tape feed holes (small holes) to the left side so that they mesh with the feed wheel, and close the tape gate.

4. Momentarily press the tape reader switch to START, then release it.

The tape will read in and stop at the end of each statement while the computer checks for errors. If an error is detected, an error message is printed, and then the remainder of the tape will be read. After the entire tape has been loaded, the user should correct and type in any statements in which an error was detected.

If there is a high speed paper tape reader in the system, it may be used to load a tape as follows:

1. Select BASIC language (see Section 2.1).

2. Type NEW and press the RETURN key (unless it is desired to merge the tape with the existing program).

3. Load the tape in the high speed reader.

4. Type the command LOAD $PTR and press the RETURN key.

The tape will be loaded, and an error message will be printed for each line in which an error is detected.

2.11 How to Load a BASIC Program From Cards

If there is a card reader in the system, it may be used to load a BASIC program that has been punched on cards with a standard key-punch or otherwise using standard Hollerith codes (see Appendix 5). On some keypunches it is necessary to use multiple punching to obtain some of these codes.

If the card reader is an optical mark sense type then the cards may also be hand marked by use of an EDS BASIC Card Programmer (see Appendix 6). EDS BASIC Cards (form EDS 202) are recommended for this purpose. The first card must have a "BASIC Cards" code (see below) in column one for proper translation of the card codes.

Holletith cards and "BASIC Card Code" cards may be intermixed as long as the first card of each group has the proper "Hollerith Cards" or "BASIC Cards" code in column one. Each card may have one of these codes in column one if desired.

Once the card deck has been prepared, the BASIC program may be loaded as follows:

1. Select BASIC language (see Section 2.1).
2. Type NEW and press the RETURN key (unless it is desired to merge the cards with the existing program).
3. Turn on the card reader and put the deck in the hopper. Be sure the deck is properly oriented. If the reader has a START or READY button it must be pressed to make the reader accessable to the system.
4. Type the command LOAD $CRD and press the RETURN key. The cards will be loaded, and an error message will be printed for each line in which an error is detected.

Special Card Codes - In addition to standard Hollerith codes and the codes on the Educational Data Systems' BASIC Card Programmer, the following card codes are recognized by the system:

RETURN	9-8	Must follow any command or statement.
CTRL E	9-8-5	
Hollerith Cards	9-8-4	These special codes will be recognized only in column one of a card.
BASIC Cards	9-8-3	

Hollerith cards are assumed unless a "BASIC Cards" code is encountered. Any undefined code will be entered as a reverse slash character.

3. HOW TO LOG OFF

It is necessary to log off before leaving the terminal to avoid being charged for additional connect time and to prevent others from using your account. To log off, the terminal must first be in the control mode as indicated by a # symbol printed by the computer. If not already in control mode, press CTRL C (hold down the CTRL key and press the C key) and wait for the # to be printed. Then type the command

```
BYE
```

and press the RETURN key. The terminal will be logged off, and the account's status will be printed. The account status consists of:

1. Account number (group and user),

2. Net file use charges (accrued charges for use of other user's programs or data files, less charges for use of this account's files by other users*),

3. CPU time and connect time used while logged on,

4. CPU time and connect time allotments remaining (or amount overtime), and

5. Disc block usage (number of blocks in use and the total number allotted) and assigned Logical Unit number.

An overtime indication for CPU or connect time indicates that more time has been used than had been allotted to the account, and the user will not be allowed to log on to this account again. The system manager should be notified of this condition, and he may allot more time to the account.

Once the terminal is logged off, it enters an idle state and will respond only to the ESC key to allow a user to log on.

*Note: unless a file is deleted, the file's account is not credited for charges to other users until an accounting program is run by the system manager. Therefore, the net charges due may actually be less (or more negative) than the amount shown here. A negative amount indicates a net income.

4. GENERAL DESCRIPTION OF SYSTEM USE

All system commands are executed by a processor, and each interactive port always has one processor selected for current use on that port. When in control mode, as indicated by a # symbol printed by the computer, the selected processor is SCOPE (System Command Processor). Any line entered at this time is analyzed by SCOPE by looking for a processor with the Filename given first in the line. The processor's Filename, therefore, becomes the system command itself; for this reason, most processors have a verb for a Filename. In the case of a language processor, SCOPE will automatically select the proper processor if the first Filename encountered is a program file.

4.1 Modes of Operation

A user's terminal on the IRIS system may be in one of three basic states as follows:

1) Idle state. Before logging on, and after logging off, the terminal is in an idle state. Characters typed by the user will not be echoed. The computer will respond only to the ESC key, whereupon it will request that the user supply his account ID to log on.

2) Control mode. After logging on, the user's terminal is in control mode as indicated by a # symbol printed by the computer. Only in this mode will system commands such as BASIC, SAVE, KILL, LIBR, or BYE be accepted. The LIBR*S command will give a listing of all available system commands. The user may return to control mode at any time by pressing CTRL C (hold down the CTRL key and press the C key). In most cases, a program Filename will be accepted as a system command, and the appropriate processor will be automatically selected by the system; i.e., RUN for a BASIC program, TUTOR for an instructional program, etc.

3) Processor mode. Each system command and user language is provided by a machine code program called a processor. Some processors, such as SAVE and KILL, perform their entire task and return the user to control mode. Others, such as BASIC, process each line entered by the user and then perform a carriage return to signal the user that input is enabled for the next type-in. The EXECUTE processor prints an up-arrow symbol and EDIT prints an asterisk as prompt characters for their own control modes (as opposed to the system control mode).

4.2 Error Correction

Characters typed by the user are not examined by the processor until the RETURN key is pressed. Therefore, typing errors may be corrected before being detected by the processor, thereby saving time and preventing error messages. Two control keys are recognized by the system for error correction. They are:

CTRL H	The CTRL H acts as a backspace, deleting the previous character typed. This backspace function may be used repeatedly to delete several characters. The character being deleted is printed when the CTRL H key is pressed. On most systems, the CTRL A also acts as a backspace.
CTRL X	Use of the CTRL X cancels the entire line just typed in. A backslash symbol is printed to indicate that the line has been deleted, and a carriage return is performed.

In some cases a typed character may not be accepted by the computer. Such cases are:

Input not enabled
Transmission error
Input buffer full

If input is not enabled the character (other than ESC) will simply be ignored, except that the last character typed will be retained until input is enabled, at which time it will be echoed and entered into the I/O buffer. To terminate an operation in progress and return to the processor's input mode, the ESC key may be pressed at any time. In the case of a transmission error or buffer full the terminal's bell will ring. Try typing the same character again. In the case of a transmission error it may be accepted the second time; if not, press CTRL P to suppress parity checking (a % symbol will be printed) and try again. If the input buffer is full, a CTRL H, CTRL X, or ESC must be used before further input will be accepted.

4.3 About Your Account

Each user is assigned to an account by the system manager. Several users may be assigned to the same account and may use the system simultaneously from different terminals. Each account contains the following information:

Account ID
Assigned Logical Unit number
Privilege level
Account number (group and user)
Allotted connect time (minutes)
Allotted CPU time (seconds)
Allotted disc blocks
Number of disc blocks now in use
Peak number of disc blocks used
Net file use charges

The assigned account is located by the Account ID entered when the user logs on, whereupon the assigned Logical Unit number, privilege level, and account number are brought into core to identify the user.

Any Filenames supplied by the user are assumed to be on his assigned Logical Unit unless a Logical Unit number is given along with the Filename. Exception: system command processors are assumed to be on the system disc (Logical Unit zero). See Glossary for a discussion of Logical Units.

The user's privilege level may be zero, one, or two, indicating "general", "privileged", or "manager" privileges, respectively. Only the system manager account can access the ACCOUNTS file, but all level two accounts are accorded the other privileges of a manager.

The account number is used to identify files created by a user. The division of the account number into group and user numbers is for the convenience of the system manager; there is no special significance to either the group number or the user number within the system except as described for the LIBR @g command (see Section 5.1).

Each user account may be allotted up to 32767 minutes of connect time and up to 32767 seconds of CPU time. Either or both times may also be specified as "no limit" by the system manager. A user's account is charged for connect time from the time he logs on until the time he logs off regardless of how much the system is used during that time. The account is charged for CPU time only when the computer is actually performing a task for the user. Either or both allotted times may go negative while a user is logged on; he will be informed of this overtime condition when he logs off, and no one will be allowed to log on to that account until more time is allotted by the system manager.

Each account is allotted some number of disc blocks for saving programs and building data files. The disc block allotment is a constant set by the system manager. The number of disc blocks in use will initially be zero and is updated by the system each time the user builds, deletes, or adds blocks to any file on his account, but the user will never be allowed to exceed his allotted number of blocks. The peak disc block usage is updated each time a user logs off.

Each time a user opens a file belonging to another account, the cost of that file is added to net charges in the user's own account. If a file is deleted, then any income accrued by that file is deducted from the owner's account. Each account is also credited for income earned by its files when the accounting program is run by the system manager.

Each user must have an account on Logical Unit zero to be allowed to log on to the system, but he may or may not be allotted disc space on unit zero. Each user must also have an account on each Logical Unit on which he is to be allowed to build files. Each such account entry other than on unit zero is used only for disc block allottment and usage on that Logical Unit. Typically, each account would be allotted space on only one Logical Unit, and the account would be assigned to that unit so that it is not necessary to supply a Logical Unit number each time a Filename is given. Access to existing files on any Logical Unit is determined only by the protection of the files being accessed regardless of whether or not the user has an account on that Logical Unit.

5. MORE SYSTEM COMMANDS

This section describes various utility commands which may be used to obtain a catalog listing of programs available to the user, to change a file's attributes or to cause its attributes to be printed, to send a message to a user on another port, etc.

5.1 How to Use LIBRary

To obtain a library listing, your terminal must first be in control mode as indicated by a # symbol printed by the computer. If not already in control mode, press CTRL C and wait for the # symbol. Now type the command

```
LIBR
```

and press RETURN. A complete listing of all files belonging to the user's account and on the user's assigned Logical Unit will be printed. The listing is preceded by a heading identifying each column as follows:

*	File type or language. One letter will be printed in this column to identify the file type. See page 5-3 for a complete list of file type letters.
NAME	The file's complete Filename will be printed, except in the case of a private file on the user's own account or any other account at the same or lower privilege level, in which case the password portion is not printed. A colon is printed to represent a CTRL E offsetting a password.
PRIV	The privilege level of the file. This is the same as that of the user who created the file.
COST	If the file's owner is charging for access to the file, this column indicates the amount charged for each use of the file.
SIZE	This is the number of disc blocks used to store the file.
PR	This is the protection status of the file as specified when the file was saved. See "How to SAVE a Program" in this manual for an explanation of these two digits.

After the complete listing, the number of disc blocks available for creating or expanding files on the selected Logical Unit will be printed.

Several more specific forms of the LIBR command are available for use when it is desired to list only a certain subset of accessible files in the library:

LIBR @	Lists all accessible files on all accounts.
LIBR @g	Lists only those files belonging to any account in group g.
LIBR @g, u	Lists only those files belonging to the account group g, user u.
LIBR *B	Lists only files of type specified (BASIC programs in this example). See the table of type letters on the next page. A maximum of five types may be specified to be listed.
LIBR NAME	Lists only files whose Filename begins with the characters given ("NAME" in this example). The command shown would list such Filenames as NAME, NAME3, NAMEXYZ, NAME<u>E</u>PASS, etc.
LIBR <u>E</u>KEY	Lists only files whose Filenames include the complete password given ("KEY" in this example). If there is more than one user to an account, it is suggested that each user select a single password for all his files, so that this library listing will list only his files.
LIBR ↑	Will cause all selected entries to be alphabetized before being listed. Without the up-arrow, the files are listed in order of occurrence in the INDEX. An alphabetized listing may be somewhat slower than an order-of-occurrence listing.
LIBR >h	Will cause only files which have not been accessed for more than h hours to be listed.

The command forms listed above will list only such files on the user's assigned Logical Unit. To obtain a catalog of files on some other unit, enter the Logical Unit number and a slash (/) ahead of the selection characters. For example:

```
LIBR 3/
```

will list all files on the user's account on Logical Unit three.

Commands may be combined in any manner except that if a Logical Unit is specified it must be given first. For example:

LIBR 2/TRE *B *T *D @6 <u>E</u>JKL<u>E</u> ↑ >50

will give an alphabetical listing of each accessible file on Logical Unit number two if and only if:

a) the Filename begins with the letters "TRE" and includes the password "JKL", and

b) it is either a BASIC program, a text file, or a formatted data file, and

c) it belongs to any user account in group six, and

d) it has been more than fifty hours since the file was last accessed.

The following table lists all file types and their designating letters as currently assigned:

No.	Letter	File Type
0	P	Permanent system file
1	S	System processor or file
2	B	BASIC processor or program
3	A	Stand-Alone processor or program
4	X	EXECUTE processor or program
5	G	GPM program
6	M	MUMPS processor or program
7	W	COURSE WRITER processor or program
10		
11		
12		
13		
14		
15		
16		
17		
20	Q	Stand-alone compiler
21	J	Stand-alone relocating assembler
22	L	Stand-alone relocating loader
23	R	Relocatable binary object tape image
24	I	Indexed relocatable binary library
25		
26		
27	Z	Temporary file
30	T	Text file (ASCII)
31	F	Formatted data file
32	C	Contiguous data file
33		
34		
35		
36	$	Peripheral device driver

5.2 How to CHANGE File Characteristics

Certain characteristics of a file may be changed by the user who created the file. Other users who have been granted full access to a file (i.e., users against whom the file is neither read protected nor write protected) may also make such changes. Although intended primarily for data files, other files (such as BASIC programs) may also be changed. The characteristics which may be changed are (1) the Filename, (2) the cost, and (3) the protection.

To change a file's characteristics, the terminal must be in control mode as indicated by a # symbol printed by the computer. Enter the system command

```
CHANGE Filename
```

where Filename is the name of the file to be changed, and press RETURN. If for any reason the file cannot be changed, an appropriate error message will be printed; otherwise, instructions will be printed, and the user will be asked to enter the new name. Type a new Filename if desired and press RETURN, or press only the RETURN key to leave the name unchanged.

The current cost (charge to others for access to the file) will be printed, and the user will be asked to enter a new cost. Type in the new cost, or press RETURN if no change is desired.

The current protection digits will be printed next. These digits are explained under "How to SAVE a Program". Type a two-digit number to replace the current protection, or press RETURN to leave the protection unchanged.

A user on a privilege 2 account can also change the processor control bits. See "More on CHANGE and SCOPE" in the IRIS Manager Reference Manual for more information.

5.3 How to QUERY File Characteristics

The characteristics of any file may be determined by use of the QUERY command. The terminal must first be in control mode as indicated by a # symbol printed by the computer. Then enter the system command

```
QUERY  Filename
```

where Filename is the name of the file to be queried, and press RETURN. If the file is not on your assigned Logical Unit then the name must be entered in the form lu/Filename. The command will be rejected if the file is not found or if it is read or copy protected; otherwise, the file's type, privilege level, protection, size, age, cost, and earnings will be printed. Additional information is given for certain types of files; e.g., the record length and format of a data file or the starting address of a machine code (stand-alone or executable) file. For the system manager, the file's header block address and L and I control bits will also be printed.

Several files may be queried at the same time by entering the command in the form

```
QUERY  file1, file2, file3, . . .
```

in which case all pertinent information about each file will be given in succession.

Before giving any information about a file, QUERY checks for any irregularities. If the file's header does not agree with the INDEX entry, or if the number of disc addresses in the header does not agree with the size, an appropriate error message will be printed. The system manager should be notified at once if such an irregularity occurs. Do not use or delete the file until the system manager has checked it out.

