edusystem20

reference manual

digital

digital equipment corporation

Roland Ross SAIT
Computer Centre
Room CC2-8

DEC-08-E20RA-A-D

edusystem20
reference manual

digital equipment corporation

First Printing, July 1975

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance to the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Digital.

Copyright © 1975 by Digital Equipment Corporation

The HOW TO OBTAIN SOFTWARE INFORMATION pages, located at the back of this document, explain the various services available to Digital software users.

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM		

LIMITED RIGHTS LEGEND

Contract No. _____

Contractor or Subcontractor: Digital Equipment Corporation

All the material contained herein is considered limited rights data under such contract.

CONTENTS

FIGURES

TABLES

PREFACE

This manual is provided to fill the needs of three types of users:

a. Students
b. Advanced Programmers
c. System Managers

Students can learn fundamentals of programming the Edusystem 25 in the BASIC language by reading Chapters 1 and 2. Examples which illustrate the use of a BASIC statement often require an understanding of other statements. For this reason, each section in these chapters contains information based on previous sections. Students should be encouraged to learn new material and to reinforce existing knowledge at the same time.

For students, this manual is most effective when terminals are available. Experimental programs should be tested and run as new topics are learned. Access to the computer is the most important single factor in learning how to program.

Advanced programmers can use this manual as a reference source. Each BASIC statement is explained and syntax requirements are provided in each section and in Appendix B for easy reference.

Chapter 3 contains information on initializing the PDP-8/E computer and instructions for use of optional system programs. Most of this material is directed to the system manager.

The following conventions are observed in this manual:

_____	Where clarification is required in the examples used in this manual, underlined copy denotes information printed by the system on the terminal; copy not underlined indicates user-typed entries.
↵	This symbol is the notation for typing the carriage return (CR or RETURN) key on the terminal.
{ }	Braces indicate optional elements in a statement syntax.
GOTO line number	Capital letters indicate the exact format of statements; small letters denote the logical content of elements in the statement. In this example, the word GOTO must appear as shown, while an actual number must be substituted for "line number".

CHAPTER 1

TALKING TO THE COMPUTER

1.1 THE TERMINAL KEYBOARD

The terminal you are using may be an alphanumeric display terminal or a teleprinter (hard copy) terminal. In most cases these terminals operate identically, so this manual uses the word terminal when referring to all of these devices.

Terminals are used to relay your instructions to the computer, telling it to carry out specific operations. The computer (or system) performs the required operations and prints the results on the same terminal. It also prints error messages on the terminal when you make certain mistakes. A list of these error messages appears in Appendix A.

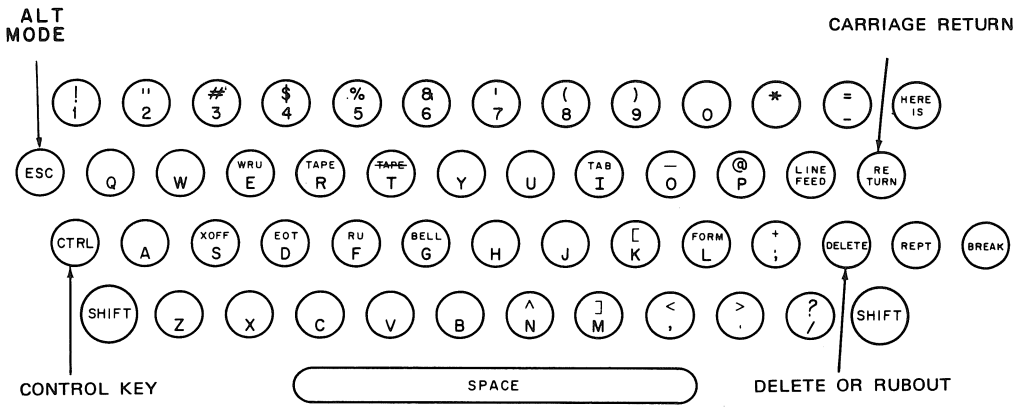
Diagrams of typical terminal keyboards are shown in Figure 1-1.

To type a character on the terminal, simply press the appropriate key. For example, to type the letter P, press the key on which P appears.

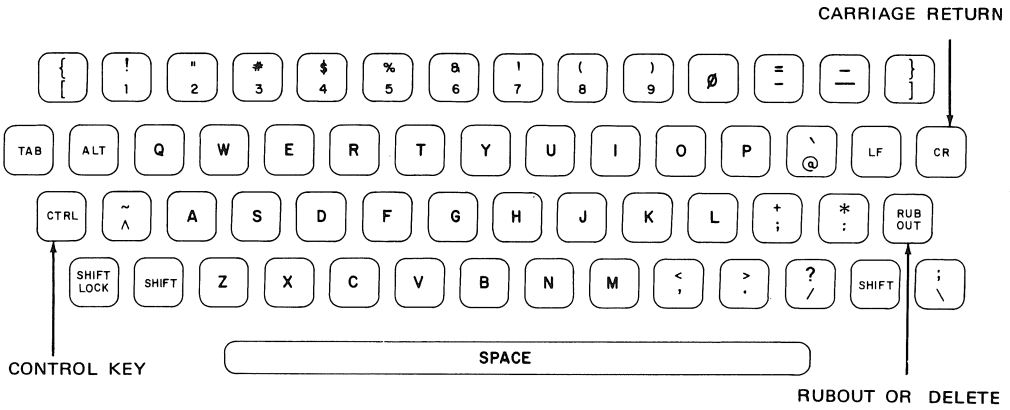
Some keys have two characters printed on them. For example, the number 3 on the top row of keys has a number sign character, #, above it. This number sign is a shifted character.

There are two SHIFT keys located at the left and right ends of the bottom row of keys. To type a shifted character, hold down either SHIFT key while typing the character.

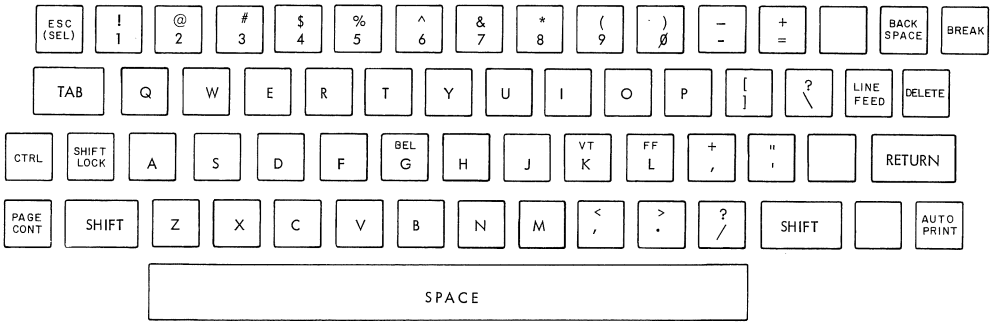
This manual represents a shifted letter with the combination: SHIFT/Character.



TELETYPE KEYBOARD



VTØ5 / LA3Ø KEYBOARD



VT5Ø / LA36 KEYBOARD

Figure 1-1
Typical Terminal Keyboards

Similarly, to type the combination CTRL/Character, hold down the CTRL (Control) key, located at the left side of the keyboard above the SHIFT key, while typing the character.

Use the space bar to insert spaces as you type. This makes your printouts and expressions easier to read.

Finally, locate the key labelled RETURN or CR (for Carriage Return). It is on the upper right side of the keyboard. This is the key that actually transmits your typed message to the computer. For this reason, it is the most important key on the keyboard and the one most frequently used.

Once you have some familiarity with the terminal keyboard, you are ready to communicate with the computer.

1.2 IMMEDIATE MODE

There is only one basic difference between an adding machine and a computer: the computer can remember. It can perform instructions you give it immediately or it can perform them at a later time.

When the computer system performs your instructions immediately (i.e., in "immediate mode"), the computer works like an adding machine. It is very fast and it has many sophisticated features adding machines don't have, but basically in immediate mode it's an adding machine.

This section describes how to use the computer in immediate mode. So for now, think of the computer as an adding machine.

1.2.1 It's an Adding Machine!

To add two numbers, type in the word PRINT, then type in the first number, a plus sign (+), and finally the second number. Now review the information you have typed on the terminal. If you've made a mistake, type the RUBOUT or DELETE key to backspace and erase the incorrect character. Each time the RUBOUT key is typed, the system erases one more character.

For example, suppose you type:

```
PRINT 4 +- 7
```

To remove the minus sign, type the RUBOUT key three times, which removes the 7, the space, and the minus sign. Now retype a space and the 7. The printed line now looks like this:

```
PRINT 4 +- 7___ 7
```

To execute the line you just entered, type the RETURN key. In computer printouts, carriage returns are denoted by↵.

For example, to add 45 and 15, type the following line:

```
PRINT 45 + 15↵
```

Since you are in immediate mode, the computer immediately prints the answer on your terminal: 60 and then READY. Note that on some terminals zeros are slashed (Ø) to differentiate them from the capital letter O. When the computer types READY, it is finished performing your instructions and is ready to accept further directions.

If you have access to a terminal while reading this manual, don't be afraid to experiment with the computer, since you can't damage it from a terminal. In fact, you can learn more by actually experimenting than you can from a manual alone.

Subtraction is accomplished in a similar manner, with the use of a minus sign (-), as shown below. The items underlined in this printout indicate the system prints them on the terminal.

```
PRINT 45 - 15
 30

READY
```

Multiplication is accomplished by using an asterisk (*):

```
PRINT 45 * 15
 675

READY
```

Division requires a slash character (/):

```
PRINT 45 / 15
 3

READY
```

To raise a number to a power, use the up-arrow (↑) or caret (^) sign. (This can be typed as SHIFT/N on some terminals. The terminal used in this manual prints a semi-circle for this character.) For example, 8 squared is:

```
PRINT 8 ^ 2
 64

READY
```

The arithmetic operators discussed above can be combined to form longer expressions. For example, consider the gravity formula in physics: $F=G(M1)(M2)/D^2$. In order for the computer to understand this formula and give a value for F, it would have to be written: G*M1*M2/D↑2. Of course, since you haven't defined a value for G, M1, M2 or D, the computer assumes the value of each of these is zero. So it can't give you an answer, it can only give you an error message.

```
PRINT G*M1*M2/D^2↵
ERROR  3
READY
```

There are 48 different errors which can occur on your computer system. Error 3, shown above, occurs when you attempt to divide by zero. The complete list of all 48 errors appears in Appendix A of this manual.

It is often convenient to express very large and very small numbers in a kind of short-hand. Programmers use scientific notation, which is a method of multiplying numbers by powers of ten. For example, 100 is really 10^2, 1000 is 10^3, etc. The number 300 can be expressed as 3×10^2. When the exponent is negative, of course, the number is a fraction or decimal. For example, .005 can be expressed as 5×10^{-3}. The BASIC language uses the letter E to replace the characters, x 10. Table 1-1 shows scientific notation examples.

Table 1-1
Scientific Notation Examples

Number	Scientific Notation	BASIC Notation
1000	10^3	1E+3
360	3.6×10^2	3.6E+2
-22000	-2.2×10^4	-2.2E+4
.00041	4.1×10^{-4}	4.1E-4
-.07	-7×10^{-2}	-7E-2

The largest number you can represent in this manner is about 1E+38 (i.e., a one followed by 38 zeroes); the smallest decimal is about 1E-38 (i.e., a decimal point followed by 38 zeroes and then a one). In both cases, negative numbers are also allowed.

Figure 1-2 shows the range of numbers allowed in the EduSystem BASIC language. Zero is also allowed. Notice that this diagram is not drawn to scale.

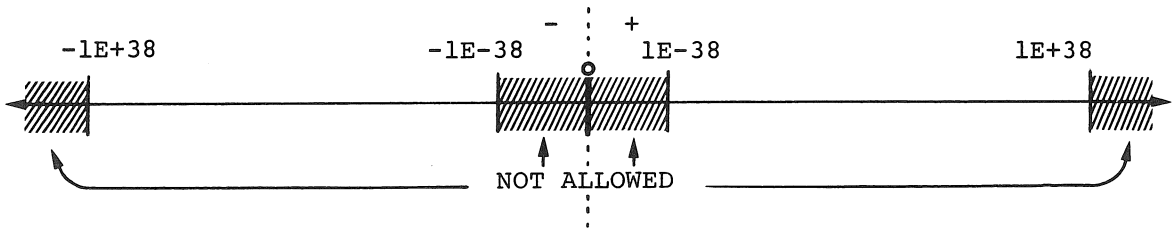


Figure 1-2
Range of Legal Numbers

1.2.1.1 Hierarchy - If more than one arithmetic operation appears on a line, the computer performs the operation with the highest priority first. The order of priority in the BASIC language is:

exponentiation	↑	
multiplication	*	(* and / have equal priority)
division	/	
addition	+	(+ and - have equal priority)
subtraction	-	

Consider this example:

```
PRINT 3 + 4 * 6↵
 27

READY
```

Since multiplication has a higher priority than addition, in the above example, 4 and 6 are multiplied together first and then 3 is added to the result.