5.4 How to QUERY Account Status

An alternative form of the QUERY command, namely

```
QUERY @          or          QUERY @n/
```

will print the current total number of disc blocks available (same as given at the end of a LIBR listing), the number of disc blocks allotted to your account, the current and peak disc usage of your account, the net charges for use of other user's files, and your account number and privilege level. The first form gives this information with regard to your assigned Logical Unit, while the second form gives the information for any active Logical Unit specified by n.

5.5 How to MAIL a Message

A one line message may be mailed from any port to any other port. To send a message, your terminal must be in system command mode as indicated by a # symbol printed by the computer. Then type the command

```
MAIL   p   any message
```

where p is a number from zero through the highest numbered port to indicate the destination of the message. For example, a user on port seven might type

```
MAIL 0  MANAGER -- PLEASE INSTALL UNIT 3
```

which would cause

```
PORT 7:  MANAGER -- PLEASE INSTALL UNIT 3
```

to be printed on port zero. Note that the first part of the printed message identifies the sender's port so that a reply may be sent. For example, the user at port zero might reply

```
MAIL 7 SORRY, THE SYSTEM MANAGER IS NOT HERE.
```

The message will not be printed at the destination port until that terminal is in an input mode and at the left margin (nothing yet typed in by the user). Thus, the message will not be printed during another print out or in the middle of an input. Although the sender presses the RETURN key at the end of his message, his terminal's carriage will not be returned until the message begins printing at the destination port. The ESC key may be pressed to abort a message that is never accepted by the destination port.

There is one exception to the "input mode, left margin" rule: two ports may simultaneously send messages to each other. Both will be printed after both users have pressed the RETURN key.

5.6 How to INSTALL a Logical Unit

Only Logical Unit #0 (the system disc) is accessible after an IPL. Other units are inaccessible until they have been "installed". Any user may install a Logical Unit, providing it and the Physical Unit are not locked against the user, by typing the system command

```
INSTALL d.p
```

where d is the number of the disc or magnetic tape drive where the unit is to be installed, and p is the partition number. For example, to install a cartridge of a dual drive which has been designated as drive number 2, the command would be

```
INSTALL 2.1
```

since partition zero usually would be the fixed disc. Most drives are partitioned into two or more Logical Units. For instance, each disc of a dual disc drive must be treated as a separate Logical Unit, and a high stack disc which is too large to be treated as a single unit is partitioned into several Logical Units. Also, the system manager may partition a disc or cartridge into multiple units for assignment to various users or groups. The drive numbers are assigned to physical drives in the CONFIG file. Only the system manager, however, may install a new disc cartridge which has never been formatted as an IRIS Logical Unit; also, it is absolutely essential that the disc diagnostic tests be run on such a cartridge first.

INSTALL will read from the Physical Unit at location d.p to determine whether it is already formatted; if so, INSTALL will print its Logical Unit number and ask whether it should be installed. A positive response by the user will cause the unit to be put on line. If the user wishes to change the Logical Unit number, he may respond negatively, and a new unit number will be requested; it should be noted, however, that this command will be rejected if the Logical Unit is locked against change by the user.

If the Physical Unit is not formatted, and the user is the system manager, he will be asked for a Logical Unit number, and the unit will be formatted and installed using the number entered. However, the number will be rejected if there is already another Logical Unit installed with the same number.

The INSTALL command will also be rejected if there is already a unit installed at location d.p, if there is already another unit installed with the same Logical Unit number, or if the Physical Unit is protected against installation by the user.

5.7 How to REMOVE a Logical Unit

It is necessary to inform the system of the intent to remove a Logical Unit before the unit is actually shut down for removal. The terminal must be in system command mode as indicated by a # symbol printed by the computer. Then enter any of the following commands:

```
1)    REMOVE n/

2)    REMOVE DRIVE d

3)    REMOVE DRIVE d.p
```

where n is a Logical Unit number, d is a drive number, and p is a drive partition number. Note that the unit to be removed may be identified either by its Logical Unit number or by its physical location. Command 2 above will attempt to remove all Logical Units on the specified drive. It is necessary to remove all Logical Units on a given drive before shutting it down to change any cartridge on the drive (see Section 5.8). Possible errors include:

NO SUCH LOGICAL UNIT. The specified Logical Unit is not installed on the system.

LOGICAL UNIT IS PROTECTED. The user's account is not privileged to remove the specified unit. Only certain users as specified by the system manager are allowed to remove a Logical Unit.

FILE OPEN ON PORT #x. The user on the indicated port has one or more files open on the specified unit. There must be no files open on a unit when it is removed.

After checking for the above errors, the entry in the Logical Unit Table is cleared, thus making the unit inaccessable. REMOVE then indicates that the Physical Unit may be removed unless there is another partition on the Physical Unit that is still installed. The unit may be installed again at a later time without loss of data.

```
********************
*                  *
*     CAUTION      *
*                  *
********************
```

To remove or change the system disc (Logical Unit zero) it is necessary to shut down the system completely, change the cartridge, and do an IPL (see "Startup and Shutdown Procedure" in the IRIS Manger Reference Manual). An attempt to restart without an IPL may result in the loss of all data on the cartridge.

5.8 How to Change a Disc Cartridge

Before changing a disc cartridge it is not only necessary to REMOVE all Logical Units on the cartridge to be removed, but all other Logical Units on the same physical drive must be removed as well. This includes, for example, the fixed disc on a Diablo 44, an Iomec 2002 or 2012, or a Caelus CD24, 303, or 306. Therefore, if Logical Unit zero is on the fixed disc, it is necessary to shut down the system, change the cartridge, and do an IPL to resume system operation (see "Startup Procedures" in the IRIS Manager Reference Manual). Physically independent drives, such as with a Diablo 33, do not require removing the fixed disc or shuting down the system in order to change the removable cartridge. After all Logical Units on the same physical drive have been removed, stop the drive, change the cartridge, and start the drive. When the drive is ready, the Logical Units may be INSTALLed.

5.9 How to Use COPY

COPY is a general purpose command for moving data of any type from any source to any destination. Also, data from several sources may be merged into or appended to a single destination. Each source and each destination may be any disc file or any peripheral device, the only restriction being that all sources must supply a data form that is compatible with the destination. The general form of the copy command is

```
COPY Dest = Source1,Source2,...
```

where "Dest" is the Filename under which the destination file is to be built and "Source1", "Source2",... are Filenames of source files. Each Filename may be in the form lu/Filename or may specify a peripheral device such as $LPT. If the destination file is to replace an existing file then "Dest" must be given in the form Filename!. To specify the file type of the destination file, give "Dest" in the form

```
Filename*t
```

where t is a file type letter as used by LIBR. If the file type is not specified in this manner, then the type of the first source file will be assumed. If the destination is a peripheral device, an attempt will be made to write to the device; the peripheral driver will generate an "item types don't match" error if it cannot accept the type of data offered.

The above command form will build a destination file and load the contents of the specified source files into it. However, if the file being copied is a formatted data file, the command must be given in the form

```
COPY Dest ← Source
```

where the "←" operator signifies that the destination is to be a file image of the source rather than copying only the contents. Thus, the format map or other attributes will also be copied to the destination.

In the case of multiple sources, the data from sources 2 through n are appended to the image of source 1, merging the sources as appropriate for the file type. Multiple source files are illegal if copying a BASIC program or if the sources are formatted differently. Data can also be appended to an existing file by giving the command in the form

```
COPY Filename + Source1, Source2,....
```

which causes the destination file to be opened instead of building a new file, and the sources are appended as if it were the first source.

Multiple sources are allowed only in cases where it makes sense to concatenate or merge the data; specifically, the following groupings are allowed:

1) ASCII sources (text files and byte oriented peripherals such as paper tape, cassette tape, or cards).

2) Binary sources (machine code files, including processors, binary format paper tape, binary image magnetic tape, etc.).

Other types of files may be copied on a one-for-one basis. This is useful for copying a data file or a BASIC program from one Logical Unit to another, or to copy it to or from a binary oriented peripheral, e.g. to dump an image of a data file as a binary paper tape.

In some cases, operator intervention is required before a source can be used. An up arrow symbol (↑) ahead of the source name will cause the message LOAD DEVICE THEN PRESS RETURN to be printed. One example of the use of this feature is in loading several paper tapes into the same file by giving the command

```
COPY XYZS1 = $PTR, ↑$PTR, ↑$PTR
```

which will merge three source tapes into one text file and will wait for the operator to reload the paper tape reader with the second and third tapes.

The COPY command may also be used to compare or verify data from one file or device against the data from one or more other sources by typing a question mark ahead of the destination Filename; i.e.

```
COPY  ?  Dest = Source1,Source2, . . .
```

will compare the contents of the "destination" against the source or sources specified, and any discrepency will cause a print out of both the destination and the source where the mismatch occurs. The command form

```
COPY  ?  Dest ← Source1, Source2,. . .
```

is the same except that the format map and other attributes are also compared.

Following are typical examples of the use of the COPY command:

1) `COPY  CAT1*M = $PTR`
   This will read paper tape and build a file named CAT1 to store the data. CAT1 will be a machine code file, and an error will result if the tape is not binary.

2) `COPY  1/CLOCK  =  0/CLOCK`
   This will copy the BASIC program CLOCK from Logical Unit zero to Logical Unit one. 0/CLOCK may now be deleted if desired. It is not necessary to use the same name for the destination file.

3) `COPY ENTERS = ENTERS1,ENTERS2 , ENTERS3`
   This will merge the three source files listed into one destination file. It is useful to merge text files in this manner when necessary to ASSEMBLE more than seven source files.

If copying to or from magnetic tape, only one of two special command forms may be used. They are:

```
COPY  Filename ← $MTA@n
```

where n is the tape file number to be read into the specified disc file, and

```
COPY  $MTA ← Filename
```

which will write the specified disc file on tape and print out the tape file number where it was written. Each file written on tape is automatically written immediately following the last file on the tape.

6. DATA FILES

All information stored in disc memory is organized into files. These include system files, processor files, program files, and data files. An example of a system file is the Real-time Executive (REX); an example of a processor file is the BASIC interpreter. Neither system nor processor files may be modified by the user. An example of a program file is a BASIC program. A processor or a program may write data into or read data from a data file.

6.1 Types of Data

Files may include data of several types, including binary, floating binary, packed decimal, and ASCII. BASIC programs are ordinarily stored in a compressed binary form for immediate interpretation and minimum storage space. However, the data produced by any language or any processor may be stored as strings of ASCII characters. This provides a common ground for transfer of data between one language or processor and another. For example, EDIT may be used to edit any ASCII text file.

6.2 Data File Structures

Each data file is made up of one or more blocks: a file header block and zero or more data blocks. The header block contains the following information: Filename, file type, privilege level, protection, record length, number of records, cost, and number of blocks. A formatted data file's header also contains a record format map and, unless there are more than 128 data blocks in the file, the disc address of each data block in the file. If data are written beyond the 128th data block then the file is automatically extended by the system. In an extended file, the disc addresses in the header point to header extender blocks, each of which may contain up to 256 disc addresses of data blocks. Thus, the maximum number of data blocks per file is 32768.

No data block is allocated until data are actually written into the block. Likewise, no header extender block is allocated until it is required to contain the disc address of at least one data block. Each data block is 256 words long and is divided into records.

Within a formatted file, each record is of the same length and format. The record length may be from one to 256 words, and each record may contain from one to 64 items. Each item may be an ASCII string, a binary floating-point number, a decimal number (one-, two-, three-, or four-word decimal format), or one or more binary words.

The contiguous file, which is discussed further in Section 6.5, has neither a format map nor a list of data block disc addresses. The entire file must be allocated in physically sequential disc blocks at the time the file is created. The size of a contiguous file is limited only by the size of the Logical Unit; thus, given a suitable Physical Unit, a contiguous file may exceed 65000 data blocks.

6.3 How to FORMAT a Data File

A new data file can be created and formatted by use of the system command FORMAT. To execute any system command, the terminal must be in control mode as indicated by a # symbol printed by the computer. If not already in control mode, press CTRL C and wait for the # symbol. Now type the command

```
FORMAT Filename
```

where Filename is any legal file identifier (see Glossary in the back of this manual). If the Filename given is already in use on the specified Logical Unit, then this command will be rejected, and another Filename must be tried. Or, if the user is replacing his own data file, he may enter the command in the form

```
FORMAT Filename!
```

The exclamation mark allows an existing file to be replaced.

Once the Filename has been accepted by the system, the computer will print

```
ITEM #0:
```

and allow the user to begin formatting the file. A shorthand notation is used to specify each type of item in a record. The abbreviations used are:

S	String of ASCII characters. The letter S must be followed by a number specifying the dimension (maximum number of characters) of the string. If an odd number is given then the dimension will be the next higher even number.
F	Floating point binary number. This number form is not used by Business BASIC but may be used by other processors.
D	Decimal number. This is a binary-coded decimal number of the form used by EDS Business BASIC. D or D1 specifies a one-word decimal integer in the range $\pm$7999. D2, D3, and D4 specify two-, three-, and four-word floating-point decimal numbers, respectively.

B Binary data. The letter B must be followed by a number specifying the dimension (number of words) of the item. Only one-word binary items are used by BASIC, but binary groups may be used by other processors.

These item identifiers must be separated by commas as they are entered by the user. A group of several items of the same type and dimension may be specified by entering the number of identical items ahead of the letter. For example, 3S16 would specify that the next three items are to be strings of up to 16 characters each. The RETURN key may be pressed after entering any item identifier, and the computer will ask for the next item by number. When all items have been specified, simply press RETURN again to close the file.

There are three ways to correct a typing error:

1. Use CTRL H to backspace and correct the error,

2. Use CTRL X to delete the line being typed, and retype the line, or

3. Press the ESC key. This aborts the formatting procedure, deletes the file, and returns the terminal to control mode. If an existing file was being replaced then it is restored to normal status.

The file created by FORMAT consists of only a header block. Data blocks will be automatically added by the system as data are written into the file. The file will be given maximum protection (protection digits 77 as described in "How to SAVE a Program") and zero cost, but these may be changed as desired by the use of the CHANGE command (see Section 5.2). The protection and cost may also be specified at this time by giving the command in the form

```
FORMAT  <pp>  $ddd.cc  Filename
```

where pp is the desired protection and $ddd.cc is the desired charge for access to the file.

The following example shows how a typical data file might be formatted:

```
#FORMAT  <20>  PAYROLL

ITEM #0:  S20, 4D2

ITEM #5:  2S14, 2D3, S3, D1

ITEM #11:

RECORD LENGTH = 41 WORDS
```

Portions printed by the computer are underlined. This example will create a data file named PAYROLL. Each record of the file will be 41 words long. Since each block of a file is 256 words, there will be six records per block with ten words unused at the end of each block. Each record will contain a string of up to 20 bytes, four two-word decimal floating-point numbers, two strings of up to fourteen bytes each, two three-word decimal floating-point numbers, a string of up to four bytes (string lengths are rounded up to an even number of bytes), and a one-word decimal integer. When item #11 was requested, the user pressed RETURN to indicate that all items had been specified. The protection 20 specified will allow full access to the file except that lower privilege users will not be allowed to write into or delete the file.

A data file may also be created and formatted by use of a BUILD statement in a BASIC program. Refer to the Business BASIC Programming Manual for more information.

6.4 How to KILL a Data File

The procedure for deleting any type data file is the same as for deleting a saved BASIC program or any other file. Type the system command

```
KILL Filename
```

where Filename identifies the file to be deleted. See "How to KILL a Saved Program" in this manual for more information on deleting files.

6.5 Contiguous Data Files

The contiguous file structure was conceived to accommodate mixed record formats in a data file whose size is limited only by the size of a Logical Unit, yet permit random access to any record or item in the file with an absolute minimum number of disc transfers. No more than one disc transfer is required to access any block in a contiguous file, whereas up to three disc transfers may be required to locate and access a block in a formatted file (one to read the header, a second to read the proper header extender if the file is extended, and finally the data block may be read into core). To accomplish this, all necessary information is brought into core when the file is opened.

The information kept in core for an open contiguous file consists of the header block disc address, the number of records, and the record length (number of words). For the system to calculate the disc address of a randomly selected record from this information, it is necessary that the file occupy only physically sequential disc blocks. Since it is unlikely that more blocks will be available in sequential locations at a later time, it is necessary that the file be initially built with the maximum number of blocks expected to re required. Only by means of a subsequent "garbage collection" can a contiguous file be expanded (see "Garbage Collection Procedures" in the IRIS Manager Reference Manual). Of course, a larger contiguous file might be built and the data copied into it, but this would be impossible if the file was half as large as the Logical Unit.

A contiguous file may be built by entering a system command of the form

```
FORMAT $ddd.cc <pp> [r:w]   Filename
```

where $ddd.cc and <pp> are the cost (if any) and the protection (if other than 77). The Filename may, of course, include a Logical Unit number and may be followed by a "!" to permit replacing an old contiguous file by the same name. The only thing new about this command is the field set off by square brackets; the brackets themselves indicate that a contiguous file is to be built. The value r before the colon indicates the number of records the file is to contain, and the value w indicates the number of 16-bit words per record. Any decimal values are allowed for r and w as long as 1+r*w/256 sequential disc blocks can be located on the specified Logical Unit. One additional block is required if r*w/256 has a fractional part. It is strongly recommended, however, that w be 256 or a smaller power of two so that some number of records will exactly fit in each block; otherwise, some records will span two or more blocks, thus requiring more than one disc transfer to access a single record and perhaps even to access a single item. The following is an example of a command to build a contifuous file:

```
FORMAT <33> [3000:128]   2/INDATA
```

This command will build a contiguous file named INDATA with protection 33 on Logical Unit number two. It will contain three thousand records of 128 words each, thus occupying 1501 disc blocks.

Access to a contiguous data file is similar to a formatted file except that a beginning byte number must be supplied instead of an item number. Since there is no format map, the system cannot check for matching item types; therefore, it is left to the user to avoid writing a string and reading it back into a numeric variable or vice versa. The user must also be careful not to supply a byte number which is in the middle of a previously written numeric item. Such errors will usually not be detected by the system and will not affect system operation, but the data retrieved from the file will be unrecognizable. Refer to the Business BASIC Programming Manual for more information.

7. TEXT FILES

A text file is an unformatted data file which contains a single text string of from zero to 16,777,215 characters or until a Logical Unit is filled, whichever occurs sooner. The text is tightly packed with lines of text separated only by RETURN codes and pages separated by FORM codes. All characters are stored as seven-bit ASCII codes with the eighth bit unconditionally set to one. Each block of a text file can hold 512 characters, and all blocks except the last block in the file will be filled. The string is terminated by one or more zero bytes (octal 000, not an ASCII zero which is 260 octal).

7.1 How to Load a Text File

A text file may be created from any terminal by use of the EDIT command (see "How to EDIT a Text File" in this manual) which allows the text to be either typed in or, if available on paper tape in ASCII code, to be loaded through the terminal's tape reader.

If a high-speed paper tape reader is included in the system it may also be used to load an ASCII tape, such as an assembly language source tape. To load a tape in this manner, place the tape to be loaded in the high-speed reader, and then type the system command

```
COPY  Filename=$PTR
```

where Filename is the name under which it is desired to load the tape. If the Filename is acceptable, a text file by that name will be built, and the tape will be loaded into the file. If the master terminal is not in use then its tape reader may also be used as a peripheral device to load the tape by giving the system command

```
COPY  Filename = $PTM
```

which will load the tape as described above. This command may not be given from the master terminal itself.

COPY checks for even parity on all characters read from tape. Any character in which a parity error is detected will be replaced by a back-slash code. All null codes (blank tape), rubout codes, and line feed codes will be ignored.

However, if the second frame after the leader is a null (blank frame), COPY checks that each 50th character thereafter is also followed by a null. An error is indicated if the nulls are not found as expected. See Section 7.5 in this manual.

7.2 How to EDIT a Text File

The EDIT processor may be used to edit or examine an existing text file or to create a new text file. To edit an existing file, type the system command

```
EDIT   sfile,dfile
```

where sfile is the Filename of the text file to be edited (the source file) and dfile is the Filename under which the edited text is to be stored (the destination file). This command will be rejected if sfile does not exist, is not a text file, or is read protected or copy protected. The command will also be rejected if the Filename dfile is in use to identify another user's file or any file other than a text file. Note that a new source file may be selected at any time by use of the F command.

If it is desired to examine a text file without editing it or creating a new file, the destination Filename may be omitted. In this case, the system command

```
EDIT   Filename
```

would select the text file identified by Filename to be examined. Any attempt to edit or reproduce the file will result in an error since no destination file has been provided.

A new text file may be created by typing the system command

```
EDIT   ,Filename
```

which will create a text file under the Filename given and will allow text to be entered through the terminal's keyboard or tape reader. To load text from a tape, first use the H command to change the string delimiter to any character known not to be on the tape, type the letter I to set insert mode, and start the tape reader. After reading the entire tape, shut off the reader, type the delimiter character, and press RETURN to terminate the insertion.

-nT Type from beginning of nth line back from current line through end of current line. Does not move the pointer.

U Print number of lines in current page.

V Print line number of line where pointer is positioned within current page.

W Print page number of current page of source file. If pages have been appended, this gives the number of the last page appended.

XEND Exit from editor after duplicating remainder of source file (also see CTRL C below).

XKIL Exit from editor and abort the destination file. If another text file by the same Filename was being replaced then only the destination file is deleted, and the old file is restored to normal status.

Y Print number of bytes remaining in the edit buffer (space available for more additions or insertions in the current page).

Z Move pointer to end of current page.

* Note: in commands marked by an asterisk, n may be negative.

/ Represents the symbol currently being used as the delimiter character (string terminator). The slash is recognized as the delimiter until changed by an H command. A RETURN also acts as a string terminator except in insert mode.

$\underline{C}$ A CTRL C will cause an exit from the editor after writing the current page and closing the destination file.

$\underline{Z}$ A CTRL Z entered as part of a string is entered into the string as a RETURN code.

The RETURN key acts as a command activator except when in insert mode, in which case the RETURN code is inserted as part of the string.

All commands operate only within the current page unless otherwise specified.

Special cases for n=0:

0C	Illegal
0D	Illegal
0G	Insert a copy of the current page of the source file.
0J	Same as J
0K	Delete from current pointer position back to beginning of line.
0L	Same as L
0N	Illegal
0P	Replace current page with its original form from the source file.
0Q	Illegal
0R	Illegal
0S	Illegal
0T	Same as T

Any pointer move command (J, L, or M) merely stops with no error indication if the beginning or end of the current page is reached. All other commands cause an error typeout if the command cannot be carried out for any reason.

Any search command (C, N, Q or S) searches forward starting at the current pointer position. A successful search leaves the pointer positioned after the string. The C, E, and S commands search only to the end of the current page; if unsuccessful, an error message is printed, and the pointer is not moved.

If an A or G command is not completed because the buffer is filled, an error message is printed. However, part of the appended or inserted page will be retained.

7.3 How to Merge Text Files

Several text files may be combined into a single large file by typing the system command

```
COPY   Filename=file1,file2, . . .
```

where the Filename given identifies the new combined text file, and file1,file2,... identify the text files to be combined. This command will be rejected if the Filename given is already in use to identify another user's file or any file other than a text file. It will also be rejected if any of the files file1,file2,... is not a text file. This form of the COPY command is particularly useful for merging assembly language source files when it necessary to include more than seven source files in an assembly.

The source files file1, file2, etc., are not disturbed by use of the COPY command. However, since all of the text has been reproduced in the combined file, the original source files may now be deleted if desired.

7.4 How to KILL a Text File

The procedure for deleting a text file is the same as for deleting a saved BASIC program or any other file. Type the system command

```
KILL Filename
```

where Filename identifies the file to be deleted. See "How to KILL a Saved Program" in this manual for more information on deleting files.

7.5 How to Punch a Tape of a Text File

An ASCII tape of a text file may be punched at any terminal by using the EDIT processor to type the entire file with the terminal's tape punch on. However, if a high-speed paper tape punch is included in the system, the user may type the system command

```
COPY $PTP=Filename
```

where Filename identifies the file to be punched. The entire file will be punched with even parity. Leader and trailer will also be punched, and a null (blank frame) will be punched after the first byte and after every 50th byte thereafter. COPY uses these nulls to check for characters being picked up or dropped by the paper tape equipment (see Section 7.1 in this manual). If the master terminal is not in use then its tape punch may also be used as a peripheral device by giving the system command

```
COPY $PTM = Filename
```

which will punch a tape in the same form as described for $PTP. This command may not be given from the master terminal.

8. ASSEMBLY LANGUAGE

Assembly language is a symbolic representation of machine code, the only language understood by the computer itself. For information on programming in assembly language, refer to the Data General publications How to Use the Nova Computers and Introduction to Programming the Nova Computers or to the Digital Computer Controls publication D-116 Programmer's Manual. A program called an assembler is used to translate from assembly language to machine code. Assembly language is then used to write other programs such as the BASIC processor, which interprets the BASIC language and selects the proper machine code subroutines to perform the operations specified by the user's BASIC program.

The IRIS system provides extensive facilities for writing and debugging programs in assembly language; in fact the IRIS system may itself be extended by using these facilities to create new subroutines and processors to be added to the system. The IRIS System Reference Manual provides the information required to write programs in the proper form to become part of the system. The IRIS Manager Reference Manual tells how to load such extensions as new system components.

8.1 How to Manipulate Source Files

An assembly language source file is simply a text file containing a program written in assembly language. All of the techniques described under "Text Files" (Section 7 of this manual) may be used to manipulate assembler source files. Up to seven source files can be handled in a single assembly; COPY must be used to combine source files if a a program to be assembled is segmented into more than seven files.

8.2 How to ASSEMBLE a Program

ASSEMBLE is a disc-to-disc machine language assembler which is fully compatible with the stand-alone Nova absolute assembler. One restriction, however, is placed on programs to be assembled: the first word of object code generated must be the lowest address in the object file.

To begin an assembly, type the system command

```
ASSEMBLE  Filename, source1, source2, . . . , source7
```

where Filename is the name under which the object file is to be generated, and source1, source2, . . . are names of existing text files containing assembler source code. Up to seven source files may be specified. Refer to "How to Merge Text Files" in this manual if the number of source files exceeds seven.

```
*****************************************************
* Note: on some systems the ASSEMBLE processor is   *
* loaded under the name ASM.  In this case, substitute *
* ASM for ASSEMBLE in all commands described in     *
* this section.                                     *
*****************************************************
```

The terminal's carriage will not be returned immediately after entering the command. During this pause, ASSEMBLE is locating the source files and building the object file. A carriage return will then be executed to indicate successful completion of these initial tasks, and each .EOT and .END pseudo-operator will be printed during pass one of the assembly. Any errors detected during pass one will also be printed.

Before starting the listing on pass 2, ASSEMBLE will print "PRESS RETURN". Roll the paper up at this time to the desired starting point, then press the RETURN key. This will position the listing properly on each page.

The listing may also be sent to a data file or a peripheral device by entering the command in the form

```
ASSEMBLE Filename, @dest, source1, source2, ... source6
```

where dest is the name of a peripheral device such as $LPT (line printer) or a Filename under which a data file is to be built to store the listing. Note that only six source files may be specified in this case, and the user will not be asked to press return before the listing is started.

If an object file only is desired (no listing), type a minus sign ahead of the object Filename; i.e., enter the command in the form

```
ASSEMBLE -Filename, source1, source2, ... source7
```

This "no list" assembly may also be specified on individual source files by typing a minus just ahead of each source file name which is not to be listed. A typical command might be

```
ASSEMBLE PAYROLL, @$LPT, -DEFS, PRS1, PRS2
```

indicating that the object file is to be named PAYROLL, the source file DEFS is not to be listed, and source files PRS1 and PRS2 are to be listed on the line printer.

If a listing only is desired (no object file), merely omit the object Filename, and enter the command in the form

```
ASSEMBLE ,source1, source2, ... source7
```

or

```
ASSEMBLE ,@dest, source1, source2, ... source6
```

In some cases it may be desirable to do a preliminary assembly, producing neither a listing nor an object file, for diagnostic purposes only. In this way, all errors which can be detected by the assembler will be printed, and the source files can be re-edited before final assembly and debugging. To do this, substitute a minus for the object Filename; i. e., enter the command in the form

```
ASSEMBLE -, source1,source2,. . . source7
```

Another form of diagnostic assembly is possible which will print the symbol table without the listing. This is done by typing a minus ahead of each source Filename but not ahead of the object Filename; i.e.

```
ASSEMBLE Filename, -source1, -source2,... -source7
```

or

```
ASSEMBLE , -source1, -source2, . . . -source7
```

will print the symbol table after doing a no-list assembly. The first form will generate an object file, the second will not.

An assembly may be caused to pause at any time by pressing the terminal's space bar. Press the RETURN key to resume the assembly. This pause feature may be used if it is necessary to reposition the paper or to load fresh paper or a new ribbon, etc. The assembly may be aborted at any time by pressing the ESC key; however, the object file being generated will be lost unless pass 2 has been completed and the symbol table is being printed.

Besides all of the standard assembler errors, one additional error may be indicated by ASSEMBLE. It is:

Z   Out of disc space

in which case, no more object code will be generated. Also, an L error will occur if object code is generated at any address less than the first word of code generated, and that object code will be lost.

The port's active file is used for the assembler's symbol table. Should symbol table overflow occur, only the first such error indication is printed during pass one. However, all errors are printed during pass two.

8.3 How to EXECUTE a Program

The EXECUTE processor may be used to run or debug a stand-alone machine code program. Programs which may be executed include:

1. Stand-alone programs already on the disc, such as GPM.

2. New machine code files created by use of ASSEMBLE.

3. Machine code programs available as a binary object tape. The system Manager can use PLOAD to load such a tape as a disc file.

Programs which have not been debugged may be executed since EXECUTE is an interpreter; it has special facilities such as break-points and a trace mode for debugging a new program.

EXECUTE can be used only on the master terminal, which must first be in control mode as indicated by a # symbol printed by the computer. To select a program to be run, type the system command

```
EXECUTE   Filename   k
```

where Filename is the name of a stand-alone or executable file, and k is the amount of virtual core to be used (1k=1024 words). Such files are indicated by the letter M or X, respectively, in a library listing. An up arrow is printed as a prompt character to indicate that EXECUTE is in the command mode.

A single letter is used for each command. Most commands also require an address which must be given as an octal number. However, some commands consist of the letter only, and others require more than one octal parameter. In the following list of commands, all such parameters are represented by lower case letters. Provided EXECUTE is in the command mode as indicated by a ↑ symbol printed by the computer, you may enter any command by typing it in the form shown, then pressing the RETURN key.

An, x	Change contents of an accumulator (selected by n=0, 1, 2, or 3), the carry flip-flop (selected by n=4), ION (the "interrupt on" switch, selected by n=5), or the interrupt mask (selected by n=6) to the value x. When a program is stopped for any reason, the current contents of all registers are saved. Restarting the program causes all registers to be restored to the saved values, which may be changed by the A command or interrogated by the Q command.
Bx	Insert breakpoint at location x in the program. If the breakpointed location is encountered as an instruction while running the program, execution will be terminated, and the current status (similar to Q command) will be printed. If x is zero or omitted, the breakpoint will be removed.
Dx	Dump (print) the program or its data in octal starting at location x. Press ESC to terminate the print out.
Ex	Allow entry of octal numbers or symbolic instructions starting at location x. The colon command (see below) may be used to enter into a single location. Symbolic instructions similar to assembly language may be entered, but labels are not allowed. Either relative or absolute addresses may be given. Press ESC to terminate entry mode.
Ix: string	Input ASCII text string starting at location x. Any letter, digit, or symbol and many control characters may be used, but characters recognized by the system (see Appendix 3) may not be entered. Type a CTRL Z to enter a RETURN code as part of a string.
Jx	Jump into the program and begin executing at location x. If the program was previously run, all registers will be restored to their contents at the time execution was stopped.
J	Same as Jx except that execution resumes at the point it was last stopped. If the program was not previously run, execution will begin at the starting address given in the file's header.
Kx, y, z	Store the constant z in locations x through y, inclusive.
Lx	List the program as symbolic instructions starting at location x. Press ESC to terminate listing.