If two operators have the same priority (e.g., multiplication and division), the computer automatically assigns precedence from left to right.

```
PRINT 3 + 4 * 6 / 2 - 9↵
 6

READY
```

Since multiplication and division have higher priorities than addition and subtraction, the above expression is evaluated as follows:

4 and 6 are multiplied together; the result (24) is divided by 2; that result (12) is added to 3; finally, that result (15) minus 9 yields the answer 6.

Parentheses can be used to change the order of precedence. The operation(s) specified within parentheses is always performed first. For example:

```
PRINT 5 * (7-3)↵
 20

READY
```

The expression within parentheses, 7 minus 3, is evaluated first; then the result (4) is multiplied by 5.

Parentheses can also be nested within other parentheses. In that case, the operation specified in the deepest pair of parentheses is performed first. For example:

```
PRINT 5 - (20 * (8 + 4))↵
-235

READY
```

In the above case, the deepest pair of parentheses contain the expression 8 + 4. This is evaluated first. Then the result (12) is multiplied by 20, since this expression is still in a pair of parentheses. Finally, that result (240) is subtracted from 5.

When nesting expressions, always be sure the total number of left parentheses equals the total number of right parentheses.

1.2.1.2 Arithmetic and Trigonometric Functions - In the BASIC language, the system recognizes certain 3-letter words as functions. Table 1-2 lists these functions and their corresponding meanings.

Table 1-2
Arithmetic and Trigonometric Functions

Function	Meaning
ABS (expression)	absolute value of the expression
ATN (expression)	arctangent of the expression; the result is in radians
COS (expression)	cosine of the expression in radians
EXP (expression)	e (2.71828...) raised to the power of the expression
FIX (expression)	integer part of expression; truncate decimal portion
INT (expression)	integer value $\leq$ value of the expression
LOG (expression)	natural logarithm (base e) of the expression
MOD (expression, (expression)	remainder of division of first expression by the second

Table 1-2 (Cont.)
Arithmetic and Trigonometric Functions

Function	Meaning
RND (expression)	random number between 0 and 1
SGN (expression)	1 if the expression is positive; -1 if the expression is negative; 0 if the expression is zero
SIN (expression)	sine of the expression in radians
SQR (expression)	square root of the expression
TAN (expression)	tangent of the expression in radians

To find the cosine of 0°, for example, type in:

```
PRINT COS(0)
 1

READY
```

1.2.1.3 Relational Operators - Often a program compares the values of two or more variables. Use the set of relational operators shown below to compare these values. Table 1-3 shows the operators and combinations of operators as well as their meanings.

Table 1-3
Relational Operators

=	equal to
>	greater then
<	less than
>=	greater than or equal to
<=	less than or equal to
<>	not equal to

Expressions which use these operators are frequently found in IF statements (see Section 1.3.9). They are evaluated simply as true or false.

1.2.1.4 Using Letters as Variables - Often in programming, you may want to use different numbers in the same formula or equation. Instead of rewriting the equation with the new numbers each time, you can substitute a variable in the equation and redefine its value when needed.

This BASIC feature is one of the most powerful in computer science. It allows you to manipulate variables without even knowing what their values are at any given time.

Assigning a variable can be accomplished simply by using an equal sign (=). For example,

```
X = 5↵
Y = 1.2↵
Z = 22E-4↵
```

Any letter can be a variable in an assignment statement. In addition, any letter followed by a single digit can be a variable. In this way, 11 variables can be formed for every letter in the alphabet (e.g., B, BØ, B1, B2,...,B9). You can use up to 286 variables. If you need more than that, see Section 1.3.14, Subscripts and Arrays.

Working with variables instead of constants allows a program to be written and executed faster.

1.2.2 It's a Typewriter!

You have already seen how the system prints answers to arithmetic expressions automatically when you use a PRINT statement in immediate mode. Actually, PRINT statements are far more versatile.

Multiple expressions can be evaluated and printed on one line with the use of commas.

For example:

```
PRINT 1,2,3
 1             2             3

READY
PRINT 2,2^2,2^3,2^4
 2             4             8             16

READY
```

Semicolons in the PRINT statement between expressions reduce the space between printed values. For example:

```
PRINT 3;3^2;3^3;3^4
 3  9  27  81

READY
```

In immediate mode, the computer can also act as a typewriter. It can print alphanumeric messages with the use of the PRINT statement. Quotation marks (") must be used before and after the message to be printed. For example:

```
PRINT "WHAT AM I DOING HERE?"
WHAT AM I DOING HERE?

READY
```

Everything within a set of quotation marks is called a literal. A literal is printed exactly as it appears, regardless of spelling errors, syntax inconsistencies or logic mistakes. This is a good example of how a computer does exactly what it is instructed to do... no more, no less. Remember: the computer does what you tell it to do, not what you thought you told it to do.

Literals and numeric data can be intermixed. For example:

```
PRINT "THE NUMBER OF DAYS IN A WEEK IS" 7↵
THE NUMBER OF DAYS IN A WEEK IS 7

READY

PRINT "DOES 30 + 9 EQUAL" 40+9 "?"↵
DOES 30 + 9 EQUAL 49 ?

READY

PRINT "THERE ARE" 60 * 24 "MINUTES IN A DAY."↵
THERE ARE 1440 MINUTES IN A DAY.

READY
```

Remember, the system prints everything within a pair of quotation marks, character by character. In the second example, above, the arithmetic expression within the quotation marks is not evaluated, it is merely printed. The expression outside of the quotation marks, however, is evaluated automatically.

The PRINT statement is the most powerful statement in the BASIC language and the one used most often by programmers to communicate with the computer. It is used extensively in the following chapters to help illustrate other features.

1.2.2.1 Character Strings - The EduSystem can manipulate literals of six characters or less by assigning them a letter of the alphabet, followed by the dollar sign ($). So you can have up to 26 string variables defined at one time. (The number of separate string variables can be increased by using subscripts. See Section 1.3.14.) For example,

```
D$ = "FAMOUS"↵

READY
```

The string variable D$ is now defined. It contains the six characters, FAMOUS. These characters are internally numbered 1 through 6.

The length of a string can be any number from 0 to 6. To find the length of a string, use the LEN function as follows:

```
PRINT LEN(D$)↵
 6

READY
```

Portions of a string can be extracted by using the MID function:

```
PRINT MID(D$,5,2)↵
US

READY
```

The MID function prints a specified number of characters (in the above example, 2 characters) starting from a specified position (in the above example, character #5) of a specified string variable (in the above example, string variable D$).

The CAT function combines the contents of two strings as long as the total number of characters is not greater than six. For example, if two string variables are defined:

```
A$ = "DO "↵
B$ = "IT!"↵
```

a CAT function combines both strings as shown below:

```
PRINT CAT(A$,B$)↵
DO IT!

READY
```

Notice that combining two strings whose total number of characters exceeds six results in the first string plus however many characters can be used in the second string to total six:

```
PRINT CAT(A$,D$)
DO FAM

READY
```

Finally, the CHR$ function is used to match an ASCII character code to a character. As an example, the letter Q corresponds to character code 81. This letter can be printed by executing the statement:

```
PRINT CHR$(81)
Q

READY
```

Other, non-printing characters can be output by using the CHR$ function. The following special characters can be printed using the CHR$ function:

Bell	CHR$(7)
Line feed	CHR$(10)
Form feed	CHR$(12)
Carriage return	CHR$(13)

ASCII character codes are also useful in alphabetizing names and words, since the codes correspond to the letters in alphabetical order. Relational operators, =, >, <, >=, <=, <>, can be used with strings in the same manner as they are used with numeric operations.

For a complete list of ASCII characters and their codes, see Appendix C of this manual.

1.3 WRITING A PROGRAM

A program is a list of instructions the computer obeys. The computer must execute these instructions, or statements, in the order you specify.

Consequently, the system executes each statement completely before going on to the next statement. Once it executes a statement, it proceeds to the next statement immediately and begins to execute it automatically.

As the programmer, you can direct the system in advance to execute statements in a special order. This is accomplished within the program itself. Once the program is written, however, the system executes it one statement at a time.

Line numbers are used to order statements in a program. They must appear in the program in ascending order. In the EduSystem, however, you can type program lines in any order since lines are automatically sorted as they are entered.

Normally, the system executes statements, one by one, from the lowest numbered statement to the highest one. Of course, you can direct the system to execute statements out of order, by using certain statements in the program.

To enter a statement in a program, simply type in a line number, leave a space, and then type in the statement. Any number from 1 to 2046 (but no commas or decimal points) can be used as a line number. Store the entire line by typing the RETURN key, as if you were executing a statement in immediate mode. Notice that in immediate mode, when the line number is omitted, the statement is executed; in the programming mode, the statement is simply stored for later execution.

Leave an interval between line numbers in a program in case you want to insert other lines between the ones you have written. Programmers conventionally use a line numbering interval of 10 starting from line 10.

```
10 PRINT...
20 PRINT...
30 PRINT...
40 PRINT...
   etc.
```

Real pessimists, however, use a line number interval of 20.

To replace a statement in a program, simply retype the line number of the old statement and type in the new statement. The new line automatically replaces the old one.

To delete a line from a program, type the line number of the old statement and then type the RETURN key. No trace of the deleted line remains in computer memory.

Most of the statements discussed in the following sections can be executed directly from a keyboard or from a program. Some, however, can be executed only from a program while others can be executed only in immediate mode. To differentiate these options, braces { } are used in the syntax description (throughout this manual) to designate optional items.

When braces enclose a line number, the statement can be executed either from a keyboard or from a program.

When a line number appears without braces, the statement can be executed only from a program.

When no line number appears before a statement, that statement can be executed only from the keyboard in immediate mode.

Storage space can be saved by using the backslash character (\ ) to separate statements on one line. For example, the statements:

```
100 PRINT X↵
110 PRINT Y↵
120 PRINT Z↵
```

can be stored as one line:

```
100 PRINT X \ PRINT Y \ PRINT Z↵
```

In the program examples throughout this manual, however, each statement is preceded by its own line number.

Finally, all statements in the BASIC language can be abbreviated by typing only the first three characters of the word. For example, the PRINT statement works equally well as PRI. You are permitted - in fact, encouraged - to use the abbreviated forms whenever possible to save time in writing programs. Throughout this manual, however, in an effort to make the statements as self-explanatory as possible, full statement names are used consistently.

In the sections that follow, this manual discusses how you can direct the computer to do your work for you. Each programming statement is discussed in a separate section so that you can refer to specific statements without having to reread the entire manual.

1.3.1 PRINT Statements: Printing a Message

The PRINT statement is used to print messages on the terminal. In addition, values of variables, expressions and constants can be printed and lines can be skipped to separate outputs.

Syntax: {line number} PRINT {any combination of text and expressions}

Examples:
```
40  PRINT 9*2
    PRINT "THIS IS IMMEDIATE MODE."
80  PRINT "THIS SUM IS" 278*Y
    PRINT 14, 23*X, 19.3-44
90  PRINT
```

In the above examples, the statements without line numbers can be executed in immediate mode, but the statements with line numbers must be programmed. This is also the convention used in the following sections.

The system evaluates constants or variables which follow the PRINT statement and prints these numeric values on the terminal when the PRINT statement is executed. A literal expression is printed when the characters are enclosed in quotation marks. When nothing follows a PRINT statement, the terminal simply skips a line in the printout.

Terminal paper is separated into five print zones, each containing 13 character spaces. When the system encounters a comma in a PRINT statement, it immediately skips to the next available print zone. For example, the statement:

```
PRINT 1,2,3
```

prints these three constants at the beginning of each of three print zones.

```
PRINT 1,2,3
 1             2             3

READY
```

In the same way, literals are also printed at the beginning of print zones. For example,

```
PRINT "THIS IS REALLY", "ALL ONE LINE."
THIS IS REALLY              ALL ONE LINE.

READY
```

To suppress this automatic spacing, use semicolons instead of commas.

```
PRINT 1;2;3
 1  2  3

READY

PRINT "THIS IS REALLY"; "ALL ONE LINE."
THIS IS REALLYALL ONE LINE.
```

Generally, a PRINT statement instructs the system to include a carriage return and line feed operation. This is why the terminal is positioned at the left margin of the paper after each PRINT operation. To suppress the carriage return/line feed, type a semicolon after the last expression in the PRINT statement. A subsequent PRINT statement then begins printing on the same line.

There are two ways to skip over print positions on a page: print blank characters or use the TAB function. Any number of blank characters can be printed in a PRINT statement between quotation marks. For example:

```
PRINT "      " 123 "JUST TESTING              FERDINAND!"
      123 JUST TESTING              FERDINAND!

READY
```

The TAB function instructs the system to begin printing on the print position specified. These print positions are numbered from the left margin to the right, 1,2,3,...,72. The print position on which you want to begin printing must be specified in the TAB function within parentheses (e.g., TAB(12), TAB(M+5)). To print the word HERE beginning on print position 30, for example, simply execute the following line:

```
PRINT TAB(30) "HERE"↵
                              HERE

READY
```

PRINT statements are normally used to identify certain outputs in a program. They are used frequently throughout this manual to form explanatory headings in programs.

1.3.2 END and STOP Statements: Halting a Program

The END statement is used to terminate a program. When the system reaches an END statement, it ceases execution and is ready to accept new statements either in immediate mode or in a program.

Syntax: line number END

Example: 2000 END

Once you have typed the END statement with a line number, you have written a program. The program may be short (one line) and it may not do anything, but it is a program. All programs stop when they reach an END statement. For this reason, END should always be the last statement in the program. It should have the highest line number.

In the EduSystem, an END statement is not required to terminate a program. But since other systems and languages do require END statements, this manual uses one in every program to develop the habit.

The program shown below consists of a PRINT statement and an END statement:

```
10 PRINT "THIS IS A PROGRAM!"
20 END
```

Notice that both statements have preceding line numbers. When this program is run, the system executes the lowest numbered line first (the PRINT statement) and then executes progressively higher numbered lines (in this case, the END statement).

To run this program, type in the word RUN and type the RETURN key. (Detailed information on running a program can be found in Section 1.4.)

```
RUN↵
THIS IS A PROGRAM!

READY
```

When the system executes the END statement, the program has run completely. To re-execute the program, type RUN and the RETURN key again. Since the program is stored in memory, it can be run over and over again simply by typing RUN each time.

Sometimes it is necessary for a programmer to end a program in more than one place. This can be accomplished with a STOP statement.

Syntax: line number STOP

Example: 40 STOP

Consider the following program:

```
10 PRINT "THIS IS THE FIRST PART."
20 STOP
30 PRINT "THIS IS THE SECOND PART."
40 END
```

Run this program.

```
RUN↵
THIS IS THE FIRST PART.

READY
```

Notice that the program halts execution when it reaches the STOP statement on line 20. Therefore, lines 30 and 40 are never executed. Actually, there is a way to begin program execution at a specific line. See Section 1.3.4 or 1.4.3 for this information.

1.3.3 CTRL/C: Emergency Stop

The CTRL/C combination is used to abort a program and regain control of the keyboard. On occasion, program execution continues when you don't expect it to continue. The only way to stop program execution from the terminal is to type CTRL/C. For example, consider the following program:

```
10 PRINT 1
20 PRINT 2
30 PRINT 3
40 PRINT 4
50 PRINT 5
60 PRINT 6
70 PRINT 7
80 PRINT 8
90 PRINT 9
100 END
```

If you decide to stop this program before it runs completely, type CTRL/C while it is running. This prints the word STOP on the terminal. A carriage return/line feed is performed after the READY message is printed.

```
RUN↵
 1
 2
 3
 4
 5

STOP

READY
```

The program halts and you now have control of the terminal keyboard once more. Program execution cannot continue from any place in the program but the lowest numbered line (in this case, line 10). To execute the program again, type RUN and the RETURN key.

CTRL/C is an emergency stop. It should only be used when the program does completely unexpected things and there is no assurance that it will stop by itself. The program listed on the next page happens to be one of those cases.

Using CTRL/C while a program is running can sometimes lead to dangerous situations, especially when you are handling data. Don't use CTRL/C unless absolutely necessary.

1.3.4 GOTO Statements: Changing Direction

As mentioned before, the system executes statements in order, working from the lowest line number to the highest. You can change this order by using a GOTO statement in the program.

Syntax: {line number} GOTO line number

Examples:
```
20 GOTO 3000
   GOTO 30
```

When the system executes the GOTO statement, it branches to the line number specified in the GOTO statement.

The following example shows how the GOTO statement creates a programming loop. Remember that an emergency stop is performed by typing CTRL/C.

```
10 PRINT "HERE WE GO."
20 PRINT "HERE WE GO AGAIN."
30 GOTO 20
40 END

RUN↵
HERE WE GO.
HERE WE GO AGAIN.
HERE WE GO AGAIN.
HERE WE GO AGAIN.
HERE WE GO AGAIN.
STOP

READY
```

When you execute the GOTO statement in immediate mode, the system locates the line specified by the GOTO statement and begins to execute the program in memory from that line. For example, assume the program shown below is in memory. (This program first appears in Section 1.3.2.)

```
10 PRINT "THIS IS THE FIRST PART."
20 STOP
30 PRINT "THIS IS THE SECOND PART."
40 END
```

Executing a GOTO 30 statement from the keyboard instructs the system to begin execution from line 30, as follows

```
GOTO 30↵
THIS IS THE SECOND PART.

READY
```

Instructions on how to run a program with a GOTO statement can be found in Section 1.4.3.

1.3.5 GOSUB...RETURN Statements: Subroutines

When a certain operation must be executed a few times in a program it is often convenient to write the operation (called a subroutine) only once and refer to it throughout the program. This is accomplished with GOSUB (go to subroutine) and RETURN statements.

Syntax:

```
{line number}  GOSUB  line number
      .
      .
      .
      .
line number  RETURN
```

Examples:

```
GOSUB 200
```

```
50 GOSUB 1000
      .
      .
      .
      .
1000 statement
      .
      .
      .
      .
1100 RETURN
```

The subroutine to which the GOSUB statement refers can be anywhere in the program, as long as it is immediately followed by a RETURN statement. A program can have more than one GOSUB statement, but each RETURN statement must correspond to at least one GOSUB.

When the system encounters a RETURN, control returns to the next line after the previously executed GOSUB.

Your program can be laid out as shown below:

```
5 PRINT "THIS PROGRAM ILLUSTRATES THE USE OF GOSUB AND RETURN."
10 GOSUB 1000
20 PRINT "THIS IS THE FIRST PART OF THE PROGRAM."
30 GOSUB 1000
40 PRINT "THIS IS THE SECOND PART OF THE PROGRAM."
50 GOSUB 1000
60 PRINT "THIS IS THE THIRD PART OF THE PROGRAM."
70 GOSUB 1000
80 STOP
1000 PRINT "AS YOU CAN SEE,"
1010 PRINT "A SUBROUTINE CAN BE MORE THAN ONE LINE LONG..."
1020 PRINT "REMEMBER TO FOLLOW EACH SUBROUTINE WITH A RETURN."
1030 PRINT
1040 RETURN
1050 END
```

Terminal Output

```
RUN
THIS PROGRAM ILLUSTRATES THE USE OF GOSUB AND RETURN.
AS YOU CAN SEE,
A SUBROUTINE CAN BE MORE THAN ONE LINE LONG...
REMEMBER TO FOLLOW EACH SUBROUTINE WITH A RETURN.

THIS IS THE FIRST PART OF THE PROGRAM.
AS YOU CAN SEE,
A SUBROUTINE CAN BE MORE THAN ONE LINE LONG...
REMEMBER TO FOLLOW EACH SUBROUTINE WITH A RETURN.


THIS IS THE SECOND PART OF THE PROGRAM.
AS YOU CAN SEE,
A SUBROUTINE CAN BE MORE THAN ONE LINE LONG...
REMEMBER TO FOLLOW EACH SUBROUTINE WITH A RETURN.


THIS IS THE THIRD PART OF THE PROGRAM.
AS YOU CAN SEE,
A SUBROUTINE CAN BE MORE THAN ONE LINE LONG...
REMEMBER TO FOLLOW EACH SUBROUTINE WITH A RETURN.


READY
```

Notice the STOP on line 80. This is a precautionary measure to ensure the program cannot encounter a RETURN without previously executing a GOSUB statement.

1.3.6 ON-GOTO and ON-GOSUB Statements: More Directions

The ON-GOTO statement permits the program to transfer control to one of a set of line numbers depending on the value of a formula.

Syntax: {line number} ON expression GOTO line number {, line number, ...}

Examples:
```
50 ON D GOTO 100, 200, 300
   ON X-2 GOTO 110, 130, 180, 20
```

The expression is evaluated and then truncated to an integer. If the integer value of the expression is 1, the program transfers to the first line number specified; an integer value of 2 transfers the program to the second specified line; etc. The expression after truncation cannot be zero, negative or greater than the number of line numbers in the list. If the value is any of these, the system ignores the ON-GOTO statement and continues program execution at the next line.

The ON-GOSUB statement is used in the same manner as ON-GOTO, discussed above.

Syntax: {line number} ON expression GOSUB line number {,line number,...}

Consider the following example:

```
400 ON T+1 GOSUB 510, 520
```

The system transfers control to line 510 when the expression (T+1) equals 1 and to line 520 when the expression equals 2. Notice that in order for (T+1) to equal 1, T must equal Ø. Similarly when T=1, the expression (T+1) equals 2. This example shows that you can manipulate data any way you want in the expression of the ON-GOSUB and ON-GOTO statements.

In immediate mode, ON-GOTO and ON-GOSUB begin program execution at the specified line, similar to immediate mode GOTO and GOSUB statements (see Sections 1.3.4 and 1.3.5).

1.3.7 LET Statements: Assigning Variables

The LET statement assigns a value to a variable.

Syntax: {line number} {LET} variable = expression

Examples:
```
30  LET X = 45
    LET R4 = 32*M
    S = A*T↑2/2
```

The word LET is unnecessary in EduSystems. Assignment can be accomplished simply by using an equal sign (=), as shown in Section 1.2.1.4.

Consider the example shown below.

```
10 LET W = 3
15 PRINT "VARIABLE W    VARIABLE Z   VARIABLES W+Z"
20 PRINT W
30 LET Z = W * 100
40 PRINT W, Z, W+Z
45 W = 5
48 PRINT W, Z
50 END
```

```
RUN
VARIABLE W    VARIABLE Z   VARIABLES W+Z
 3
 3             300          303
 5             300

READY
```

Notice that each LET statement for a specific variable supersedes previous LET statements without affecting other variables. In this way, a variable can be used over and over again and can equal different values in the same program. You can print or perform arithmetic operations on any variable at any time in the program.

```
5 PRINT "POWERS OF 2"
10 E = 0
20 PRINT "2 TO THE ZERO POWER ="; 2^E
30 E = 1
40 PRINT "2 TO THE FIRST POWER ="; 2^E
50 E = 2
60 PRINT "2 TO THE SECOND POWER ="; 2^E
70 E = 3
80 PRINT "2 TO THE THIRD POWER ="; 2^E
90 E = 4
100 PRINT "2 TO THE FOURTH POWER ="; 2^E
110 END

RUN
POWERS OF 2
2 TO THE ZERO POWER = 1
2 TO THE FIRST POWER = 2
2 TO THE SECOND POWER = 4
2 TO THE THIRD POWER = 8
2 TO THE FOURTH POWER = 16

READY
```

1.3.8 Counters: Incrementing Variables

You can use a form of the LET statement to count for you. Since each LET statement in a program supersedes previous LET statements for the same variable, a variable can be incremented as shown below.

```
40 LET G = G+1
```

or

```
40 G = G+1
```

This statement assigns a new value for G. Now the new value is equal to the old value plus one. If G was originally 13, for example, it is now 14. Of course, the variable does not have to be incremented by only one each time; it can be incremented by any number or fraction. In fact, it can also be multiplied by a factor, divided, raised to a power, or decremented (subtracted from).

Using variables as counters helps keep track of specific operations. This information can be used later in the program. For example:

```
10 K = 1
20 PRINT "THIS IS STATEMENT NUMBER"; K
30 K = K + 1
40 GOTO 20
50 END

RUN
THIS IS STATEMENT NUMBER 1
THIS IS STATEMENT NUMBER 2
THIS IS STATEMENT NUMBER 3
THIS IS STATEMENT NUMBER 4
THIS IS
STOP

READY
```

Be sure the GOTO statement specifies the correct line number. It must not direct the system to execute the first LET statement. If this happens, the variable K is always 1 or 2.