Mx, y, z	Move the contents of locations x through y, inclusive, to the area starting at location z. The source and destination areas may overlap without loss of data.
Nx, y, z or Nx, y, z, m	These commands are the same as the respective S commands except that a not-equal comparison causes the contents of a cell to be listed.
Ox	Output the ASCII string starting at location x.
Px, y	Punch locations x through y, inclusive, on the high speed paper tape punch in binary loader format. Note: leader and trailer may be punched by use of the Y command.
Px	Punch an end block on the high speed speed punch with a starting address x.
P	Punch an end block on the high speed punch with no starting address.
Q	Query status. Prints the contents of all registers as they were saved when execution was stopped.
R	Read binary format paper tape on the high speed reader.
Rx	Same as R except that all addresses on the tape are displaced the same amount so that the first word on the tape goes into address x.
Sx, y, z	Search locations x through y, inclusive, for the value z. Prints the location and its contents if the value is found.
Sx, y, z, m	Same as Sx, y, z except that the contents of each location are ANDed with the mask m before being compared with the value z. May be used to search for a certain type of instruction. For example, the command

```
S600,1120,25,101777
```

will search locations 600 through 1120 for any instruction that references location 25.

Tx	Trace the program starting at location x. For each instruction executed, the location is printed, and the instruction is printed in symbolic form. Also, the contents of any register that changed as a result of the instruction will be printed. If interrupts are enabled or disabled, an E or D will be printed.
T	Resume running in the trace mode. Similar to J command except the program is traced.
X	Exit and clear the active file. The X command should be used to leave EXECUTE to restore the active file to its normal size. CTRL C may be used to leave EXECUTE temporarily for the purpose of saving a copy of the active file, but the user should re-enter EXECUTE and exit with an X command when finished.
Yx	Causes x blank frames to be punched on the high speed paper tape punch. If x is zero or is omitted, approximately three feet of blank tape will be punched. May be used to punch leader and trailer before and after use of P commands.
Zx	Search for relative reference. The 256 words centered on location x are searched for any storage reference instruction that references location x using relative addressing. Any such instruction is listed in symbolic form.
a:x	Store the value x at location a. Similar to the E command except that only a single location is entered.

If a HALT instruction or an illegal I/O instruction is encountered, EXECUTE will print an appropriate message and return to command mode. The user may request a return to command mode at any time by pressing CTRL V. A J or T command may later be used to resume running the program at the point where it was stopped.

While EXECUTE is running a program, the special system control characters (such as ESC, CTRL C, RETURN, and CTRL A) are ignored by the system and passed through to the program being run. A CTRL V must be used to return to command mode before these active characters will be recognized by the system.

8.4 How to Punch a Tape of a Program

To obtain a binary object tape of a machine code program file, the high-speed paper tape punch must be included in the system. To punch such a tape, type the system command

```
COPY  $PTP=Filename
```

where Filename is the name of the object program file. The program will be punched in standard binary format with leader and trailer on the tape. Locations containing a standard HALT command (octal 63077) will not be punched.

8.5 How to KILL a Program File

Object program files may be deleted in the same manner as described for BASIC programs in "How to KILL a Saved Program" in this manual unless the file is a processor. To delete a processor, type the system command

KILL $\underline{E}$key$\underline{E}$ Filename

where "key" is the password to KILL, as set by the system manager, and Filename is the name of the processor to be deleted.

9. HOW TO USE TUTOR

This chapter provides the course author with the information he needs to write a course using the TUTOR language. A quick summary is given at the end of the chapter.

9.1 Naming a TUTOR Program

First, a name must be chosen for the course, It must not exceed ten valid alphabetic or numeric symbols and must start with a letter. Mnemonic conventions may be used in selecting course names, e.g. the first three characters might indicate the subject.

Examples:	ALGSIM is a course in the area of algebra dealing with simultaneous linear equations.
	CHEMOL is a course in the area of chemistry dealing with molecular weights.
	BIOPHO is a course in biochemistry on phospholipids.

These names are normally listed in the catalogue of available courses and are the words the students type to begin taking a course. Use QUERY to determine whether the desired Filename is already in use. You can now use the program CRAM to generate your course. CRAM is described in Sections 9.7 through 9.9.

9.2 Writing a TUTOR Program

The operations that the computer performs are controlled by a simple code language. The course author uses two letter operation codes to identify questions, correct answers, and incorrect answers to TUTOR. Supplemental instructions, hints, and anticipated responses are all under control of the course author through the use of appropriate codes. The course author also controls the instructional flow by anticipating the student's performance. Thus, branching and concealment of unnecessary drill for better students is allowed, and return to the main flow of instruction is easily performed.

There are seventeen operation codes each consisting of two characters. They are:

LA This code followed by a text of not more than 6 characters is placed at a point in the course to which the author may wish to branch. It is called LABEL code, and its text is called the label.

Examples:
```
LA1
LAOPEN
```

BR A branch code followed by a label causes a branch (transfer) to the point in the course so labeled. The matching label may be placed anywhere in the course, and control is taken up at that point. The label after BR cannot exceed 6 characters. Several BR statements may refer to a single LA code.

Example:
```
BRHELLO     Corresponding LA code would be --
LAHELLO
```

RD This code is used to display a text frame. (With a CRT terminal, the screen is cleared). The course may be started with this code if no reference to it is required by the program after it has once been displayed. It should be preceded by an LA code if reference will be made. To go on to the next instruction, the student must press the RETURN key.

___ This is the blank (space) code. It is used to introduce continuation lines when the text accompanying a QU, RD, TY, ER, or UN code will not fit on one line.

Example:
```
RDTHIS IS A SAMPLE OF A MESSAGE THAT IS
   TOO LONG TO FIT ON ONE LINE.

   NOTE THAT EACH OF THESE LINES BELOW
   THE FIRST ONE USES BLANK CODE.
```

QU This code displays the text frame following it (a question) and reads the student's response to the question. A course may also begin with a QU code if no future reference will be made to this question. After displaying the text of the QU and any following blank codes, the computer matches the student's reply starting with the next code (which must be a CA, CB, CM, or CS code) to determine whether the reply is correct, wrong, or unrecognizable.

CA This code precedes the correct answer. If its text matches the student's answer character for character, control moves past any CB, CS, CM, or blank codes immediately following the CA to the TY or BR immediately following them if such exists. If no TY or BR comes next, the computer types an automatic "Correct, press RETURN" message, and control moves to the next QU or RD in the sequence when the student presses the CR or return key. A chaining capability allows several anticipated answers which are to be treated alike to be placed after any CA, CB, CM, WA, WB, CS, or MS statement. These eqivalent answers should be separated by semicolons, and there should be no spaces anywhere except where the student is expected to insert a space.

Examples:

```
QUtext 1
CAanswer 1
TYtext 2
QUtext 3
```

While the student is viewing text 1, he writes in an answer. If it matches answer 1, text 2 is written under his answer. When he presses the RETURN key, text 3 is displayed on the screen.

```
QUtext 1
CAanswer 1; answer 2
BRA
LAB
RDtext 2
LAA
QUtext 3
```

This time if the student gives answer 1 or answer 2 (using the semicolon chaining) the computer branches to label A and erases text 1, replacing it with text 3. Note that this bypasses text 2 which may have contained remedial material, and may be branched to somewhere else in the program by a BRB line.

```
QUtext 1
CAanswer 1
CAanswer 2
TYtext 2
```

If you are lazy, the procedure at the left is for you. It would type an automatic "Correct, press RETURN" if the student gave correct answer 1 before going in to show text 3. If he gave answer 2, he would see text 2 before going on.

CB This code may be used to provide a sequence of alternative correct answers that the author wishes to treat as equivalent. The string of CBs should follow the last CA (if there is any). If the computer finds a student's answer matching one of the CB tests, it will bypass the others and any following CS or CM and execute the TY and/or BR following the last CB if there are any, otherwise, the automatic message is printed and control moves to the next QU or RD or to the 99 code.

Example:

```
    QUtext 1
    CAanswer 1
    TYtext 2
    CBanswer 2; 3
    TYtext 3
    QUtext 4
```

If answer 1 is given, text 2 is displayed under it. When the RETURN key is pressed, text 4 will replace what is on the screen. If either answer 2 or answer 3 is given, text 3 will follow on the screen. Text 4 will replace that when the RETURN key is pressed. Any other answer will cause automatic appearance of text 4 without telling the student he missed the question.

NOTE: The semicolon chaining capability has pretty well done away with the use of the CB code. In the example the two CB codes would be replaced by CAanswer 2; answer 3. In other respects it is as a CB code.

CS This code (Correct String) is to be used if the answer(s) following the code will be accepted if found anywhere in the student's response. It may be used to find a certain word, character, or portion of a word in the student's response.

CM This code is to be used if you are expecting all of the strings which are separated by semicolons to be somewhere in the student's answer to count him correct. In other respects it is as a CB code.

WA This code is for an anticipated wrong answer. It should follow the last CA or CB and their related TY and/or BR codes. It is treated similarly to the CA except that the student's answer is marked wrong and he is expected to give another response, which will be reprocessed through the CA-CB-WA sequence. If no TY follows it an automatic "Wrong, try again" message will appear before the student is expected to give another answer.

WB This code (alternative Wrong Answer) is similar to a CB code except that the answer is marked wrong and the student is expected to give another response if no BR follows. WB is used to provide a sequence of anticipated wrong answers which will be treated similarly. The need for WB is not great since the semicolon chaining capability can be used with WA.

WS This code (Wrong String) is used for wrong answers in the same way CS is used for correct answers. If any of the answers following it (separated by semicolon) are found anywhere in the statement's answer, he is counted wrong and procedure is as for WB after that.

Example:

```
    QUtext 1
    CA134
    CM1;3;4
    TYtext 2
    WS2;5
    TYtext 3
    WS1
    TYtext 4
    WS3;4
    TYtext 5
    ERInclude combination
         of 1,2,3,4, and 5
```

The CA was included so that a HELP given by the student would print out "134", but the CM allows answering in any order; e.g. the student would be correct if he typed "The answer is 1, 4, and 3". He would then see text 2. If CA and CM didn't stop the search for a match, the first WS line is searched. If either a 2 or 5 is found somewhere in the student's answer, he is shown text 3 and expected to try again. Furthermore, if 1 occurred in the answer other than with a 3 and 4, text 4 would be shown. If either 3 or 4 occurred in a combination without the 1, text 5 would be shown. If none of the above was in the answer the ER message would appear, the student not counted wrong but expected to try again. (See ER description).

Example:

```
    QUtext 1
    CAanswer 1
    BRA
    WAanswer 2
    BRB
    WAanswer 3
    TYtext 2
    WAanswer 4
    RDtext 3
    LAA
      .
      .
```

If the student gives answer 1, he is branched to label A often seeing the automatic "Correct, press RETURN" message. If he gives wrong answer 2, he is counted wrong and branched to B without comment. If he gives answer 3 he is marked wrong, text 2 is written under his answer, and he is expected to give another answer. If he gives answer 4, he will be counted wrong and be expected to give another answer (which he will have been told to do by the automatic "Wrong, try again" message). If his answer is none of the above, text 3 will be displayed next. (An ER or UN would have prevented that).

Example:

```
    QUtext 1
    CAanswer 1
    WAanswer 2
    WBanswer 3
    WBanswer 4
    BRB
    RDtext 2
    LAB
    RDtext 3
```

If answer 1 is given, the student sees text 2 after an automatic "Correct, press RETURN" message. If 2, 3, or answer 4 is given, the student will be counted wrong and see text 3 next comment on his answer.

Note that the WA, WB statements could have been replaced by a WAanswer 2; answer 3; answer 4 statement.

MA This code is used to check for missing strings. It checks to see if the text following it is not found anywhere in the student's response. If the text is not found, the action is identical to a WA code. A TY cod3 must always follow it to inform the student of his error.

Example:

```
QUtext 1
CA Y=MX; Y=XM
TY Correct
MS=
TY Write an equation
MS M; X; Y
TY Your equation
   should include
   variables named
   M, Y, and X
UN Read the question
   more carefully and
   try again
```

Text 1 is displayed. If student types in Y=MX or Y=XM, he is counted correct. If no equal sign is included in his response, he is counted wrong is told "Write an equation", and is expected to give another response. If any of M, X, or Y does not appear in his answer, he is counted wrong, gets another message, and is expected to give another response. If M, Y, and X are in the answer and also an equal sign, but the answer is still not as expected, the UN message is typed, and he is counted wrong (see UN description).

UN This code is used to comment on unanticipated answers. When UN is not used after a question, an unanticipated response will be ignored and the next QU or RD text will appear. When the UN code is used, the text following the code is displayed and the computer waits for the student to answer the question again. Often, the course author can anticipate wrong spelling in a student's response. He can use the UN to ask for a repeat of the answer. The question may require a true of false answer, and the student may respond with "Perhaps". The computer would skip to a UN, which could print: "Type either a TRUE or FALSE".

Several UN codes can be used for any given question. They will be executed one at a time as different unanticipated wrong answers are given. This allows for several hints or warnings to be given one at a time to help the student reach the right answer.

The last UN message will be repeated until an anticipated answer is given unless that last UN is followed by a BR statement which branches away from that UN. If the UN statements do not make it clear how the student is to respond to a question which is neither true-false nor multiple choice, the last UN statement should be followed by a BR.

Example:

```
QUtext 1
CA TRUE; T
WA FALSE; F
TYtext 2
UN TYPE EITHER
  TRUE OR FALSE
QUtext 3
```

If TRUE or T is typed as an answer, text 3 appears after the automatic "Correct, press RETURN" message. If FALSE is typed as an answer, text 2 is displayed and the student is expected to try again. (His answer is counted wrong though). If neither of these answers is given, he is counted wrong and the UN message appears under each successive answer until finally the student decides to write either TRUE or FALSE.

Example:

```
AUtext 1
CAA
WAB; C
TYtext 2
ER TYPE A, B, or C
UN I CAN'T PUT UP
  WITH THIS.  YOU'RE
  THROUGH!
BR OUT
RDtext 3
```

The reaction to anticipated answers is the same as in the previous example. However, the ER will be given for the first unanticipated answer with the computer waiting for another try (student not counted wrong). The second time, the student receives a different response if he persists in giving unanticipated answers and is counted wrong. The last message is printed, and he is branched to the label OUT, which might be at the end of the course.

ER Exactly as UN without counting student wrong. Use as a first comment if it is not too clear what method of response the student is to give, and coaching is desired.

TY The TY code is a message to be typed out by the computer without erasing the QU text or the answer given. It is used to comment on an answer given by the student. It should immediately follow a CA, CS, WA, or MS or the last CB or WB in a sequence of CB's or WB's. See the previous examples on its usage.

99 This code without any accompanying text indicates to the TUTOR program termination of the course. The student will see a summary of his performance and be asked if he wishes further courses immediately after this code is processed. CRAM automatically inserts the 99 code so the author need not be concerned about its entry.

9.3 Operation Features of TUTOR

Operational features of TUTOR courses available to the student or author are:

a) after any question, he can type HELP, and the first anticipated response following the question will be shown to him. (Hopefully this is a CA, CB, CS, or CM). If it was CA, CB, or CS, only the text up to the first semicolon will be shown; if CM, then all of the text will be shown. He is marked wrong and is asked to type in the correct answer.

b) after being shown any frame he can type GOTO followed by any of the following:

1) a number identifying the frame to go to.

2) a plus sign followed by a number indicating the number of frames to go forward starting at the present frame.