Even numbers can be printed:

```
10 LET Y = 2
20 PRINT Y;
30 LET Y = Y + 2
40 GOTO 20
50 END
```

```
RUN
 2  4  6  8  10  12  14  16
STOP

READY
```

Odd numbers can also be printed, simply by adjusting the first value of the variable to an odd integer.

```
10 LET U = -1
20 LET U = U + 2
30 PRINT U;
40 GOTO 20
50 END
```

```
RUN
 1  3  5  7  9  11  13  15  17
STOP

READY
```

A multiplication table can be generated easily by using a counter.

```
10 PRINT "NUMBER", "TIMES 2", "TIMES 3", "TIMES 4"
20 N = 0
30 PRINT N, 2*N, 3*N, 4*N
40 N = N + 1
50 GOTO 30
60 END
```

```
RUN
```

NUMBER	TIMES 2	TIMES 3	TIMES 4
0	0	0	0
1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16
5	10	15	20
6			

```
STOP

READY
```

1.3.9 IF Statements: Testing Conditions and Making Decisions

The IF statement allows the system to branch to other parts of the program when certain conditions are reached. As the programmer, you specify the conditions and you specify where the program branches to.

Syntax: {line number} IF expression THEN line number or executable statement

Examples:
```
45 IF G = 99 THEN 3000
60 IF M * N>P THEN PRINT "FINISHED!"
```

A program may include certain computations that are executed over and over again. Rather than typing CTRL/C to stop program execution, use the IF statement to indicate how many times the computation has been done. When a counter reaches a certain point, simply branch to another part of the program.

For example, the program below extracts the square root from the first ten consecutive numbers.

```
10 PRINT "NUMBER", "SQUARE ROOT"
20 LET N = 1
30 PRINT N, SQR(N)
40 LET N = N + 1
50 IF N = 11 THEN 70
60 GOTO 30
70 PRINT "THIS COMPLETES THE PROGRAM."
80 END
```

```
RUN
NUMBER          SQUARE ROOT
 1               1
 2               1.414214
 3               1.732051
 4               2
 5               2.236068
 6               2.44949
 7               2.645751
 8               2.828427
 9               3
 10              3.162278
THIS COMPLETES THE PROGRAM.

READY
```

Unless the specified terminal condition in the IF statement is true, the program ignores the line and continues program execution on the next highest line number automatically.

In the above example, notice that the IF statement on line 50 directs the program to branch to line 70 when the variable N is equal to 11. By directing the program to line 70 from line 50, the GOTO statement on line 60 is skipped over. When the expression (N = 11) is false, the system ignores line 50 and executes the next highest line number, line 60.

The IF statement is executed only when the expression is true; when the expression is false, the system simply ignores the IF statement and executes the next line number. In this way, the IF statement is a powerful method by which you can instruct the system to execute statements in the order you specify.

Once again, be sure to use the right line number in the GOTO statement. If, in the above example, you write 60 GOTO 20, N is redefined after each programming loop and the program prints:

```
1           1
```

over and over again, since N never reaches 2.

Here is another example, showing how you can nest certain statements between two or more counters.

```
 10 LET H = 1
 20 PRINT "ON HAND NUMBER"; H; "I HAVE:"
 30 LET F = 1
 40 PRINT F; "FINGERS"
 50 LET F = F + 1
 60 IF F > 5 THEN 80
 70 GOTO 40
 80 LET H = H + 1
 90 IF H < 3 THEN 20
100 END
```

In the above example, the number of hands, represented by variable H, is incremented in line 80. This number is tested on line 90. If the number of hands is less than 3, the expression (H<3) is true and the program branches to line 20. On the other hand (pardon the expression) if the number of hands equals or is greater than 3, the system simply ignores line 90 and executes line 100 END. This is because the expression (H<3) in the IF statement is false.

The number of fingers on each hand, represented by variable F, is incremented in line 50, within the loop created by variable H. The IF statement that tests the value of F is in line 60. When the expression (F>5) is false, the system simply ignores line 60 and executes the next highest numbered line, line 70. When the expression is true, however, line 60 instructs the system to execute line 80 directly.

The output of this program is:

```
RUN
ON HAND NUMBER 1 I HAVE:
 1 FINGERS
 2 FINGERS
 3 FINGERS
 4 FINGERS
 5 FINGERS
ON HAND NUMBER 2 I HAVE:
 1 FINGERS
 2 FINGERS
 3 FINGERS
 4 FINGERS
 5 FINGERS

READY
```

Other executable expressions can also be included in an IF statement. For example,

```
20 IF X=15 THEN PRINT "END OF DATA"
```

Obviously, the system prints END OF DATA when the expression (X=15) is true and does not print the message when X≠15. In the latter case, the system simply ignores line 20 and continues execution at the next highest line of the program.

Similarly, any executable expression or statement can be included in the IF statement. Some examples are shown below:

```
40 IF B=99 THEN X=X+1
90 IF P3>.001 THEN STOP
45 IF L*R2=3.14159 THEN INPUT "NEW RADIUS";R2
33 IF V$ = "NAME" THEN GOSUB 1200
```

When one of the statements on a multiple statement line is an IF statement, all of the statements beyond the IF statement are executed regardless of whether the IF expression is true. For example,

```
10 X = 50
20 IF X = 30 THEN PRINT "X=30" \ PRINT "FIRST LINE"
30 IF X = 50 THEN PRINT "X=50" \ PRINT "SECOND LINE"
40 END
```

```
RUN
FIRST LINE
X=50
SECOND LINE

READY
```

In the above example, notice that the second statement on the multiple statement lines 20 and 30 is always executed, even when the preceding IF statement's expression is false. For this reason, it's best to use IF statements on separate lines.

1.3.10 FOR...NEXT Statements: Programming Loops

On many occasions a FOR...NEXT statement can be used to replace an incremented counter and an IF statement. Actually, FOR and NEXT are two separate statements.

Syntax:

```
{line number}  FOR simple variable = expression
    :                  TO expression {STEP expression}
    :
{line number}  NEXT simple variable
```

Examples:

```
35  FOR J = 1 TO 8          50  FOR M = R + 2 TO Y STEP 3
 :                           :
 :                           :
70  NEXT J                  100 NEXT M
```

The system keeps track of the variable specified in the FOR statement. When program execution reaches the NEXT statement, the system automatically tests to see whether the final value of the variable is reached. If it is not, the NEXT statement increments the variable by one and then branches to the line directly below the FOR statement. When the final value of the variable is reached, the NEXT statement allows the system to execute the next highest numbered line.

The optional STEP parameter in the FOR statement allows you to increment or decrement the variable by more or less than 1. Fractions and variables can be used. Consider the example below.

```
10 PRINT "THESE ARE THE FIRST SIX NUMBERS."
20 FOR P = 1 TO 6
30 PRINT P;
40 NEXT P
50 PRINT "FINISHED."
60 END

RUN
THESE ARE THE FIRST SIX NUMBERS.
 1  2  3  4  5  6 FINISHED.

READY
```

The FOR...NEXT loop can be of any length and loops can be nested, as long as the same variable is not used for more than one loop.

Notice how much shorter and clearer the example in section 1.3.9 becomes by using FOR...NEXT loops. The left hand brackets define the loops for easy analysis.

```
10 FOR H = 1 TO 2
20 PRINT "ON HAND NUMBER"; H; "I HAVE:"
30 FOR F = 1 TO 5
40 PRINT F; "FINGERS"
50 NEXT F
60 NEXT H
70 END

RUN
ON HAND NUMBER 1 I HAVE:
 1 FINGERS
 2 FINGERS
 3 FINGERS
 4 FINGERS
 5 FINGERS
ON HAND NUMBER 2 I HAVE:
 1 FINGERS
 2 FINGERS
 3 FINGERS
 4 FINGERS
 5 FINGERS

READY
```

1.3.11 INPUT and LINPUT Statements: Guiding the User

The INPUT statement is used to enter information into the program. Program execution halts temporarily while this information is typed in.

Syntax: line number INPUT {variable ,variable,...,variable}
line number LINPUT string variable

Examples:

```
20 INPUT X
40 INPUT M, R$, G
50 LINPUT T$
```

Sometimes it is convenient to ask the person using your program to enter information in the form of a numeric value or a literal. This is accomplished with the INPUT statement.

The INPUT statement assigns the value(s) entered to the variable(s) specified. The variable(s) can then be used in the program; LET statements are not required. Any combination of alphanumeric characters up to and including six characters can be input.

An INPUT statement prints a question mark (?) to prompt the user to answer. Often the question mark is not enough to explain what information the program is looking for. So it is a good idea to first print a message, in the form of a question, before executing the INPUT statement. Remember: a semi-colon suppresses the carriage return/line feed in a PRINT statement.

Consider the following example:

```
 10 PRINT "THIS PROGRAM TELLS YOU HOW OLD YOU'LL BE IN THE YEAR 2000."
 20 PRINT "DO YOU WANT TO KNOW";
 30 INPUT L$
 40 IF L$ = "NO" THEN 100
 50 PRINT "WHAT IS YOUR PRESENT AGE";
 60 INPUT A
 70 PRINT "WHAT YEAR IS THIS";
 80 INPUT Y
 90 PRINT "YOU WILL BE "; A + 2000 - Y; "YEARS OLD IN THE YEAR 2000."
100 END
```

```
RUN
THIS PROGRAM TELLS YOU HOW OLD YOU'LL BE IN THE YEAR 2000.
DO YOU WANT TO KNOW? YES
WHAT IS YOUR PRESENT AGE? 15
WHAT YEAR IS THIS? 1975
YOU WILL BE  40 YEARS OLD IN THE YEAR 2000.

READY
```

A special form of the INPUT statement (but not the LINPUT statement) allows you to print a prompting question without using a separate PRINT statement. Use this form of the INPUT statement, separating the input variable(s) from the question with a semicolon:

```
70 INPUT "WHAT YEAR IS THIS"; Y
```

More than one variable can be assigned in an INPUT statement. The program below is a more general form of the previous one.

```
10 PRINT "THIS PROGRAM TELLS YOU HOW OLD YOU'LL BE IN A GIVEN YEAR."
20 INPUT "YOUR AGE, THIS YEAR, THE YEAR YOU'RE SHOOTING FOR"; A, Y, Y9
30 PRINT "YOU WILL BE"; A + Y9 - Y; "YEARS OLD IN THE YEAR"; Y9
40 END
```

When running this program, enter the numbers requested in order. The numbers must be separated by commas. Then type the RETURN key.

```
RUN↵
THIS PROGRAM TELLS YOU HOW OLD YOU'LL BE IN A GIVEN YEAR.
YOUR AGE, THIS YEAR, THE YEAR YOU'RE SHOOTING FOR? 15,1975,2010↵
YOU WILL BE 50 YEARS OLD IN THE YEAR 2010

READY
```

As mentioned before, alphanumeric characters can be entered in an INPUT statement when a string variable is used. String variables can also be intermixed with simple numeric variables in the same INPUT statement.

For more than six characters per string variable, use the LINPUT (Line INPUT) statement. LINPUT accepts a line of characters up to a carriage return. It separates this line into sections of six characters apiece and automatically assigns each section to a subscripted string variable.

Consider this example using an INPUT statement first:

```
10 INPUT "WHO WAS THE FIRST PRESIDENT OF THE U.S."; P$
20 PRINT P$
30 END

RUN↵
WHO WAS THE FIRST PRESIDENT OF THE U.S.? GEORGE WASHINGTON↵
GEORGE

READY
```

Normally, an INPUT statement accepts only the first six characters of an input as the string variable (in the above case, only the name GEORGE). Using LINPUT, however, the entire message can be accepted. You must use separate PRINT statements to print questions when executing LINPUT statements; including the message within a LINPUT statement is not allowed.

```
10 PRINT "USING LINPUT STATEMENTS."
20 PRINT "WHO WAS THE FIRST PRESIDENT OF THE U.S.";
30 LINPUT P$
35 N = P$(0)
40 PRINT N
50 PRINT P$(1)
60 PRINT P$(2)
70 PRINT P$(3)
75 PRINT
80 PRINT P$(1); P$(2);P$(3)
90 END
```

```
RUN
USING LINPUT STATEMENTS.
WHO WAS THE FIRST PRESIDENT OF THE U.S.? GEORGE WASHINGTON
 17
GEORGE
 WASHI
NGTON

GEORGE WASHINGTON

READY
```

In the above example, notice that the LINPUT statement creates subscripted variables, where the zero element (i.e., P$(Ø)) contains the number of characters, including imbedded spaces, subsequent elements (i.e., P$(1), P$(2),...) each contain six characters. The number of characters is stored as an alphanumeric string in the variable P$(Ø). To use this number, you first have to convert it to a numeric value with an assignment statement (in the above case, line 35).