3) a minus sign followed by a number indicating the number of frames to go back starting at the present frame.

4) a label identifying an LA statement followed by the frame to go to. (This is primarily for use by authors.)

c) at any time he may press the ESC key and get a termination message before exiting from TUTOR. This does not work in DIAG mode (to be discussed in Section 9.4).

d) after being shown any frame he may press CTRL and C simultaneously instead of giving an answer, and he will be exited from TUTOR without any message.

9.4 Testing the Program

After an author has written a TUTOR course or course segment he can immediately run it as any student might do. If he is signed on with at least a level 1 account, he has one advantage over the student. However; if he types DIAG at the time a response is appropriate, he will be transferred to a diagnostic mode. He may then continue as a student might or he can type DUMP to get a listing of the course as entered, starting at the beginning of the frame he is currently on. He can press the ESC key to stop the listing. Conditions will then be restored as they were before DUMP was requested; i.e. the program will expect an answer or RETURN. Typing DIAG again will cause a return to student mode.

9.5 Random Access Mode

Any course can be placed in a random accessing mode. This allows the instructor to show students a random sampling of questions for a quiz-type of review or actually to give a quiz, recording scores for later printout. To use a course in this mode, the instructor should contact the computer installation manager and tell him how many random frames he desires the student to see when he takes the quiz.

9.6 Summary

1. Begin the course with an LA, RD, or QU code.

2. CA, CM, CS, or CB always follows a QU code.

3. The sequence after QU is: (CA, CS, string of CB's, CM), MS, (WA, string of WB's), ER, UN. The TY code is recommended after each of these (except ER and UN) and BR is acceptable to use if the normal sequence of operations is to be changed. All of these are optional except the first CA, CB, CS, or CM.

4. A BR code text must have matching LA text but LA's do not need a corresponding BR.

5. UN or ER codes are executed one after another as unanticipated answers are given until the last UN is encountered. If no BR follows, the last UN or ER is repeated.

6. The course must end with 99 as the last code followed by no text (inserted by CRAM).

9.7 CRAM

CRAM is the file creation and editing program for TUTOR. In using CRAM, you need to provide only one Filename--the name of the file you are going to edit or the name you wish a new file to have. There are six major steps to editing a TUTOR file:

1. You: From system control mode, type CRAM and press RETURN.
   Computer: What is the source file ?
   You: Enter the name of the file you wish to edit or create.

2. Computer: Leave line limit at 72 characters?
   You: If you want 72, enter "Y" or just press RETURN; or if you want some other number, enter "N", after which the computer will ask: What length? And you must enter the line limit that you want.

3. Computer: The file is being indexed
   Computer: *** Records will be searched
   Computer: 0 1 . . . . ***
   Computer:
   Computer: Check for invalid lines:
   You: If "Y" is entered (an unpredictable number of entries will be printed here). Computer: End of list.
   You: If "N" is entered. Computer: File is ready. Last line is xxx.
   Computer: Edit code?
   Explanation: The program is locating each line so it can find one quickly when asked to find a particular line. Then each line is examined for validity of length and TUTOR operation code. Each invalid line found is indicated by the printing of its number and a brief reason for its invalidity. Finally you are told how many lines are in the file.

4. All editing functions may now be used. See list Section 9.8.

5. You: x
   Computer: File is being saved--
   Computer: 0 1 . . . **
   Computer:
   Computer: #

Explanation: Temporary files were deleted.

9.8 CRAM Editing Functions

P-- Print the line specified (the line number is represented here by dashes). The line number will be repeated for clarity. Additional lines can be printed by pressing the RETURN key in response to the question 'More?' which is printed after each line.

C-- Change the contents of the line specified. The line is repeated and you must confirm that it is the correct line to change, then you will be asked to enter the replacement line. After one line is changed you will be returned to the edit code selection point.

I-- Insert a line with the specified line number. The preceeding line will be printed for reference of location. You will be asked if the line should go after the displayed line. If you say "N", you will be returned to the code selection point and no changes will be made. If you say "Y", you will be asked if you want more inserted. If you answer "Y", the number for the next line will print and you must make another entry. If you answer "N", you will be returned to the code selection point.

D-- Delete the specified line. The line will be printed for your inspection and you will be asked to confirm that it is the proper line. If you say "N", you will be returned to the code selection point. If you say "Y", the line will be deleted. If you say "Y", the delete cycle will start over, otherwise you will be returned to the code selection point. Deletions near the beginning of a long file may take 5 or 10 seconds to complete.

E-- Examine all lines related to the first question or text block following the line number given. The number of the line starting that group of lines will be printed, then all the rest of the lines will be printed as a group, showing actual spacing as it is arranged. The last line number of the group will be printed unless the last line in the group is also the last line in the file.

S-- Scan all lines for validity or for a specific TUTOR code. You will be asked if you want a validity check. If you say "Y", a print out as in step 3 in editing will be made. If you say "N", you will be asked which TUTOR code you want. After you answer, the scan will start. You will be asked if you only want to see line number where code occurs or see the entire line. At each code display you may go back to editing if you desire by answering "N" to the question whether or not to see more. At the end you will be asked if you want another scan. If you answer "N", you will be returned to the code selection point. If you answer "Y", the cycle will start over.

L-- Print the length of the specified line (how many characters are in the line and how many are allowed).

X-- Exit. Means you are finished and want your modified or created file saved in its present state. This exit may take several minutes for long files. You can now go to another port to try out the program if you desire and then can come back to make any changes observed without having to re-index.

IMPORTANT: The file is not changed or saved until you exit from the program!!!

KILL Means you wish to exit from CRAM removing temporary files that were used in the editing process. Make sure you have not made any editing changes since executing the last X command.

9.9 Comments on CRAM

You must tell CRAM when you are finished with each entry by pressing the RETURN key once.

Changing a line takes less time than deleting the old line and then inserting the new line.

Since deletions and insertions near the beginning of long files take a significant number of seconds, reviews of the file should be done every 50 lines or so as it is being created so that most corrections will be near the end.

If you make a mistake, you may "backspace", deleting characters one at a time starting at the last one entered and working back using the CTRL key simultaneously with the H key for a backspace if your terminal does not have a backspace key.

The TUTOR code for each line must be valid. If it is not, the program will not accept the line, and it will be necessary to re-enter the line.

If ESCAPE is accidently pressed, type 100 RUN and press the RETURN key. If CTRL C is accidently pressed, or if an error occurs such as disc filled, type RUN and press the RETURN key.

10. HOW TO USE FORTRAN-TO-BASIC (F2B)

The F2B Translator is a BASIC program which translates from FORTRAN to BASIC. Its purpose is to extend to the user, particularly to students of FORTRAN, a FORTRAN capability from a BASIC language system.

To use F2B, one writes a FORTRAN program according to the FORTRAN specifications given in the appendix to this section. The FORTRAN program is loaded into a data file and then translated. The translator makes several passes to complete the translation of the source program into BASIC. In addition to the source file, three other data files are used to store intermediate results. Thus, four data files are required, and these must be formatted by the user prior to translation.

In summary, the steps required to perform a translation are as follows:

1. Format four data files for the source program and intermediate data.
2. Identify the Filenames of the four data files to the F2B translator.
3. Load the source (FORTRAN) program.
4. Edit the program if necessary.
5. Translate the program. The translator will produce a punched tape of the object (BASIC) program.
6. Load the object program and run it in BASIC.

Following is a detailed account of these steps.

10.1 FORMATTING THE FOUR DATA FILES

Format four data files for storage of source and intermediate data as follows, pressing RETURN after each line typed:

```
#FORMAT ROOTNAME1
ITEM #0: S75
ITEM #1:

#FORMAT ROOTNAME2
ITEM #0: S75
ITEM #1:

#FORMAT ROOTNAME3
ITEM #0: 3F
ITEM #3:

#FORMAT ROOTNAME4
ITEM #0: S255
ITEM #1:
```

In the procedure, the four FILENAMES in the FORMAT commands consist of a ROOTNAME which is private to the user, followed by the digits 1, 2, 3, and 4. For instance, the user might use NED1, NED2, NED3 and NED4. Later, when the FORTRAN program is to be loaded, the translator will ask for the Data File ROOTNAME. In the present example, the user would type: NED.

10.2 LOADING THE FORTRAN PROGRAM

The translator consists of an editor, F2B1, and six additional programs: F2B2 through F2B7, which constitute the translator proper. F2B1 is used to load and make any necessary modifications to the FORTRAN program. When this has been done, the translation may be initiated. At any point the user may re-enter the editor to make additional changes to the FORTRAN program.

The steps to load a program are:

1. Press CTRL C; type RUN F2B1; press RETURN.
2. F2B will print: NUMBER OF STATEMENTS.
   If the program has not yet been entered, type 0. Otherwise, type the number of statements previously entered.
3. F2B will then ask for the ROOTNAME of the Data Files. Enter it and press RETURN.

4. F2B will print ENTER YOUR PROGRAM followed by a question mark. Enter your FORTRAN program by means of the paper tape reader or the keyboard. Terminate each line with a RETURN. The last line must be a null line; i.e., simply press RETURN.

10.3 EDITING THE FORTRAN PROGRAM

Once the program has been loaded, F2B will print: EDIT COMMAND ? The user may now modify, list or run the program by entering any of the following editing commands:

```
ADD n
DELETE n
CHANGE n
LIST
RUN
TAPE
```

Where n is a line number of a FORTRAN statement. The line numbers, which are assigned when the program is loaded, are printed when the program is listed. They are not part of the FORTRAN language and should not be confused with FORTRAN labels. Their only function is to facilitate editing.

1. ADD n

   This statement is used to add a line just before line n. After this command has been entered (All editing commands are terminated with RETURN) F2B will print a question mark. Type in the statement to be added, followed by RETURN. The statement will be assigned line number n, and all line numbers above will be incremented by one.

2. DELETE n

   This command deletes line n and closes the gap by decrementing all line numbers above n.

3. CHANGE n

   After entering the command, respond to the question mark printed by entering the new line, which will replace the present line n.

4. LIST

   This command causes the program to be listed in line number order with each line preceded by its line number.

5. TAPE

This command causes the source program to be listed without line numbers. Type TAPE, press RETURN, then immediately turn on the punch. The program will be listed, followed by a pause to permit the punch to be turned off.

6. RUN

After performing each editing command the editor will print: EDIT COMMAND ? and wait for the next editing command to be entered. When the editing is completed and the program is ready to be translated, type RUN, turn on the punch, and press RETURN. The translator will run, producing a punched tape and listing of the object program in BASIC.

10.4 RUNNING THE TRANSLATED PROGRAM

When the translator types **READY,** enter the translated program on tape through the reader. (IMPORTANT: Do not type NEW or any other command that would destroy the active file). To run the program, type RUN. To run a program at a later time, load F2B7 (by typing CTRL C F2B7 RETURN); then load the object program as described above.

If running the BASIC program indicates that further editing is required, return to step 10.2.1 above. When the program has been completely debugged and no further editing is required, the four data files should be killed so that they will not occupy space on the disc. This is accomplished as follows:

Press:	CTRL C
Type:	KILL ROOTNAME1, ROOTNAME2, ROOTNAME3, ROOTNAME4
Press:	RETURN

10.5 ERRORS

Errors in the FORTRAN program may be detected by the translator, or they may be detected by the BASIC interpreter which interprets the translator. In the latter case, typical BASIC error messages will be printed. If either type of error is detected, check your FORTRAN program to make sure it conforms to the FORTRAN specifications in the appendix.

If you still have trouble, ask your lab assistant for help. If the errors persist, send a listing of the FORTRAN program and a brief description of the problem to Educational Data Systems.

APPENDIX TO SECTION 10

FORTRAN TO BASIC

FORTRAN SPECIFICATIONS

1. Program form

1.1 The characters used are -
ABCDEFGHIJKLMNOPQ
RSTUVWXYZ 01234567
=+-*/(),. and a blank.

1.2 A line is a string of 72 characters in the following format:

Column(s)	Purpose
1-5	label
6	ignored
7-72	statement

1.2.1 LABEL - Any integer from 1 to 99999 inclusive. No label may be repeated within an executable program (a main program and zero or more subprograms).

1.2.2 STATEMENT - Statement formats are given in Section 4, below.

1.2.3 The one exception to the above line format is:
The COMMENT statement which has a C in column one and any characters in columns two through seventy-two.

1.3 Every program and subprogram must end with an END statement.

2. Data Types (Variables)

2.1 VARIABLE NAMES - A variable name consists of one to five alphanumeric characters, the first of which must be alphabetic. Variable names must not begin with a statement name; e.g., GOTOX is illegal.

2.2 VARIABLE TYPES - There are two types of variables: Integer and Real. The variable type is determined by the first character of the name or by the INTEGER or REAL statements.

2.2.1 INTEGER VARIABLES - If the variable name starts with one of the letters I through N and is not declared by an REAL statement or if it is declared by an INTEGER statement, it is of the integer type and may assume only integer values (positive, negative, or zero).

2.2.2 REAL VARIABLES - If the variable name starts with any letter other than I through N and is not declared by an INTEGER statement or if it is declared by a REAL statement, it is of the real type and may assume any real value.

2.3 ARRAYS - An array variable is of the integer or real type, followed by one or two subscripts in parentheses.

2.3.1 SUBSCRIPT EXPRESSIONS - The subscript may be any Integer expression which does not contain functions or subscripted variables.

2.4 FUNCTIONS - Function references are of the form

$$f(a_1, a_2, \ldots . a_n)$$

where f is the symbolic name (one to five alphanumeric characters, the first of which must be alphabetic), and the a's, called actual arguments, may be an expression of the same type as the corresponding dummy argument (See paragraph 2.4.1).

Functions are of three types: statement functions, intrinsic functions, and external functions. All functions are referenced by using the function reference in an arithmetic expression. In the case of user defined functions (i.e., statement or external functions), the actual arguments may not refer directly or indirectly to a user defined function.

2.4.1 STATEMENT FUNCTIONS - A statement function is defined internally to the program unit in which it is referenced. It is defined by a single statement of the form:

$f(a_1, a_2, \ldots . a_n) = e$

where f is the function name, e is an arithmetic expression, and the a's, called dummy arguments, are variable names (no arrays). Aside from the dummy arguments, the expression e may contain constants, variables, and references to intrinsic functions, external functions, and previously defined statement functions.

When a statement function is referenced, the actual arguments from the function reference replace the dummy arguments in the expression, e; the expression is evaluated; and the value of the function, f, is returned to the calling reference.

2.4.2 INTRINSIC FUNCTIONS - The symbolic names of the intrinsic functions (See Table 1) are predefined to the F2B program and have special meaning and type as specified in Table 1.

An intrinsic function is referenced by using it in an arithmetic expression. Execution of an intrinsic function reference results in the actions specified in Table 1 based on the actual arguments from the function reference. The resultant value is then made available to the expression that contained the function reference.

Intrinsic Functions

Table 1

Name	Type	Definition
EXP(x)*	Real	Exponential function (e^x)
ALOG(x)	Real	Natural logarithm of x
SQRT(x)	Real	Square root of x
ABS(x)	Real	Absolute value of x
IABS(x)	Integer	Absolute value of x
SIGN(x)	Real	Sign of x(-1, 0, 1 for neg, zero, pos. numbers)
ISIGN(x)	Integer	Sign of x(-1, 0, 1 for neg, zero, pos. numbers)
SIN(x)	Real	Sine (x), x in radians
COS(x)	Real	Cosine (x), x in radians
TAN(x)	Real	Tangent (x), x in radians
ATAN(x)	Real	Arctangent (x) in radians

* x may be any expression, corresponding in type to the type of the function.

2.4.3 EXTERNAL FUNCTIONS - An external function is defined externally to the program unit that references it. It is defined by a function subprogram headed by a FUNCTION statement. See Section 5.

3. EXPRESSIONS

3.1 OPERATORS - The arithmetic operators are:

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

3.2 Expressions are built up from numbers, variables, functions, arithmetic operators, and parentheses according to the usual rules of algebra.

3.2.1 Every constant and variable in an expression must be of the same type, integer or real.

3.2.2 Integer values are achieved in the BASIC (object) program by applying the INT function to the result of each operation in the expression being evaluated.

4. STATEMENTS

4.1 ASSIGNMENT STATEMENT - The assignment statement is of the form:

v = e

where v is a variable and e is an arithmetic expression. Execution of this statement assigns the value of the expression to the variable, v.

4.1.1 The expression is evaluated according to type and is then converted to the type at the variable to the left of the equals sign.

4.2 GOTO STATEMENTS

4.2.1 UNCONDITIONAL GOTO - The unconditional GOTO statement is of the form:

GOTO k

where k is a statement label. Execution causes the statement labeled k to be executed next.

4.2.2 COMPUTED GOTO - The computed GOTO is of the form:

GOTO(k_1, k_2, . . . k_i) n

where the k's are statement labels, and i is an integer where $1 \leq n \leq i$. Execution causes the statement named by the nth label listed to be executed. n must be an integer variable.

4.3 IF STATEMENT - The IF statement is of the form:

IF(e) k_1, k_2, k_3

where e is an integer or real expression and the k's are statement lables. Execution causes the evaluation of e followed by branching to k_1, k_2, k_3 if the value of e is less than, equal to, greater than 0, respectively.

4.4 CALL STATEMENT - A CALL statement is of the form:

CALL s(a_1, a_2, ..., a_n)

OR

CALL s

where s is the subroutine name (one to five alphanumeric characters, the first of which must be alphabetic), and the a's are actual arguments. An actual argument in a subroutine reference may be a constant, a variable, or an array element name. The actual arguments must correspond in order, number, and type with the corresponding dummy arguments in the defining program.

Execution of the CALL statement passes the actual arguments and transfers control to the referenced subroutine. When the subroutine is complete, control is returned to the statement following the CALL statement, and the a's assume the final values determined by the subroutine.

4.5 RETURN - A RETURN statement is of the form:

RETURN

Every subprogram (subroutine or function subprogram) must include a RETURN statement. When this statement is reached, control returns to the calling program.

4.6 STOP - the STOP statement is of the form:

STOP

Execution of the statement causes execution of the program to stop.

4.7 DO STATEMENT - The DO statement is of the form:

DO k i = m_1, m_2, m_3

OR

DO k i = m_1, m_2

where k is the termination point (a label), and the m's are parameters.

4.7.1 TERMINATION POINT - The label k marks the end of the DO loop. The labeled statement must follow the DO statement and cannot be of the following types: IF, STOP, DO or either form of GOTO.

4.7.2 CONTROL VARIABLE - Any integer variable name.

4.7.3 PARAMETERS - The processor assigns the value of m_1 to the control variable when it initially comes to the DO statement. Upon reaching the terminating statement, it increments the control variable by m_3 (by one if m_3 is absent) and then runs through the same statements again, incrementing the control variable each time through the loop until the control variable is greater than m_2 . The processor then continues with the statement after the termination statement. The parameters must be integer variables or constants.

4.7.4 NESTED DO LOOPS - DO loops may not overlap. They may be completely separate, or one may be completely enclosed (nested) within another.

4.8 CONTINUE - Is used to mark the end of a DO loop; however, it is not required.

4.9 READ STATEMENT - The READ statement is of the form:

$$\text{READ (u, f) } k_1 , k_2 , \ldots k_n$$

where u is ignored, f is the FORMAT label, and the k's specify variables.

4.9.1 FORMAT LABEL - Identifies the label of the statement which contains FORMAT specifications for this statement.

4.9.2 VARIABLES - The variables may be given in two ways: by naming the variables or by a DO-implied loop, which is of the form:

$$(v(i), i = m_1 , m_2 , m_3)$$

OR

$$((v(i, j), i = m_1 , m_2 , m_3), j = n_1 , n_2 , n_3)$$

where i and j are subscripts as defined for the DO loop: DO k i = m_1 , m_2 , m_3 , in Section 4.7 above. Execution of this part of the READ statement results in reading in certain values of v as specified by the DO implied loops. When two loops are used, i is incremented until it reaches its termination point, at which time the j loop is incremented and i starts over again, etc. This goes on until j has reached its termination point. Then the next statement is performed.

4.10 WRITE STATEMENT - The WRITE statement is of the form:

WRITE (u, f) k_1 , k_2 , ... k_n

OR

WRITE (u, f)

where u, f, and the k's are defined as in 4.9, except that the variables are printed out, instead of being read in. Where there is no list of variables, only text is printed out, as defined by the FORMAT statement.

4.11 DIMENSION STATEMENT - The DIMENSION statement is of the form:

DIMENSION v_1 (i_1), v_2 (i_2), . . . , v_n(i_n)

where v(i) is an array declarator, v being the array name and i an integer constant.

4.11.1 ARRAY DECLARATOR - The array declarator sets aside, in memory, space for a variable v with dimensions i. i may be an integer between 1 and 999 (dimension one) or a pair of integers whose product is between 1 and 999 (dimension 2).

4.12 COMMON STATEMENT - A COMMON statement is of the form:

COMMON a_1 , a_2, . . . , a_n

where each a is a variable or array name. In any given COMMON statement, the entities occurring in the list of variable names are declared to be in common. All subunits (main program and subprograms) within an executable program store their common variables and arrays in the same area, thus providing a means by which the subunits can communicate with each other. Variables and arrays from a given subunit, which are declared to be in common, are stored in the sequence in which they are declared, starting at the beginning of common. Elements of an array are sequenced according to the SUCCESSOR FUNCTION (See paragraph 4.12.1).

4.12.1 SUCCESSOR FUNCTION - Array elements are stored sequentially in memory; the position of each element in the sequence is determined by the SUCCESSOR FUNCTION:

Dimensionality	Subscript Declarator	Subscript	SUCCESSOR FUNCTION (Position in Sequence)
1	(A)	(a)	a
2	(A, B)	(a, b)	$a + A \cdot (b-1)$

4.13 EQUIVALENCE STATEMENT - The EQUIVALENCE statement is of the form:

EQUIVALENCE $(k_1), (k_2), \ldots . (k_n)$

where each k is a list of the form:

$a_1, a_2, \ldots, a_n$

where a_1 is a variable name or the name of a single element in an array. Array names are given a single subscript whose value is an integer (the value of the SUCCESSOR FUNCTION - See paragraph 4.12.1). The EQUIVALENCE statement permits the sharing of space by two or more variables. Where arrays are equivalenced, they are stored in the same space in such a way that the elements named in one EQUIVALENCE statement variable list $(a_1, a_2, \ldots, a_n)$ are stored in the same location, and other elements of the arrays are lined up accordingly.

4.13.1 COMMON, EQUIVALENCE RELATIONSHIP - After an element is placed in common, it cannot be equivalenced.

4.14 FORMAT STATEMENT - The FORMAT statement is of the form:

FORMAT $(a_1 t_1 z_1 t_2 z_2 \ldots t_n z_n a_2)$

where $(a_1 t_1 z_1 t_2 z_2 \ldots t_n z_n a_2)$ is the format specification, and:
1) t is a field descriptor
2) z is a field separator
3) a is a series of slashes or is empty.

4.14.1 FIELD DESCRIPTORS - Field Descriptors are of the form:

rFw.d
rI w
nH$h_1 h_2 h_3 \ldots h_n$
nX

where:
1) r is the repeat count
2) w and n are the field widths
3) d is the number of places to the right of the decimal.
4) F, I, H, and X, called specifications, indicate the type of input/output required.

4.14.2 I or F SPECIFICATIONS - In the READ statement, a check is made to determine if there is a corresponding element in the Input list. If there is, w characters, including decimal point (if F-specification) and sign (if no sign, positive is implied) with d places to the right of the decimal place (if F-specification) is read in and assigned to the corresponding element. In the WRITE statement, if there is a corresponding element in the Output list, it is written out in similar fashion.

4.14.3 X-SPECIFICATIONS - In READ, the next n characters are ignored; in WRITE, n spaces are printed.

4.14.4 H-SPECIFICATIONS - In READ, n characters are read directly into the FORMAT statement, in place of the next n characters after the H. In WRITE, the next n characters after the H are printed. The processor starts at the n+1 character after the H when analyzing the rest of the FORMAT.

4.14.5 / (SLASH) - This signals the start of a new record. A new line is started with WRITE and a new input is started with READ.

4.14.6 There is a second way of repeating the specifications which is of the form:

$$m(a_1 t_1 z_1 \ldots . t_n z_n a_n)$$

where the field discriptors and field separators in the parentheses are repeated m times.

5. SUBPROGRAMS

An executable program consists of a main program and zero or more subprograms. The subprograms can be of two types: subroutines and function subprograms.

The main program or any subroutine may transfer control to a subroutine (other than itself) by means of a CALL statement. The CALL statement may also pass parameters to the subroutine. When the subroutine is complete, it returns control to the calling routine by a RETURN statement. Parameter values may also be returned to the calling routine.

A program unit (main program or subprogram) calls a function subprogram by using the name of the function in an expression. This transfers control to the function subprogram and passes the actual arguments of the function as parameter values. The function subprogram evaluates the function and returns the value of the function.

5.1 SUBROUTINE - The first statement of a subroutine must be a SUBROUTINE statement. It has the form:

SUBROUTINE $s(a_1, a_2, \ldots, a_n)$

OR

SUBROUTINE s

where s is the symbolic name of the subroutine (one to five alphanumeric characters, the first of which must be alphabetic) to be defined, and the a's, called dummy arguments, are variable names (no arrays). The subroutine name may not be used except in CALL and SUBROUTINE statements. After completion of the subroutine, the parameters that were passed to the subroutine are passed back, with new values if the variables were affected by the subroutine. A subroutine may not call itself.

5.2 FUNCTION subprogram. The first stateme[illegible] of a FUNCTION subprogram must be a FUNCTION statement. It has the form:

$$\text{FUNCTION } f(a_1, a_2, \ldots, a_n)$$

where F (one to five alphanumeric characters, the first of which must be alphabetic) is the symbolic name of the function to be defined, and the a's, called dummy arguments, are variable names (no arrays). The value of f at the time of execution of the RETURN statement is passed to the calling routine. A function subprogram may not call itself.

11. HOW TO USE GPM

GPM is an acronym for General Purpose Macrogenerator, a symbol string processor conceived by C. Strachey at Massachusetts Institute of Technology. Educational Data Systems' implementation of GPM is based on the original article on GPM which appeared in Computer Journal (published in England) in October, 1965, except that the BIN, DEC, and BAR primitives have been omitted, and two Read string primitives have been added. Also, the name of the UPDATE primitive has been changed to the more definitive REDEF, and some active symbols have been changed to use only standard ASCII characters.

11.1 Introduction to GPM

A macrogenerator is often included as part of a machine language assembler, and the result is known as a macro-assembler. The purpose of the macrogenerator is to allow the use of macro-instructions which it replaces with an appropriate sequence of machine instructions after substituting the actual parameters provided in the macro call for the formal parameters in the macro's definition. GPM can, in fact, be used in such a manner. However, since GPM is not directly associated with an assembler, the macros may be defined to produce assembly language code for any computer, and the resulting output from GPM can then be assembled on that computer's assembler to produce machine code. GPM can also be used for other purposes such as translating from one assembly language to another (for a similar computer), typing form letters with names and addresses filled in, and other tasks of a similar nature.

GPM is a symbol stream processor which takes as its input a stream of characters and produces as its output another stream of characters. The output is produced from the input stream by direct copying except in the case of "macro-calls" in the input stream, which are "evaluated" as described below before they are put into the output stream.

11.2 Macro Calls

A macro-call consists of a macro name and a list of actual parameters, separated by commas and enclosed in brackets, e.g.,

```
[ DEF, RQ, B9A+]
```

In this example, DEF is the name of a macro which uses two parameters. RQ and B9A+ are the actual parameters supplied in this call. An error will occur if not enough parameters are supplied, but extra parameters not required by the macro will be ignored. The parameters are numbered one through nine, thus allowing a maximum of nine parameters in a macro call. The name of the macro may be referred to as parameter number zero in the call.

The result of a macro call is a value which replaces the macro call itself in the symbol stream. This value may itself contain macro calls. In some cases the macro call has a null value; i.e., it generates nothing in the output symbol stream. The DEF primitive (see 11.5.1) is one example of such a macro; however, it has the side effect of creating a new macro.

11.3 Evaluation

Before a macro call can be evaluated, the macro must have been defined by associating its name with a symbol string. This string may contain the formal parameter symbols #1, #2, etc., which will be replaced by the first, second, etc., actual parameters supplied by the call. The symbol #0 stands for the name of the macro itself. When the macro is evaluated, each of these symbols is replaced by a direct copy of the respective parameter supplied by the call. Thus, if the macro name ARB has been associated with the string RB#1C#2VV, then the macro call

[ARB, 3X, $14.]

will produce the string RB3XC$14. VV as the output string, or "value" of the macro call.

The system is completely general, and it is possible to use a macro call in place of or in conjunction with any symbol string. In particular, macro calls may be nested; i.e., macro calls are allowed in the actual parameters of other macro calls (including the name) and also in the defining string. To illustrate this point, suppose we have defined the following macros:

name	associated string
A	A#1A
B	B [A, X#1X] B
APA	P#1#1P

we would then get the following results:

macro call	result of evaluation
[A, C]	ACA
[A, ACA]	AACAA
[A,[A, C]]	AACAA
[A ,XDX]	AXDXA
[B, D]	BAXDXAB
[A, P]	APA
[APA, Y]	PYYP
[[A, P], Y]	PYYP

11.4 String Quotes

The "less than" and "greater than" symbols are used as string quotes for two reasons: it is necessary to be able to differentiate between opening and closing quotes, and the standard quotation marks are more often used in text than the < and > symbols. Enclosing any string in the string quotes prevents evaluation of any macro calls inside the quotes; instead, one set (the outside pair) of string quotes is removed. The following examples use the macros defined above:

input string	output string
Q<[A, C]> R	Q[A, C] R
[A,<[A, X]>]	A[A, X]A
[B,<[A, X]>]	BAX [A, X] XAB
Q<[>R<]>	Q[R]
Q<<[A, C]>>R	Q<[A, C]>R

The use of string quotes makes it possible to include any symbol in the input stream except an unmatched opening or closing string quote.

11.5 Primitive Macros

A primitive macro is any macro already defined as a part of GPM. The primitive macros are: DEF, REDEF, VAL, RD, and RS. The only difference between a primitive macro and a user-defined macro is that the primitives cannot be deleted or redefined by the user. Use of the primitive macros will be explained in the following paragraphs.

11.5.1 Defining a Macro

A macro is defined by use of the primitive macro DEF which requires two parameters. The first parameter becomes the name of the new macro, and the second parameter becomes the associated symbol string or "value" of the new macro. For example, the macros A, B, and APA used above would be defined by the following macro calls:

```
[DEF, A,<A#1A>]
[DEF, B,<B[A, X#1X] B>]
[DEF, APA, < P#1#1P>]
```

11.5.2 Redefining a Macro

A macro may be redefined by use of the REDEF primitive. Use of REDEF is identical to DEF, with the following restrictions:

1) The macro must have been previously defined, and

2) The new definition must not have more characters than the original definition.

For example, considering the above definitions, the following redefinition would be legal:

```
[REDEF, A, <BC#3>]
```

but the following would be illegal:

```
[REDEF, A, < BC#3D >]
```

11.5.3 The Value of a Macro

A macro's definition may be obtained without evaluating the macro by use of the VAL primitive. For example, considering the definition of macro B above:

```
[VAL, B]
```

would produce the output string:

```
B [A, X#1X] B
```

11.5.4 String Input

In some cases it is desirable to accept input from the keyboard or tape reader to be used as a parameter in a macro call rather than to be a complete macro call. The RD and RS primitives have been added by Educational Data Systems to facilitate this requirement. The form of the "read" macro call is:

```
[RD, n, t]
```

where n is a number representing the maximum number of characters to be accepted as input, and t is the input terminating character. A typical call for input might be:

```
[RD, 8, .]
```

This call will accept eight characters as input unless a period is encountered first. The character string accepted as input becomes the value of the macro call, and this value goes into the output symbol string. The terminating character is not taken as part of the input. The input string is not scanned as it is entered; therefore, active characters may be entered as part of the string.