Although the subscripted string variables now all have values, the simple string variable (in this case, P$) has no value, unless you previously assigned it a value. More information on subscripted variables can be found in Section 1.3.14.

When used in conjunction with the IF statement, INPUT allows you to enter data up to a certain point. For example, in the program shown below, input data is added together until 9999 is entered. This creates a terminal condition.

```
10 PRINT "THIS PROGRAM ADDS NUMBERS TOGETHER"
20 LET T = 0
30 INPUT "NEXT NUMBER"; I
40 IF I = 9999 THEN 70
50 T = T + I
60 GOTO 30
70 PRINT "YOUR TOTAL IS"; T
80 END

RUN
THIS PROGRAM ADDS NUMBERS TOGETHER
NEXT NUMBER? 4
NEXT NUMBER? 3.2
NEXT NUMBER? 20
NEXT NUMBER? -1.7
NEXT NUMBER? 9999
YOUR TOTAL IS 25.5

READY
```

Notice that since the IF statement in line 40 directs the system to line 70 when I = 9999, it is not added to the total (in line 50). So the printed total is the sum of all the numbers entered up to but not including the last number (9999).

1.3.12 REMARK Statements: What's Happening

The REMARK statement is used in a program to identify the entire program or certain parts of the program to the programmer. The information in a REMARK statement is ignored by the system and not printed on any device.

Syntax: line number REMARK {any combination of characters}
line number ' {any combination of characters}

Examples:
```
20 REMARK HOW'S THIS FOR AN EXAMPLE??
50 ' YOU CAN USE APOSTROPHES FOR SHORTHAND.
```

Use the REMARK statement freely throughout your program to identify specific program segments and what they accomplish. REMARK statements are a great aid in separating sections for the programmer.

The apostrophe form of a REMARK statement can also be used on the same line as a programming statement to identify what is happening on the specific program line. For example,

```
110 LET A = 1          'SET UP STARTING VALUE.
120 LET B = 10           'SET UP TERMINATING VALUE.
130 PRINT A
140 A = A + 1            ' INCREMENT VARIABLE A.
150 IF A = B THEN 170        'ARE WE DONE?
160 GOTO 130                  'NO.
170 END                      'YES.
```

1.3.13 READ...DATA Statements: Number Crunching

The READ statement, in conjunction with DATA statements, is used to enter values of variables in a program.

Syntax:
```
line number READ variable {,variable,...,variable}
        ⋮
line number DATA constant {,constant,...,constant}
```

Examples:
```
200 READ X,Y,Z              300 READ A$, K
600 DATA 2.34,45,-12.6      310 DATA "MONDAY", 4
```

The INPUT statement is inefficient to enter large amounts of values into a program. The READ and DATA statements, however, can assign values to variables when these values are known ahead of time.

Consider the following example:

```
10 REMARK  THIS PROGRAM AVERAGES FIVE NUMBERS.
20 READ A,B,C,D,E
30 DATA 3,6,5,7,8
40 PRINT "THE AVERAGE OF THESE NUMBERS IS" (A+B+C+D+E)/5
50 END
```

```
RUN↵
THE AVERAGE OF THESE NUMBERS IS 5.8

READY
```

DATA statements can be located anywhere in the program before the END statement. When the system encounters a READ statement, it searches for the first DATA statement in the program and enters the first value in that statement. If the READ statement asks for more than one value, the system proceeds to the next value in the DATA statement or moves to the next DATA statement.

For example:

```
10 DATA 1,2,3
20 DATA 4
30 READ A,B,C,D,E,F,G,H
40 DATA 5,6
50 PRINT A;B;C;D;E;F;G;H
60 DATA 7
70 DATA 8
80 END
```

```
RUN
 1  2  3  4  5  6  7  8

READY
```

Programmers generally find it convenient to place all of the DATA statements together at the beginning or at the end of the program.

You can mix numeric variables with string variables in the READ statement, but be sure the type of data requested in the READ statement corresponds to the type of data in the DATA statements. If the system expects to see a numeric value, for example, and the next item in a DATA statement is a literal, an error message results.

When using literals in DATA statements, be sure to type the lines without spaces after the last A in the word DATA. For example:

```
100 DATA   "ABC"  ,9   , "DEF" ,  12      <---------- WRONG

           100 DATA"ABC",9,"DEF",12        <---------- RIGHT
```

1.3.13.1 RESTORE Statements: Using Data Again - The RESTORE statement is used to begin reading data again from the lowest numbered DATA statement in the program.

Syntax: line number RESTORE

When a RESTORE statement is executed, the program enters the first data items again in a READ statement. For example:

```
10 REMARK  THIS PROGRAM SHOWS THE USE OF RESTORE STATEMENTS.
20 DATA 1,2,3,4,5
30 DATA 6,7,8,9
40 READ A,B,C,D,E,F,G
50 PRINT A;B;C;D;E;F;G
60 RESTORE
70 READ X,Y,Z
80 PRINT X;Y;Z
90 END


RUN
 1  2  3  4  5  6  7
 1  2  3

READY
```

1.3.14 Subscripts and Arrays: More Variables

In addition to the 286 simple variables (from A to Z9) and the 26 simple string variables (from A$ to Z$) you can use in a program, arrays can store elements.

Arrays can be one-dimensional (e.g., R(8)) or two-dimensional (e.g., T(2,4)), but the same variable cannot be used for both one-dimensional and two-dimensional subscripting. Each element in an array can have a value assigned to it by an INPUT, READ, or LET statement.

```
10 REMARK  THIS PROGRAM ASSIGNS VALUES TO A NUMERIC ARRAY.
20 LET V(0) = 3.14
30 LET V(1) = 426
40 LET V(2) = 39/4
50 LET V(3) = 2000
60 PRINT V(0);V(1);V(2);V(3)
70 END
```

```
RUN
 3.14   426   9.75   2000

READY
```

```
10 REMARK  THIS PROGRAM ASSIGNS LITERALS TO A STRING ARRAY.
20 G$(1) = "ONE"
30 G$(2) = "TWO"
40 G$(3) = "THREE"
50 PRINT G$(1), G$(2), G$(3)
60 END
```

```
RUN
ONE            TWO            THREE

READY
```

It is often easier to write a program with one array than with a large number of different variables. For example, to store 15 even numbers without an array, you must use 15 different variables.

```
10 REMARK  THIS PROGRAM STORES 15 EVEN NUMBERS THE LONG WAY.
20 READ A,B,C,D,E,F,G,H,I,J,K,L,M,N,O
30 DATA 2,4,6,8,10,12,14,16,18,20,22,24,26,28,30
40 PRINT A;B;C;D;E;F;G;H;I;J;K;L;M;N;O
50 END

RUN
 2  4  6  8  10  12  14  16  18  20  22  24  26  28  30

READY
```

You can easily see the problems that arise when 1,000 values or 10,000 values are to be stored in separate variables. Using an array, however, is considerably easier.

```
 5 REMARK  THIS PROGRAM STORES 15 EVEN NUMBERS IN AN ARRAY.
10 FOR I = 1 TO 15
30 G(I) = 2 * I
40 NEXT I
50 FOR M = 1 TO 15
60 PRINT G(M);
70 NEXT M
80 END
RUN
 2  4  6  8  10  12  14  16  18  20  22  24  26  28  30
READY
```

Notice that a variable can be used to identify a specific element in an array. Different variables can be used for this purpose in a program as long as you keep track of their values. In the above program, both I and M are used to identify specific elements in the same array, G.

A two-dimensional array can be used. The first number in a two-dimensional array identifies the column of the element and the second number identifies the row.

For example, consider the array, Y, which has 20 elements as follows:

Col. 0	Col. 1	Col. 2	Col. 3	Col. 4	
Y(0,0)	Y(0,1)	Y(0,2)	Y(0,3)	Y(0,4)	◄—— Row 0
Y(1,0)	Y(1,1)	Y(1,2)	Y(1,3)	Y(1,4)	◄—— Row 1
Y(2,0)	Y(2,1)	Y(2,2)	Y(2,3)	Y(2,4)	◄—— Row 2
Y(3,0)	Y(3,1)	Y(3,2)	Y(3,3)	Y(3,4)	◄—— Row 3

Generally, programmers store values in two-dimensional arrays in row-column order. To store the number 278 in each element of the above array, use this program.

```
10 REMARK   THIS PROGRAM PRINTS A 4X5 ARRAY.
20 FOR I = 0 TO 3
30 FOR J = 0 TO 4
40 Y(I,J) = 278
50 NEXT J
60 NEXT I
70 END

RUN

READY
```

In the above program, row zero (I = 0) is filled up first as J goes from 0 to 4. Then row one (I = 1) is filled up as J goes from 0 to 4, and rows two and three are filled in the same way.

After running the above program, array Y is stored in memory. You can determine the value of any element in array Y with a PRINT statement in immediate mode. For example,

```
PRINT Y(2,2)
 278

READY
```

To print all the values of the elements in this array, simply add a print routine to the end of the program.

```
10 REMARK   THIS PROGRAM PRINTS A 4X5 ARRAY.
20 FOR I = 0 TO 3
30 FOR J = 0 TO 4
40 Y(I,J) = 278
50 NEXT J
60 NEXT I
61 REMARK  PRINT Y ARRAY ELEMENTS.
62 FOR M = 0 TO 3
64 FOR N = 0 TO 4
66 PRINT Y(M,N);
68 NEXT N
69 PRINT
70 PRINT
80 NEXT M
90 END
```

In this case, two PRINT statements are required: one cancels the semicolon in line 66, allowing a carriage return/line feed; the other skips a line between rows for a cleaner table of numbers.

```
RUN
 278  278  278  278  278

 278  278  278  278  278

 278  278  278  278  278

 278  278  278  278  278

READY
```

1.3.14.1 DIM Statements: Dimensioning Arrays - The DIM statement prepares the system to work with arrays with a fixed number of elements. In many systems, this is a required statement for array manipulation, usually found in the beginning of programs; EduSystem BASIC does not require it.

Syntax: line number DIM array variable {,array variable{,...}}

Example:

```
10 DIM A(10), B$(2,5), G(8,8)
```

Define the size of arrays in the DIM statement, but remember that each one-dimensional array also includes the zero column. Dimensioning array L(4), for example, reserves five elements as follows:

L(0), L(1), L(2), L(3), L(4)

Similarly, a two-dimensional array has one additional row and one additional column. For example, dimensioning array P(2,2) reserves 3 x 3, or nine elements as follows:

P(0,0)	P(0,1)	P(0,2)
P(1,0)	P(1,1)	P(1,2)
P(2,0)	P(2,1)	P(2,2)

1.3.15 DEF FN Statements: Do It Yourself Functions

In addition to the arithmetic and trigonometric functions discussed in Section 1.2.1.2, you may want to define other functions for a particular program. The DEF FN statement provides a convenient method for doing this.

Syntax: line number DEF FN letter (local variable {,variable{,...}}) = expression

Examples:

```
40 DEF FNM(R) = 3.14159*R↑2
10 DEF FNA(M1,M2,D) = G*M1*M2/D↑2
```

The program shown below converts degrees from Centigrade to Fahrenheit.

```
10 REMARK  CONVERT DEGREES FROM CENTIGRADE TO FAHRENHEIT.
20 DEF FND(T) = (9/5) * T + 32
30 INPUT "TEMPERATURE IN CENTIGRADE DEGREES"; C
40 PRINT FND(C) "FAHRENHEIT."
50 GOTO 30
60 END

RUN
TEMPERATURE IN CENTIGRADE DEGREES? 0
 32 FAHRENHEIT.
TEMPERATURE IN CENTIGRADE DEGREES? 100
 212 FAHRENHEIT.
TEMPERATURE IN CENTIGRADE DEGREES? -40
-40 FAHRENHEIT.
TEMPERATURE IN CENTIGRADE DEGREES?
STOP

READY
```

Notice that the function defined in line 20 as FND(T), is used in line 40. The variable C replaces T in the above program as long as it is in the correct form, FND(C). The letter D is a dummy variable, used by the system only to identify the defined function. Up to 26 different functions (one for each letter of the alphabet) can be defined in one program, but a separate DEF FN statement must be used for each.

1.3.16 Using Arithmetic and Trigonometric Functions

The table in Section 1.2.1.2 lists and describes EduSystem's arithmetic and trigonometric functions. Actually, you can write a program to perform any one of the arithmetic functions, and it would be a good programming exercise for you to do so. (Trigonometric functions can be approximated by using Taylor's series.)