The RS primitive is similar to RD except that the input string is scanned. Any macro calls contained in the input string will be evaluated as the string is entered. Active characters such as commas or # symobls will cause an error if not contained in a macro call.

11.6 Errors

When an error is detected in the input stream it is indicated by a typeout such as:

```
ERROR #x
```

where x is a letter indicating the type of error as follows:

A	Unmatched end of macro call (right bracket)
B	Unquoted # symbol (outside of <> quotes)
C	Illegal character or value as an argument number
D	Not enough arguments supplied in call
E	Terminator in impossible place, probably missing right bracket
F	Stack overflow (symbol stream too long)
G	Macro not defined
H	Unmatched closing string quote
I	REDEF argument longer than original definition
J	Not used
K	Miscellaneous error other than those listed above

After an error, all macro definitions that were completed before the error was detected will still be in effect. However, any partially scanned call will be lost.

11.7 Local Definitions

A macro that is defined within another macro call is a temporary or "local" definition, and it will exist only until the macro containing its definition has been fully evaluated. A local definition is useful for evaluations of the form:

f(. . .) where f (x) = ...

An elegant example of its use is the successor function defined by

```
[DEF, SF<[1,2,3,4,5,6,7,8,9,10,[DEF, 1<#> #1]]>]
```

The effect of a call such as [SF, 3] is to make a temporary definition of a macro whose name is 1 and whose value string is #3, and then immediately to call it with arguments 2, 3, . . . 10. The third argument, 4, is the result and in fact [SF, n] = n+1 for n = 0, 1, . . . , 9 so that SF produces the successor of any digit. The temporary macro has the name 1 so that [SF, 0] will give the correct result. The definition

```
[DEF,SF2,<[#2,[DEF, #2,#1<, [SF,> #2<]>][DEF, 9,<[SF,> #1<], 0>]]>]
```

uses the SF macro to give the successor to a two digit number. Thus, [SF2, a, b,] first defines b as the string a, [SF, b] and defines 9 as the string [SF, a], 0 then evaluates [b] . If b ≠ 9 the result is a, b+1 but if b = 9 the result is a+1, 0.

11.8 Programming Examples

A simple example which demonstrates the use of formal parameters is

```
[DEF, GO, < THE #1 WENT INTO THE #2.>]
```

followed by calls such as

```
[GO, DOG, HOUSE]
[GO, CAR, DITCH]
[GO, MAN AND HIS SON, HARDWARE STORE]
```

RETURNS, LINE FEEDS, spaces, etc. may be included in a definition at any point to make the print out more readable.

A "program can be implemented in GPM by defining a primary macro which calls other (secondary) macros and simplifies their use by itself providing some of the input stream for the secondary macros. For example:

```
[DEF, SUCCESSOR,<[SF2,[RD,1,/],[RD,1,/]][SUCCESSOR] >]
```

defines a SUCCESSOR macro which, when called by

```
[SUCCESSOR]
```

will allow type in of a two digit number, will call the SF2 macro and provide the comma separator required between the two digits, thus causing the successor of the number entered to be printed. SUCCESSOR then calls itself to allow another number to be entered. Note the space in the definition just before SUCCESSOR calls itself. Since this space is part of the definition, it will be printed after SF2 is called, thus separating the typeouts for easier reading.

11.9 Using GPM

GPM is a stand-alone program which, therefore, may be used only under control of EXECUTE (see "How to EXECUTE a Program" in this manual) or in stand-alone mode (see "How to Use System in Stand-Alone Mode" in the IRIS Manager's Reference Manual).

GPM's initial starting address is location 2, and it may be restarted at this location at any time to clear all definitions and completely initialize the stack. GPM may also be restarted at location 3 which will initialize operation and clear any partial calls or evaluations but will leave completed definitions intact.

The ESC key will cause a carriage return without entering any characters into the input symbol stream. Also, CTRL A may be used to abort a partially entered macro call in case of a typing error. A single character backspace is not available since all input characters are immediately scanned and the result is put into the output stream. Use of CTRL A is identical to restarting at location 3.

In summary the active keys are:

[	Begin macro call
]	End macro call
<	Begin string quote
>	End string quote
,	Argument separator
#	Parameter number
ESC	Carriage return
CTRL A	Delete incomplete call

The RUBOUT and NULL characters are ignored. Any other character may be used as part of a string.

APPENDIX 1 : BASIC Error Numbers

1	Syntax error
2	Illegal string operation
3	Storage overflow
4	Format error
5	Illegal character
6	No such line number
7	Renumber aborted by ESC, program was lost
8	Too many variable names (limit is 93)
9	Unrecognizable word
10	Line number "RUN" is illegal before an initial RUN
11	Incorrect parentheses closure
12	Program is list/copy protected
13	Number too large (9.9999999999999E+62 is maximum)
14	Out of data
15	Arithmetic overflow (see note)
16	GOSUBs nested too deep
17	RETURN without GOSUB
18	FOR - NEXT loops nested too deep
19	FOR without matching NEXT
20	NEXT without matching FOR
21	Expression too complex (too much function nesting)
22	Not your file, can't replace
23	Array size exceeds initial dimensions
24	Only one dimension allowed for a string
25	
26	String not dimensioned
27	Syntax error in user-defined function
28	Subscript, channel number, or signal parameter out of range
29	Illegal function usage
30	User function not defined
31	User functions nested too deep
32	Matrices have different dimensions
33	Argument is not a matrix
34	Dimensions are not compatible
35	Matrix is not "square"
36	CALLed subroutine not in storage
37	Expression in argument for CALL
38	Error detected by CALLed subroutine
39	Formatted output exceeded buffer size

Note: Arithmetic overflow includes division by zero, square root of a negative value, log of zero or a negative value, storing a value greater than ±7999 in an integer variable, etc.

40	Channel already open
41	Bad Filename
42	No such file
43	File being deleted, replaced, or built
44	Not a data file, can't be opened or replaced
45	File is read protected
46	File is write protected
47	Disc is full, can't build a file or add records
48	User's disc block allotment used up
49	Channel not open
50	File not formatted
51	Illegal record number
52	Record not written
53	Illegal item number
54	Item types don't match
55	Statement is illegal from keyboard
56	Can't dump an empty program
57	Strings cannot be redimensioned
58	Error in format string
59	RUNMAT processor not in system
60	Too many numbers entered for INPUT
61	Matrices have different number types
62	Signal buffer is full
63	Commands are illegal in LOAD mode
64	Line number missing in LOAD mode

98	Illegal entry for INPUT	} These errors can occur only if error branching is in effect.
99	User pressed ESC or CTRL C	

APPENDIX 2 : GLOSSARY

IRIS - Interactive Real-time Information System. This is the name of the system as a whole, including TEX, the accounting and security system, and the system command processors.

REX - Time-sharing EXecutive. TEX consists primarily of software which resides in approximately the first 4000 core locations. Page zero contains system flags and counters, many widely used constants, and pointers to subroutines in TEX that may be used by processors. The INDEX, DISCSUBS, and ACCOUNTS files, which may be considered as parts of TEX, reside on the disc.

PROCESSOR - A machine language routine which operates under control of TEX and which may be called for execution directly from any interactive terminal. Examples of processors are: BASIC, SAVE, KILL, LIBR, CHANGE, and QUERY.

STAND-ALONE PROGRAM - A machine language routine which takes over full control of the system and does not operate under TEX. Stand-alone programs are loaded into core by use of the SHUTDOWN command. Examples of stand-alone programs are GPM and the computer diagnostic routines.

PROGRAM - A higher-level language routine which is executed by being either interpreted or compiled by a processor. For example, any routine written in BASIC is called a program. Since a program is not in machine code it is treated by TEX as a data file to be operated on by a processor.

ASCII - American Standard Code for Information Interchange. A seven-bit code used for data transfers in and out of the computer, usually with even parity, i.e., the eighth bit is set to a "one" or "zero" such that the number of "one" bits in the byte is even. When ASCII is used internally to the IRIS system, the eighth bit is unconditionally set to a "one". See Appendix 6 for a table of all 128 ASCII codes.

PORT - An interactive input/output channel on the IRIS system. For in-house operation, each port is usually hard-wired to one terminal. For remote operation, each port is connected to a data set, thereby allowing several terminals to use the same port (one at a time). There is an input/output buffer, a data file table, and an active file associated with each port.

BIT - One binary digit. May contain a one or a zero.

BYTE - Eight bits. One byte may contain an ASCII code or a special internal code.

ACCOUNT ID - A word or group of words and symbols up to 12 characters total which must be typed in by a user to activate his terminal and log him on to the system. There are over 4 x 10^{21} possible account IDs.

WORD - Sixteen bits. The computer transfers information one word at a time. One word may contain one machine instruction or two bytes. Two or more words are required to store a floating point number.

STRING - A record consisting of ASCII codes stored one code per byte. A string is usually terminated by a zero byte.

FILE - A storage area on the disc or magnetic tape which may be accessed from a processor or, via a processor, by a program. A file consists of a file header block, from zero to 128 header extender blocks, and from zero to 65534 data blocks.

FILE HEADER - The first block of a file. The header does not contain any user's data. What the header does contain is the Filename, related information such as file type and protection, and the disc address of each data block in the file (data block disc addresses are not listed in the header of a contiguous file). A data file header may also contain a data record format map.

FILENAME - The word Filename is used to represent the name (and optional password) assigned to a file by a user at the time he creates the file. If a password is used, it is separated from the name by a CTRL E character (written $\underline{E}$). The name and password may be any combination of letters, digits, and periods, except that the first character of the name must be a letter. Acceptable Filenames include:

NAME$\underline{E}$PASSWORD
XY14Q99
TE543382XB.5

The password is not echoed (not printed) during type-in. This is to prevent other users on your account from learning your passwords and gaining access to files you wish to keep private. The $\underline{E}$ must be typed again after the password to resume normal echo operation. Up to fourteen characters total are allowed including the $\underline{E}$ code, making a total of over 6 x 10^{21} possible Filenames.

The same Filename may be used on different Logical Units; therefore, any Filename given by a user is assumed to be on that user's assigned Logical Unit unless specified by entering the Filename in the form

lu/Filename

where lu is the number of the Logical Unit where Filename is to be found or built. The same Filename may be used to identify one file on each Logical Un it. Any time a new file is being built, the Filename may be given in the form

$ddd.cc <pp> lu/Filename!

where ddd.cc is the amount to be charged to the account of any other user who accesses the file, and pp is the desired protection (see Section 2.5). The cost and protection may be given in either order and both are optional, but the Filename must be given last. If not specified, the cost will be zero and the protection will be 77. An exclamation mark following the Filename allows this file to replace another file of the same type and on the user's own account, in chich case the old file's cost and protection will be used if not specified here.

For a peripheral driver file there is a special form of Filename consisting of a dollar sign and the device's mnemonic; for example: $PTR is the paper tape reader, and $LPT is the line printer. Dollar sign Filenames are also used for system subroutine replacements. The first character after the dollar sign in such a Filename must be a letter.

PRIVATE FILE - A file which has a password included in the Filename so as to prevent access to the file by other users on the same account and on lower privilege accounts.

PUBLIC FILE - A file with no password (no CTRL E) included in the Filename.

ACTIVE FILE - There is an active file associated with each interactive port. All programs and data entered through the port are placed in its active file. The active file does not have a Filename unless it has been saved. The active file is active in the sense that it is used for swapping; i.e., it is used to store a user's program and local data between time slices.

PASSIVE FILE - All user files stored on the disc under a Filename are passive files. A passive file may be copied into a port's active file for processing without disturbing the passive file. The SAVE command updates a passive file or creates a new passive file by copying the active file into the passive file.

DISC ADDRESS - A disc address is a means of identifying a particular disc block on a given Logical Unit. A disc address may be in any of three forms as follows:

LOGICAL ADDRESS - In this form the Logical Cylinder, Logical Track, and Logical Sector are carried as binary integers in separate words (see definitions of these terms elsewhere in the Glossary). This form is generally used only in the system's allocate and deallocate routines.

REAL ADDRESS - The form carried within the system in file headers, etc. In this form the address is packed into one binary word according to the formula

$$LC \times LRC + LT \times LRT + LS$$

where	LC	=	Logical Cylinder
	LT	=	Logical Track
	LS	=	Logical Sector
	LRC	=	Logical-to-Real Cylinder Conversion Factor
	LRT	=	Logical-to-Real Track Conversion Factor

It is not necessary to define a logical-to-real sector conversion factor since it would always equal one.

PHYSICAL ADDRESS - The address used by the disc controller. The disc driver subroutine converts the Logical Unit number and the Real Address into a Physical Address and outputs it to the disc controller. For some controllers, the Physical Address is identical to the Real Address; otherwise, the Physical Address is never stored within the system.

PHYSICAL UNIT - A secondary data storage unit such as a disc cartridge or a reel of magnetic tape. Data are transferred to and from the Physical Unit in blocks of 256 words.

LOGICAL UNIT - A Physical Unit (or one partition of a Physical Unit) which has been formatted for use by IRIS. Usually an entire Physical Unit is treated as one Logical Unit, but provision is made for partitioning a Physical Unit into two or more units; this may be necessary in the case of a very large disc pack where the Real Address would otherwise require more than one word. A Logical Unit may not include more than one Physical Unit; e.g., the fixed disc and the removable cartridge on a dual drive must be treated as two (or more) Logical Units. Each Logical Unit has a copy of BZUP in Real Address zero, an INDEX whose header is in Real Address one, an ACCOUNTS file whose header is at Real Address three, and a disc map (DMAP) whose header is at Logical Cylinder zero, Logical Track one, Logical Sector zero (Real Address LRT).

CYLINDER - A set of tracks on one disc drive that can be accessed without moving the heads. The term is derived from the high stack disc packs where the set of tracks is a set of equal sized circles around the same axis, one on each disc surface. Such circles are all on the surface of the same geometric cylinder. Note: A head-per-track disc is considered as having one cylinder (number zero), whereas the cylinder number specifies the head position on a moving arm disc.

TRACK - The path traced on a disc surface by one head in one position.

SECTOR - A fraction of one revolution of a disc. May be thought of as a pie shaped section of the disc. Within this amount of rotation, one block of data may be recorded in each track.

BLOCK - The intersection of one track with one sector. Each block will store 256 16-bit words of data. Formatted files on magnetic tape are also written in blocks of 256 words.

LOGICAL CYLINDER - An integer ranging from zero to one less than the number of cylinders in a Logical Unit. The Logical Cylinder is equal to the real cylinder unless the Physical Unit is partitioned into two or more Logical Units, in which case the physical cylinder number is derived by adding the Logical Cylinder to the number of the first real cylinder.

LOGICAL TRACK - An integer ranging from zero to one less than the number of heads on a disc drive. The Logical Track number selects one head (track) for reading or writing.

LOGICAL SECTOR - An integer ranging from zero to one less than the number of sectors on a disc. The Logical Sector selects the sector time during which a transfer will take place, thereby selecting one block within the track specified by the Logical Cylinder and Logical Track.

IPL - The first phase of the IRIS startup procedure is the Initial Program Load which brings a fresh copy of REX into core from the system disc.

SYSGEN - A System Generation is the process of writing the operating system and all other system files on the system disc (Logical Unit zero). All files currently on the system disc will be lost if a SYSGEN is performed, but user files may be saved by first copying them to another Logical Unit or to paper tape.