The built-in functions in EduSystem, however, provide a convenient way to perform these operations. The program shown below, for example, prints the hypotenuse when the other two sides of a right triangle are input.

```
10 REMARK  RIGHT TRIANGLE PROGRAM.
20 INPUT "LENGTH OF FIRST LEG"; L1
30 INPUT "LENGTH OF SECOND LEG"; L2
40 PRINT "THE HYPOTENUSE IS" SQR(L1^2 + L2^2)
50 END

RUN↵
LENGTH OF FIRST LEG? 3↵
LENGTH OF SECOND LEG? 4↵
THE HYPOTENUSE IS 5

READY
```

In the above example, notice the use of the SQR (square root) function in line 40.

Adding the INT (integer) function to the program on line 45 instructs the system to print out the hypotenuse length after truncating it to a whole number. For example,

```
10 REMARK  RIGHT TRIANGLE PROGRAM.
20 INPUT "LENGTH OF FIRST LEG"; L1
30 INPUT "LENGTH OF SECOND LEG"; L2
40 PRINT "THE HYPOTENUSE IS" SQR(L1^2 + L2^2)
45 PRINT "THIS LENGTH TRUNCATED IS" INT(SQR(L1^2 + L2^2))
50 END
```

```
RUN
LENGTH OF FIRST LEG? 1
LENGTH OF SECOND LEG? 1
THE HYPOTENUSE IS 1.414214
THIS LENGTH TRUNCATED IS 1

READY
```

Of course, if the result of the expression before truncation is an integer, the INT function does not affect the result.

```
RUN
LENGTH OF FIRST LEG? 5
LENGTH OF SECOND LEG? 12
THE HYPOTENUSE IS 13
THIS LENGTH TRUNCATED IS 13

READY
```

1.3.16.1 RANDOMIZE Statements: A Game of Chance - The random number (RND) function produces a random number between 0 and 1. This is the only function that you cannot duplicate by conventional programming methods. A given program using the RND function produces the same random numbers each time it is run. For example,

```
10 REMARK  RANDOM NUMBER EXAMPLE.
20 PRINT RND (-.52143)
30 PRINT RND (-.52143)
40 PRINT RND (-.52143)
50 END
```

```
RUN
 .2431684
 .2988412
 .7295008

READY
```

The same random numbers are returned each time the program is run.

```
RUN
 .2431684
 .2988412
 .7295008

READY
```

To calculate different random numbers every time, use the RANDOMIZE statement. This statement is usually placed near the beginning of a program, as shown below.

```
10 REMARK  RANDOM NUMBER EXAMPLE.
15 RANDOMIZE
20 PRINT RND (-.52143)
30 PRINT RND (-.52143)
40 PRINT RND (-.52143)
50 END
```

```
RUN
 .1171918
 .9209115
 .5957117

READY
```

Running this program again yields a different set of random numbers, as follows:

```
RUN
 .3720747
 .6855599
 .639657

READY
```

The best arguments to use in the RND function are between 0 and -1. They should have a large number of digits (up to 11), and the last digit should be a 1, 3, 7, or 9. These rules help ensure the most random numbers returned.

You can invent many games with the use of the RND function and the RANDOMIZE statement, so that the chances of playing the game the same way twice are remote. For example, the program shown below challenges you to guess the number the computer is "thinking" of.

```
 10 REMARK  "GUESS MY NUMBER" GAME.
 20 RANDOMIZE
 25 I = 1
 30 N = INT(100*RND(-.552367))
 40 INPUT "GUESS MY NUMBER"; X
 45 IF X = N THEN 95
 50 IF X > N THEN PRINT X "IS TOO LARGE."
 60 IF X < N THEN PRINT X "IS TOO SMALL."
 80 I = I + 1
 90 GOTO 40
 95 PRINT "YOU GOT IT IN" I "GUESSES!!"
100 END
```

```
RUN↵
GUESS MY NUMBER? 25↵
 25 IS TOO SMALL.
GUESS MY NUMBER? 75↵
 75 IS TOO LARGE.
GUESS MY NUMBER? 60↵
 60 IS TOO LARGE.
GUESS MY NUMBER? 58↵
 58 IS TOO SMALL.
GUESS MY NUMBER? 59
YOU GOT IT IN 5 GUESSES!!

READY
```

1.4 RUNNING A PROGRAM

1.4.1 RUN Statements

To run a program you have just written, simply type the word RUN and type the RETURN key.

Syntax: RUN

When the RUN statement is executed from the terminal keyboard, all variables are set to zero and then the program currently in memory is executed. When program execution is over, the system prints the word READY, signifying the computer can now obey another instruction or set of instructions.

1.4.2 GOTO Statements

A GOTO statement, executed in immediate mode, directs the system to locate the specified line and begin executing the program in memory at that line. Values of all variables previously assigned are unchanged. This is a convenient way to resume program execution from a certain point in the program after a STOP statement halts the program.

For example, assume the program shown below is in memory.

```
10 PRINT "WHERE DO WE GO FROM HERE?"
20 STOP
30 PRINT "NEW YORK"
40 STOP
50 PRINT "BOSTON"
60 STOP
70 PRINT "DENVER"
80 END
```

Portions of this program can be executed by directing the system with GOTO statements, as follows.

```
RUN ↵
WHERE DO WE GO FROM HERE?

READY

GOTO 70 ↵
DENVER

READY

GOTO 50 ↵
BOSTON

READY
```

When using GOTO statements in immediate mode, be sure no program information before the specified line is required in the remainder of the program. Using a variable, for example, that was assigned previously in the program creates illogical results. Also, accessing a program in the middle of a FOR...NEXT loop or the middle of a subroutine results in errors 44 and 30, respectively.

1.5 LISTING A PROGRAM

To print the contents of your program on the terminal, type in the word LIST and type the RETURN key.

Syntax: LIST {1st line number {,2nd line number}}

Examples:

```
LIST
LIST 40
LIST 35, 210
```

When LIST is executed without specifying the first or second line numbers, the entire program in memory is printed on the terminal.

Specifying the first line number in the LIST statement instructs the system to print only that line.

When both the first and second line numbers are specified in the LIST statement, separated by a comma, the system prints both lines as well as all of the lines between them.

If the program you have in memory is:

```
10 PRINT "LINE 10"
20 PRINT "LINE 20"
30 PRINT "LINE 30"
40 PRINT "LINE 40"
50 PRINT "LINE 50"
60 END
```

A listing can be obtained by executing LIST.

```
LIST↵

  10 PRINT "LINE 10"
  20 PRINT "LINE 20"
  30 PRINT "LINE 30"
  40 PRINT "LINE 40"
  50 PRINT "LINE 50"
  60 END

READY
```

A specific line can be printed by using the line number in the LIST statement. For example:

```
LIST 30↵
  30 PRINT "LINE 30"

READY
```

A group of lines can be listed in order by using two line numbers in the LIST statement. For example:

```
LIST 20, 50↵
  20 PRINT "LINE 20"
  30 PRINT "LINE 30"
  40 PRINT "LINE 40"
  50 PRINT "LINE 50"

READY
```

1.6 EDITING A PROGRAM

A program may require editing either before a line is stored with the RETURN key, or after the line is stored. Each situation has its own editing procedures.

1.6.1 Before a Line is Stored

Three keys can be used to correct typing errors: RUBOUT, ← (SHIFT/O), ALT MODE (ESC on some terminals).

The back arrow, ←, SHIFT/O on the keyboard, or RUBOUT is used to delete a charater from a line. The system prints a back arrow or an underline, deleting the last character from that line. More than one back arrow deletes more than one character, in reverse order.

ALT MODE is used to delete an entire line. When this key is used, the system prints $DELETED, erases that line from the program, and returns the carriage so that the line can be retyped.

1.6.2 After a Line is Stored

Once a line of the program is transmitted to computer memory via the RETURN key, several methods of correction can be used. Lines can be inserted, deleted or changed. The sections that follow describe these methods.

1.6.2.1 Inserting Lines - To add a line to a program, simply assign a line number that falls between two existing lines, enter the line number and text, and type the RETURN key.

1.6.2.2 Deleting Lines - To erase a line from memory, type the line number only and then type RETURN. The DELETE statement can also be used to erase lines from memory.

Syntax: DELETE {1st line number{, 2nd line number}}

Examples:
```
DELETE
DELETE 25
DELETE 80, 110
```

When DELETE is executed without specifying the first or second line numbers, the entire program in memory is erased.

Specifying the first line number in the DELETE statement instructs the system to delete only that line.

When both the first and second line numbers are specified in the DELETE statement, separated by a comma, the system deletes both lines as well as all of the lines between them.

1.6.2.3 EDIT Statements - Old instructions can be replaced by new ones by retyping the line. This procedure is adequate for changing simple lines, but when a line contains a long, complex formula, it is easier to use the EDIT statement.

Syntax: EDIT line number

The EDIT statement allows you to access a single line and search for the character or characters to be changed. Once the EDIT statement is executed from the keyboard, the system waits for you to type a search character. (The system does not print this search character when you type it.) The search character is one that already exists on the line to be changed. After the search character is typed, the system prints the contents of that line until the search character is printed. At this point, printing stops and you have the following options:

- Type new characters; the system inserts them following the ones already printed.
- Type a form feed (CTRL/L); the system searches for the next occurrence, if any, of the search character.
- Type a bell (CTRL/G); this signals a change of search character. Then type a new search character.
- Type a RUBOUT or ←; this deletes one character to the left each time the key is typed.
- Type the RETURN key; the system terminates editing of the line at that point, deleting any existing text to the right.
- Type the ALT MODE key; this deletes all existing characters on the line except the line number and prints $DELETED on the terminal. At this point, type a new line (omitting the line number) and store it with the RETURN key.
- Type the LINE FEED key; the system terminates editing of the line, saving all remaining characters on that line.

When the EDIT operation is complete, normally by typing a LINE FEED or RETURN key, you can enter another program line or issue an immediate mode command.

1.7 STORING AND RELOADING PROGRAMS VIA PAPER TAPE

To store a program currently in memory onto paper tape using the ASR-33 Teletype paper tape punch, perform the following procedure:

1. Turn the teletype control knob to LINE.
2. Type TAPE on the terminal and then type the RETURN key. (This establishes TAPE mode and transfers system control to the paper tape punch.)
3. Turn the paper tape punch ON.
4. Type LIST on the terminal and type the RETURN key.
5. When punching is complete, turn the tape punch OFF.
6. Type KEY on the terminal and then type the RETURN key. (This transfers system control back to the terminal.) Typing CTRL/C also cancels TAPE mode and transfers control back to the terminal.

To store a program onto paper tape using the high speed paper tape punch, perform the procedure shown below. Do not use the high speed paper tape punch when the card reader is reading cards, as this can affect card reader operation.

1. Turn the terminal control knob to LINE.
2. Type PTP on the terminal and then type the RETURN key.
3. Turn the high speed punch ON.
4. When punching is complete, turn the punch OFF.
5. Type KEY on the terminal and then type the RETURN key. Typing CTRL/C also cancels the previous PTP command and transfers control back to the terminal.

To reload a program using the paper tape reader, perform the procedure shown below.

1. Erase the program currently in memory by typing SCRATCH on the terminal and then the RETURN key. (This is because loading a program from paper tape does not automatically erase the existing program in memory. Illogical results may occur unless the previous program is erased.)
2. Turn the teletype paper tape reader to FREE.
3. Turn the teletype control knob to LINE.
4. Insert the tape in the reader.
5. Type TAPE on the terminal and then type the RETURN key to establish TAPE mode.
6. Turn the teletype reader to START.
7. When the tape is read in, turn the teletype reader to FREE.
8. Type KEY on the terminal and then type the RETURN key. Typing CTRL/C also cancels TAPE mode and returns control to the terminal.

To reload a program using the high speed paper tape reader, perform the procedure shown below. Do not use the high speed paper tape reader when the card reader is reading cards, as this can affect card reader operation.

1. Erase the program currently in memory by typing SCRATCH on the terminal and then the RETURN key.
2. Turn the high speed paper tape reader ON.
3. Turn the terminal control knob to LINE.
4. Insert the tape in the reader.
5. Type PTR on the terminal and then type the RETURN key.
6. When the tape is read in, turn the paper tape reader OFF.
7. Type KEY on the terminal and then type the RETURN key. Typing CTRL/C also cancels the previous PTR command and transfers control back to the terminal.

1.8 ERASING A PROGRAM

1.8.1 SCRATCH and BYE Statements

To erase a program currently in memory, type the word SCRATCH and type the RETURN key.

Syntax: SCRATCH

Be careful when using the SCRATCH statement. This statement completely erases the program from memory. To run the program again, it must either be loaded into memory from another device or rewritten.

The statement, BYE, works identically to SCRATCH.

Syntax: BYE

1.8.2 DELETE Statements

Use the DELETE statement to erase portions of a program from memory.

Syntax: DELETE {1st line number{ ,2nd line number}}

Examples:

```
DELETE
DELETE 25
DELETE 80,110
```

When DELETE is executed without specifying the first or second line numbers, the entire program in memory is erased.

Specifying the first line number in the DELETE statement instructs the system to delete only that line.

When both the first and second line numbers are specified in the DELETE statement, separated by a comma, the system deletes both lines as well as all of the lines between them.

CHAPTER 2

CARD READER OPERATIONS

2.1 MARKING THE CARDS

The system accepts program statements and data from specially formatted BASIC mark sense cards provided by Digital. Each card is preprinted with 40 columns of small rectangular boxes. The column number is printed at the bottom of the card below each column. Figure 2-1 illustrates the BASIC card.

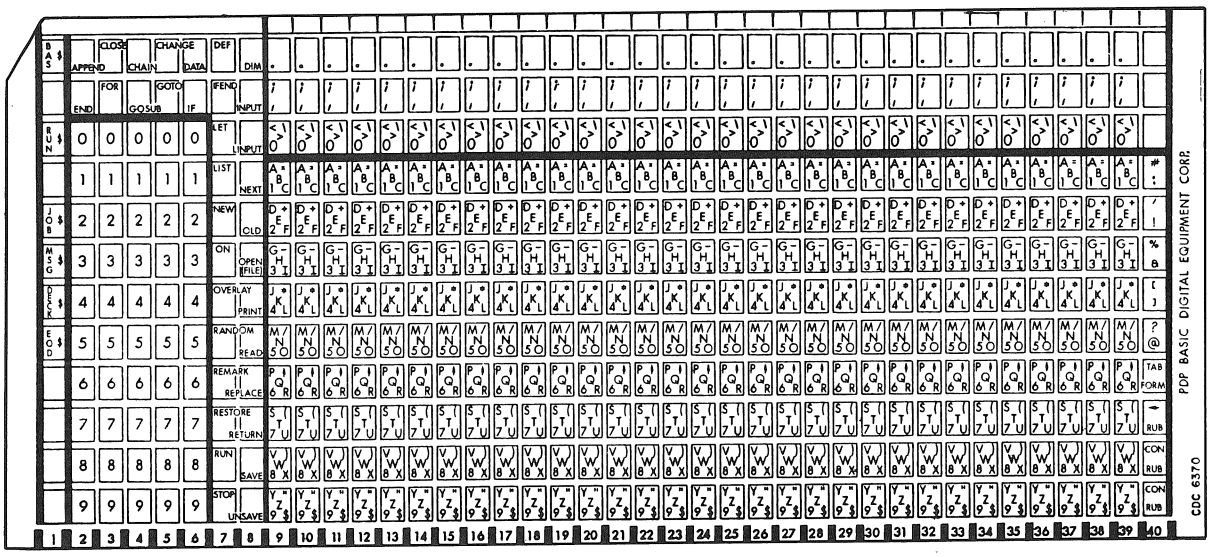


Figure 2-1
Educational Mark Sense Card

Statements are entered onto the card by marking the appropriate boxes with an ordinary soft-lead pencil (number 2 is ideal). Do not use a ball point pen or a felt tip marker.

Each box is marked with a single heavy line, drawn either vertically or diagonally. It is not necessary to fill the entire box. Avoid making stray marks, as the system reads all marks on the card — even the ones outside boxes. Stray marks cause the system to misread the card.

When two or more vertically adjacent boxes are to be marked, they can be marked with a single vertical stroke. Figure 2-2 shows examples of properly marked cards.

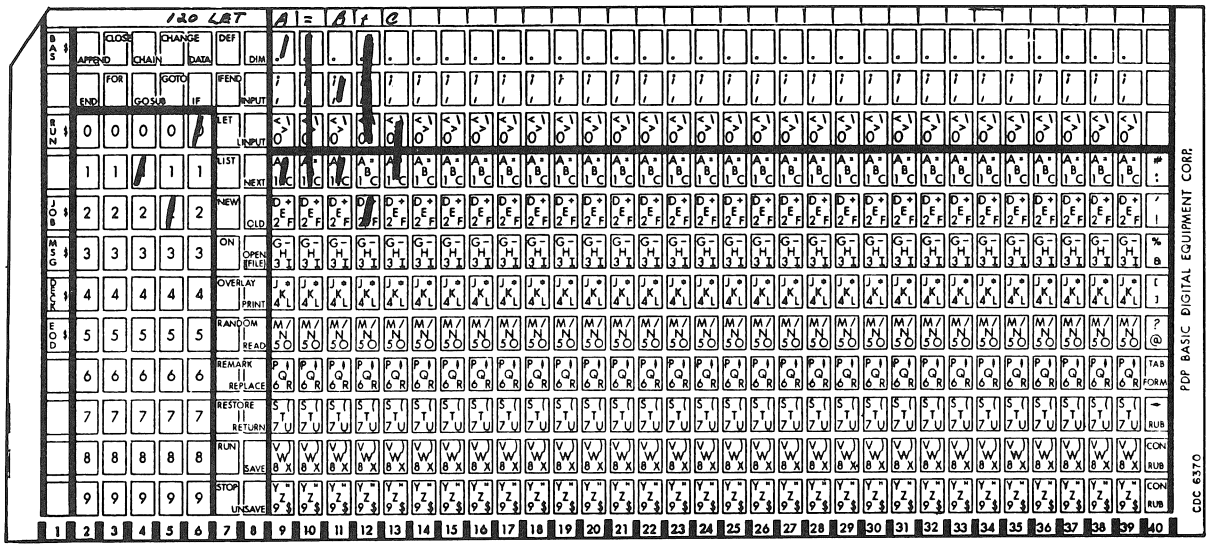


Figure 2-2a
120 LET A=B+C

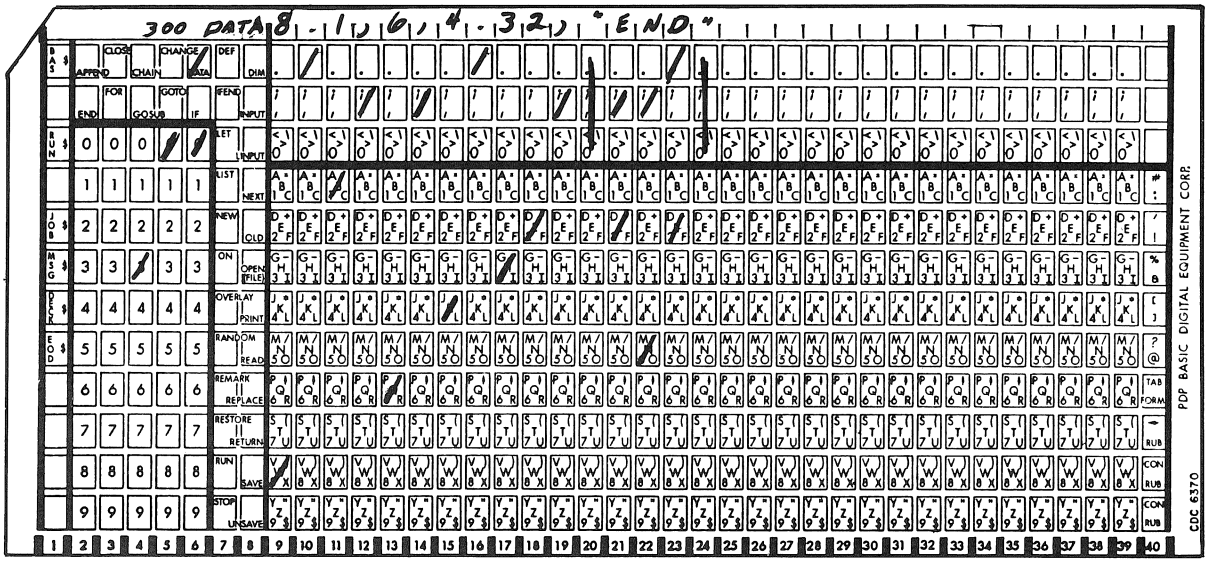


Figure 2-2b
300 DATA 8.1,6,4.32,"END"

The information on a card can be written out in the space provided at the top of the card. This helps identify program and data cards.

BASIC cards are divided into four areas or fields: the control command field, the line number field, the keyword field, and the character field. Each field is separated from an adjacent field by a thick line, as shown in Figure 2-3.

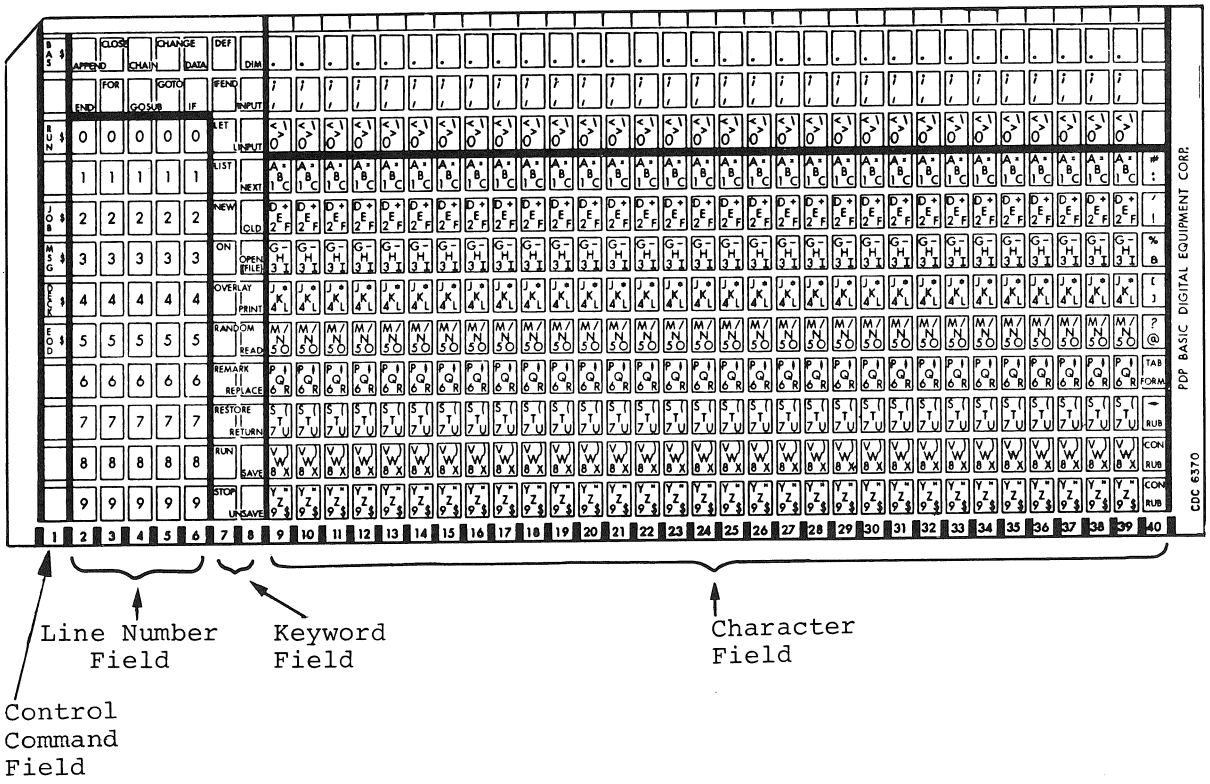


Figure 2-3
Card Field Layout

2.1.1 Control Command Field

Any one of the top three boxes in column 1 may be marked, but only one column 1 box can be marked per card. Figure 2-4 shows the control command field as it appears on the cards with the top three boxes defined.