APPENDIX 3 : ACTIVE CHARACTERS

Several keys on the terminal keyboard have significance to the operating system. Most of these require that the control key (CTRL on keyboard) be pressed simultaneously with another key. For example, CTRL A requires pressing the CTRL key and the A key at the same time. For easier writing, CTRL A is sometimes abbreviated $\underline{A}$. Most of these codes are intercepted by TEX and will never reach a processor or the user's program.

Key	Meaning	Operation
CTRL A	Same as CTRL H on most systems.	
CTRL B	Signal	Causes signal to be sent to the user's own port. Both signal values will be zero. May be used to notify a user's program that the user wishes input to be enabled if the program is checking for signals.
CTRL C	System Escape	The computer prints a # symbol and enters control model. A system command (SAVE, LIBR, etc.) may be given or a processor name may be entered. If terminal was not in type-in mode, the first $\underline{C}$ acts as an ordinary ESC, and a second $\underline{C}$ is necessary to perform the system escape.
CTRL E	Echo Toggle	Characters typed in following the $\underline{E}$ will not be echoed (printed). Pressing $\underline{E}$ a second time restores echo. Echo is suppressed for type-in of passwords and may be used at other times as desired. The first $\underline{E}$ is printed as a colon.
CTRL G	Bell	May be included in a printed string to ring the terminal's bell.
CTRL H	Backspace	Delete the last character entered. May be used repeatedly to delete several characters. Causes the character being deleted to be printed.
CTRL J	Same as LINE FEED	
CTRL M	Same as RETURN	
CTRL O	Kill Output	Causes termination of any type-out currently in progress but allows next type-out to be started. May be used to save time by suppressing type-out of known messages.

CTRL P	Parity Toggle	Causes suppression of parity checking on characters received by the computer from your terminal. Should only be used if your terminal does not generate even parity or if you wish to load a tape produced on another system without even parity. A second P restores parity checking. You can test whether parity is being checked by pressing the CTRL and LINE FEED keys simultaneously, thus sending a line feed code with bad parity. If the bell rings, parity is being checked. P causes a % symbol to be printed.
CTRL Q	XON	A CTRL Q typed by the user will be ignored. An XON code is sent by the system to start the terminal's tape reader each time input is enabled.
CTRL S	XOFF	A CTRL S typed by the user will be ignored. An XOFF code is sent by the system to stop the terminal's tape reader when input is terminated by a RETURN.
CTRL X	Cancel Input	Deletes the entire line typed in preceding the X. A\ is printed, and a carriage return is performed. Type-in remains enabled.
CTRL Z	The system simply passes CTRL Z codes on to the processors. However, some processors (including BASIC and DSP) recognize a CTRL Z as an indication that a RETURN code should be inserted in the string being entered.	
ESC	Escape	Terminates any current type-out and/or computation and restores terminal to type-in mode. Used to "escape" from an undesired or unknown situation. ESC is also used to activate a terminal when signing on.
ALT MODE	Same as ESC.	
RETURN	End Input	Each type-in must be terminated by a carriage return. An XOFF code is echoed to stop the tape reader.
LINE FEED	Ignored.	
RUBOUT	Ignored.	
BREAK	Same as CTRL B.	
HERE IS	Null	On most terminals, the HERE IS key will transmit twenty null codes (all zero bytes). Nulls are ignored by IRIS but are echoed. May be used to punch leader on the terminal's paper tape punch. If the HERE IS key on your terminal does not generate nulls, you may use the CTRL, SHIFT, REPT and P keys simultaneously to punch leader. The REPT (repeat) and P keys must be released first to avoid generating garbage codes.

APPENDIX 4: HOLLERITH AND DRATSAB CODES

The table on the next page gives the Hollerith card codes and the equivalent Dratsab codes for the standard set of 64 ASCII symbols. Most of these Hollerith codes can be produced on a standard keypunch with a single key stroke. The Dratsab code (sometimes called compressed Hollerith) is derived from the Hollerith card code as follows:

1) The top five bits of the Dratsab byte are taken from rows 8, 9, 12, 11, and 0 of the card column in that order. A punch on the card becomes a one in the corresponding bit of the Dratsab code. Any combination of the punches in these five rows is permissable for conversion to Dratsab.

2) The octal value of the punch in rows one through seven becomes the lower three bits of the Dratsab code. Only one of these seven rows may be punched. The lower three bits will be zero if there is no punch in rows one through seven.

Pictorially, each Dratsab byte is made up from the corresponding card row as shown here:

8	9	12	11	0	1-7

Any card reader which is to communicate with a standard IRIS driver over an RS-232C interface must compress the Hollerith codes in this manner. A special driver can be written, however, if necessary for a different card reader.

Hollerith and Dratsab Card Codes

Symbol	ASCII (octal)	Hollerith card punches	Dratsab (octal)	Symbol	ASCII (octal)	Hollerith card punches	Dratsab (octal)
space	240	none	000	@	300	8-4	204
!	241	8-11-2	222	A	301	12-1	041
"	242	8-7	207	B	302	12-2	042
#	243	8-3	203	C	303	12-3	043
$	244	8-11-3	223	D	304	12-4	044
%	245	8-0-4	214	E	305	12-5	045
+	246	8-12-6	246	F	306	12-6	046
'	247	8-5	205	G	307	12-7	047
(	250	8-12-5	245	H	310	12-8	240
)	251	8-11-5	225	I	311	12-9	140
*	252	8-11-4	224	J	312	11-1	021
&	253	12	040	K	313	11-2	022
,	254	8-0-3	213	L	314	11-3	023
-	255	11	020	M	315	11-4	024
.	256	8-12-3	243	N	316	11-5	025
/	257	0-1	011	O	317	11-6	026
0	260	0	010	P	320	11-7	027
1	261	1	001	Q	321	11-8	220
2	262	2	002	R	322	11-9	120
3	263	3	003	S	323	0-2	012
4	264	4	004	T	324	0-3	013
5	265	5	005	U	325	0-4	014
6	266	6	006	V	326	0-5	015
7	267	7	007	W	327	0-6	016
8	270	8	200	X	330	0-7	017
9	271	9	100	Y	331	0-8	210
:	272	8-2	202	Z	332	0-9	110
;	273	8-11-6	226	[	333	8-12-7	247
<	274	8-12-4	244	\	334	8-11-7	227
=	275	8-6	206	]	335	8-0-2	212
>	276	8-0-6	216	↑	336	8-12-2	242
?	277	8-0-7	217	←	337	8-0-5	215

APPENDIX 5: EDS BASIC CARD PROGRAMMER

A card reader on an IRIS system may be used to load a BASIC program by following the procedure in Section 2. 10. The cards may be punched in standard Hollerith card codes (see Appendix 5) or in EDS BASIC card codes by use of the EDS BASIC Card Programmer on page A6-4. The Card Programmer is used in the following manner:

1) Write the statement to be coded along the top edge of the card.

2) Mark the line number in columns 1-4.

3) Mark one statement type box in column 5 or 6.

4) Insert the card into the Card Programmer so that column 7 is just exposed at the edge of the Programmer.

5) Locate the symbol to be coded, and mark the box directly to its left on the card. Also mark the top three rows of the card as shown directly above the symbol.

6) Slide the card out one column at a time and mark each column as in step 5.

7) Mark rows 8 and 9 in the first unused column of the card. This is equivalent to a RETURN code.

If the statement is too long for one card, omit the 8-9 mark. Mark the CONTINUE box in column one of a second card, and continue the statement starting in column 7 of that card. The statement may not exceed two cards.

Spell out any words not shown on the Card Programmer. If the statement type desired is not shown, leave columns five and six blank, and spell out the statement type starting in column seven. Mark firmly, using a medium soft (number 2) pencil.

EDS BASIC Mark/Sense Card Format

Symbol	Rows	Symbol	Rows	Symbol	Rows
0	0	?	0, 1	Space	None
1	1	+	12, 11, 1	LOG	12
2	2	-	12, 11, 2	EXP	11
3	3	*	12, 11, 3	SIN	12, 11
4	4	/	12, 11, 4	COS	12, 0
5	5	↑	12, 11, 5	TAN	11, 0
6	6	(	12, 11, 6	ATN	12, 11, 0
7	7	)	12, 11, 7	SQR	12, 11, 0 1
8	8	[	12, 11, 8	INT	12, 11, 0 2
9	9	]	12, 11, 9	SGN	12, 11, 0 3
A	12, 1	=	12, 0, 1	RND	12, 11, 0 4
B	12, 2	<	12, 0, 2	ABS	12, 11, 0 5
C	12, 3	>	12, 0, 3	TAB	12, 11, 0 6
D	12, 4	≤	12, 0, 4	TO	12, 11, 0 7
E	12, 5	≥	12, 0, 5	STEP	12, 11, 0 8
F	12, 6	@	12, 0, 6	GOTO	12, 11, 0 9
G	12, 7	$	12, 0, 7		
H	12, 8	#	12, 0, 8	'	11, 0, 5
I	12, 9	%	12, 0, 9	"	11, 0, 6
J	11, 1	.	11, 0, 1	!	11, 0, 7
K	11, 2	,	11, 0, 2	←	11, 0, 8
L	11, 3	:	11, 0, 3	&	11, 0, 9
M	11, 4	;	11, 0, 4		
N	11, 5				
O	11, 6				
P	11, 7				
Q	11, 8				
R	11, 9				
S	0, 2				
T	0, 3				
U	0, 4				
V	0, 5				
W	0, 6				
X	0, 7				
Y	0, 8				
Z	0, 9				

For any symbol or word in this chart, mark the rows shown in the table. The top row of the card is row 12; the second row, 11; the third, 0; the fourth, 1, etc. Words not in the chart (FN, INV, etc.) may be spelled out using one column for each letter. After completing the statement, mark both rows 8 and 9 in the next column to indicate end of statement.

Use GOTO in place of THEN in IF statements. If a statement is too long for one card, mark the CONTINUE box on the next card and continue the statement. Mark the end of statement (rows 8 and 9) only on the second card. A statement may not use more than two cards.

EDS BASIC Card Codes

Symbol	ASCII (octal)	EDS BASIC Card Marks	Dratsab (octal)
space	240	none	000
!	241	11-0-7	037
"	242	11-0-6	036
#	243	12-0-8	250
$	244	12-0-7	057
%	245	12-0-9	150
&	246	11-0-9	130
'	247	11-0-5	035
(	250	12-11-6	066
)	251	12-11-7	067
*	252	12-11-3	063
+	253	12-11-1	061
,	254	11-0-2	032
-	255	12-11-2	062
.	256	11-0-1	031
/	257	12-11-4	064
0	260	0	010
1	261	1	001
2	262	2	002
3	263	3	003
4	264	4	004
5	265	5	005
6	266	6	006
7	267	7	007
8	270	8	200
9	271	9	100
:	272	11-0-3	033
;	273	11-0-4	034
<	274	12-0-2	052
=	275	12-0-1	051
>	276	12-0-3	053
?	277	0-1	011
@	300	12-0-6	056
A	301	12-1	041
B	302	12-2	042
C	303	12-3	043
D	304	12-4	044
E	305	12-5	045
F	306	12-6	046
G	307	12-7	047
H	310	12-8	240
I	311	12-9	140
J	312	11-1	021
K	313	11-2	022
L	314	11-3	023
M	315	11-4	024
N	316	11-5	025
O	317	11-6	026
P	320	11-7	027
Q	321	11-8	220
R	322	11-9	120
S	323	0-2	012
T	324	0-3	013
U	325	0-4	014
V	326	0-5	015
W	327	0-6	016
X	330	0-7	017
Y	331	0-8	210
Z	332	0-9	110
[	333	12-11-8	260
\	334	see text	
]	335	12-11-9	160
↑	336	12-11-5	065
←	337	11-0-8	230

(continued on next page)

EDS BASIC Card Codes (Continued)

Symbol	Internal (octal)	EDS BASIC Card Marks	Dratsab (octal)
<=	356	12-0-4	054
>=	354	12-0-5	055
LOG	145	12	040
EXP	146	11	020
SIN	147	12-11	060
COS	150	12-0	050
TAN	152	11-0	030
ATN	151	12-11-0	070
SQR	144	12-11-0-1	071
INT	140	12-11-0-2	072
SGN	142	12-11-0-3	073
RND	143	12-11-0-4	074
ABS	141	12-11-0-5	075
TAB	345	12-11-0-6	076
TO	344	12-11-0-7	077
STEP	346	12-11-0-8	270
GOTO	100	12-11-0-9	170

BASIC Mark/Sense Card Programmer

Space	LOG	EXP	▩	SIN	COS	TAN	ATN
	☒	☐	☐	☒	☒	☐	☒
	☐	☒	☐	☒	☐	☒	☒
0	☐	☐	☒	☐	☒	☒	☒
1	A	J	?	+	=	.	SQR
2	B	K	S	−	<	,	INT
3	C	L	T	•	>	:	SGN
4	D	M	U	/	≤	;	RND
5	E	N	V	↑	≥	'	ABS
6	F	Ø	W	(	@	"	TAB
7	G	P	X	)	$	!	TO
8	H	Q	Y	[	#	←	STEP
9	I	R	Z	]	%	&	GOTO

Mark only in top three rows for these functions. (No marks for space)

Educational Data Systems
Newport Beach, California

For any symbol or word in this chart, mark the row in which it appears, and mark the top three rows as shown directly above the symbol or word. Words not in the chart (FN, INV, etc.) may be spelled out using one column for each letter. After completing the statement, mark both rows 8 and 9 in the next column to indicate end of statement.

APPENDIX 6: ASCII CODES

code	name	keys	code	name	keys	code	name	keys	code	name
000	NUL	csP	240	space		300	@	sP	140	
201	SOM	cA	041	!	s1	101	A		341	a
202	EOA	cB	042	"	s2	102	B		342	b
003	EOM	cC	243	#	s3	303	C		143	c
204	EOT	cD	044	$	s4	104	D		344	d
005	WRU	cE	245	%	s5	305	E		145	e
006	RU	cF	246	&	s6	306	F		146	f
207	BELL	cG	047	'	s7	107	G		347	g
210	BS	cH	050	(	s8	110	H		350	h
011	TAB	cI	251	)	s9	311	I		151	i
012	LF	cJ	252	*	s:	312	J		152	j
213	VT	cK	053	+	s;	113	K		353	k
014	FORM	cL	254	,		314	L		154	l
215	RET	cM	055	-		115	M		355	m
216	SO	cN	056	.		116	N		356	n
017	SI	cO	257	/		317	O		157	o
220	DLE	cP	060	0		120	P		360	p
021	XON	cQ	261	1		321	Q		161	q
022	TAPE	cR	262	2		322	R		162	r
223	XOFF	cS	063	3		123	S		363	s
024	~~TAPE~~	cT	264	4		324	T		164	t
225	ERR	cU	065	5		125	U		365	u
226	SYN	cV	066	6		126	V		366	v
027	LEM	cW	267	7		327	W		167	w
030	CAN	cX	270	8		330	X		170	x
231	EM	cY	071	9		131	Y		371	y
232	SUB	cZ	072	:		132	Z		372	z
033	ESC	csK	273	;		333	[	sK	173	{
234	FS	csL	074	<	s,	134	\	sL	374	|
035	GS	csM	275	=	s-	335	]	sM	175	}
036	RS	csM	276	>	s.	336	↑	sN	176	~
237	US	csO	077	?	s/	137	←	sO	377	DEL

Notes: sP means that shift P will generate an octal 300 code (@), csP means that control shift P will generate an octal 200 code (NUL), and cP means that control P will generate an octal 220 code (DLE), etc.

The LF, RET, and ESC codes are duplicated by special keys. DEL is generated by the RUBOUT key on most terminals.

All codes are shown in octal with even parity as they are used for transmission and as they appear on a list tape. Internal to the system, all ASCII codes carry a one in the top bit instead of parity; e.g., the internal code for A is 301 octal. Add 200 to any code whose first digit is zero or one to get the internal code.