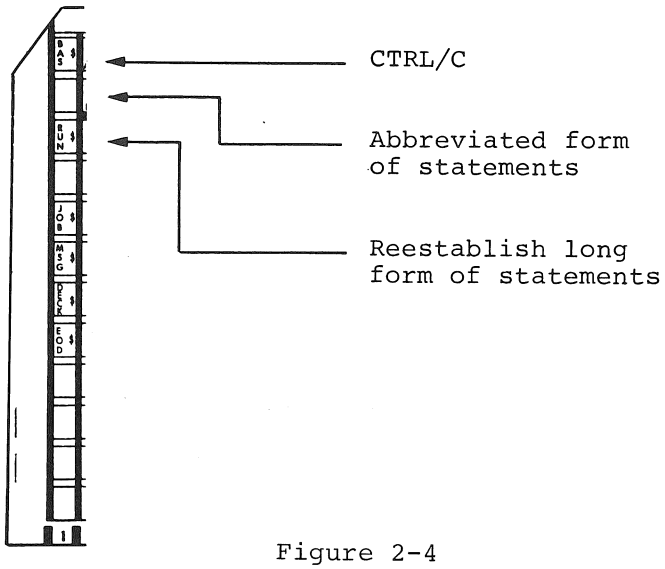


Figure 2-4
Control Command Field

The second box in column 1 instructs the system to read the marked boxes in the keyword field statement in the abbreviated form. The first three letters of each statement are used in this abbreviated mode of operation. The box marked PRINT, for example, is entered as PRI. This is a legal form of the statements in the EduSystem and it saves memory space. A listing of a program entered in this way shows the abbreviated form of all these statements.

The abbreviated mode is in effect until the third box in column 1 is marked or a CDR statement is executed to deassign the card reader (see Section 2.3), thereby disabling the mode.

2.1.2 Line Number Field

The line number field consists of five columns, each containing ten boxes numbered from 0 through 9. Line numbers up to and including 2046 can be marked in this field. To enter a line number on a card, mark the box in each column that corresponds to the desired digit. Any column that is not marked is ignored. Only one of the boxes corresponding to the digits 0-9 should be marked in any column. Figure 2-5 shows two ways to enter the line number 15.

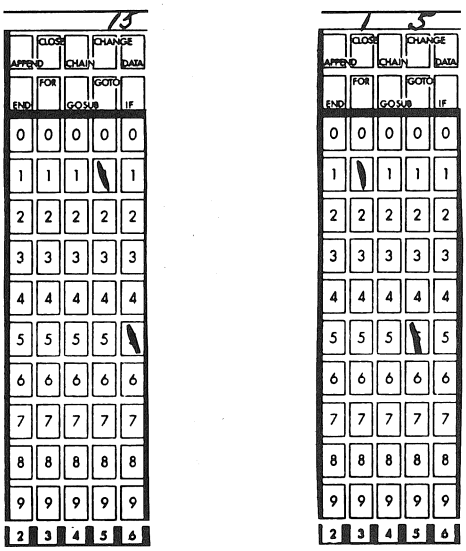


Figure 2-5
Line Number Field

2.1.3 Keyword Field

Each box in the keyword field contains a BASIC keyword. To select a keyword, simply mark the box that contains the desired keyword. These boxes are arranged in alphabetical order. Mark only one box in this field. Since the BASIC card is used with several dialects of BASIC, not all the keywords on the BASIC card apply to EduSystem BASIC. Figure 2-6 shows the keyword field.

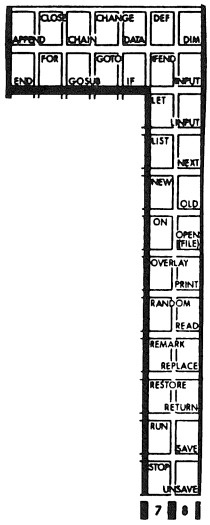


Figure 2-6
Keyword Field

Legal BASIC language keywords not shown in this field can usually be entered in the character field (see below).

2.1.4 Character Field

The character field is used to specify:

- arguments to control commands
- arguments to language keywords
- language keywords in addition to the keyword selected in the keyword field
- language keywords not appearing in the keyword field
- program data
- keyboard commands

The character field is divided into two subfields separated by a heavy horizontal line: the zone subfield and the character subfield.

The boxes in columns 9 through 39 are identical and contain all the alphanumeric characters and most of the punctuation characters. Boxes in column 40 contain additional punctuation and special characters.

To select any of the alphabetic characters or the dollar sign ($) character, two boxes must be marked: a character box and a zone box. First mark the box containing the desired character. Then determine the position of that character in its box (i.e., top-left, middle, bottom-right), and mark the zone box (top, middle, or bottom) that corresponds to the selected character's position. Figure 2-7 shows all the characters that can be selected in this way.

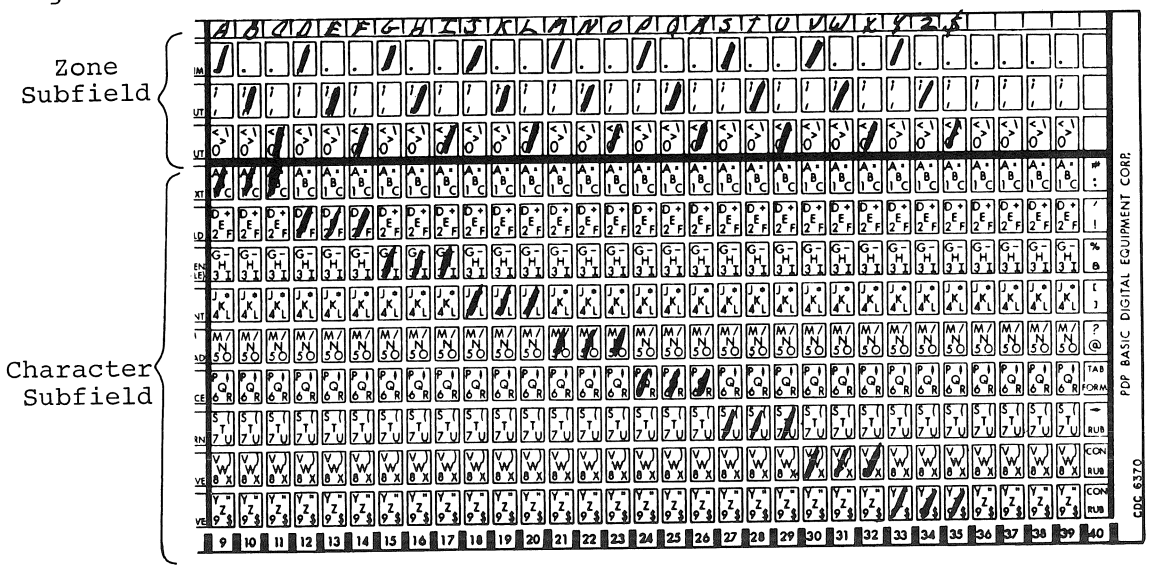


Figure 2-7
Selecting Alphabetic Characters and the Dollar Sign Symbol

The technique used for alphabetic characters is also used to select the special characters <, >, and ; in the zone field, as shown in Figure 2-8.

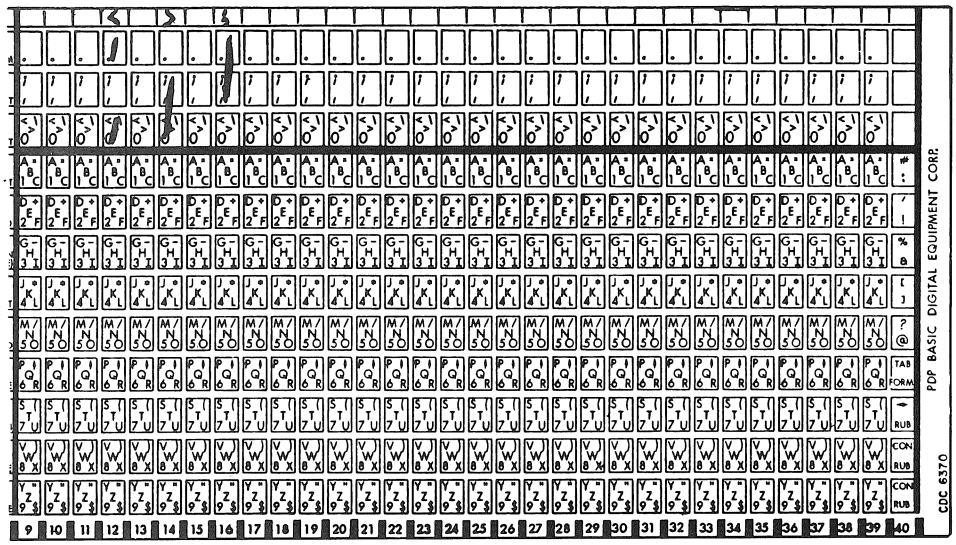


Figure 2-8
Selecting the Characters: <, >, and ;

To select the numeric characters, period (.) and comma (,), simply mark the box containing the character. Figure 2-9 shows these characters.

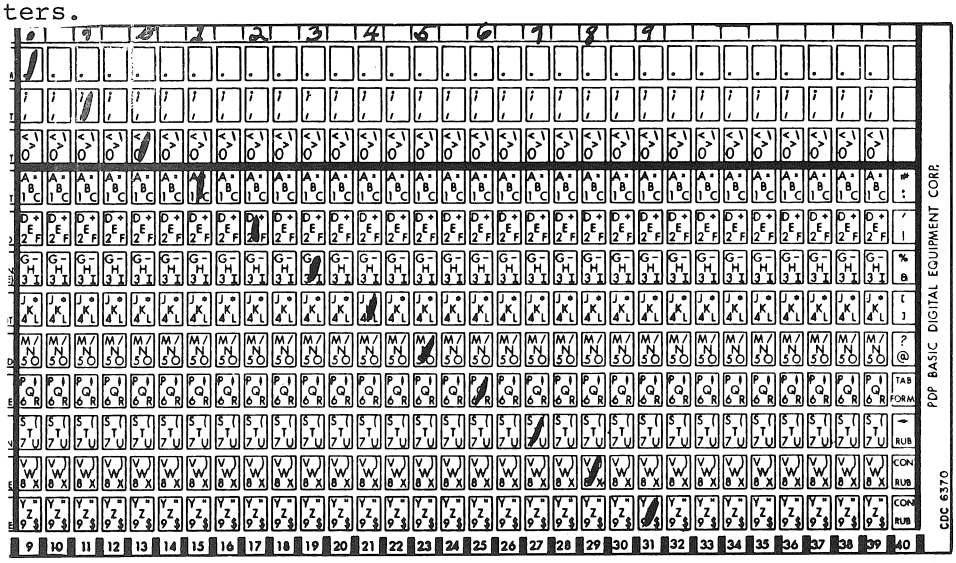


Figure 2-9
Selecting Numbers, Period and Comma

To select the characters located in the upper right corner of the boxes, mark the box containing the desired character and all zone boxes as shown in Figure 2-10.

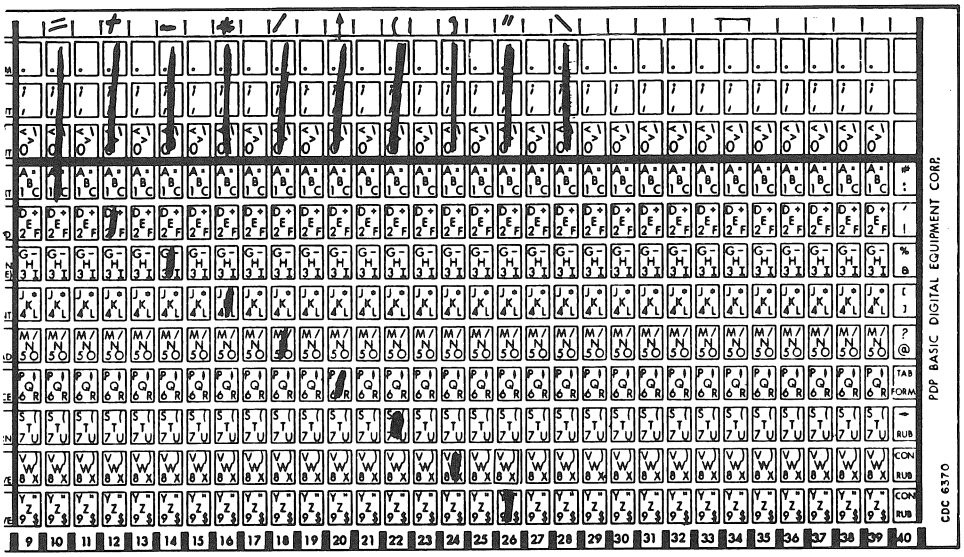


Figure 2-10
Selecting Special Characters

To select one of the special characters in column 40, first mark the corresponding box in the desired column. Then, if the character is in the top half of its box, mark the top and middle zone boxes. If the selected character is in the lower half of its box, mark the middle and lower zone boxes. Although these characters appear only in column 40, they can be marked in any column of the operand field, as shown in Figure 2-11. Simply use column 40 as a template.

Note that @, TAB, FORM, and ← cannot be interpreted by the system; question marks are printed to replace them. Character positions on the bottom of row 40, marked RUB (for rubout) and CON (for continue) perform special functions that are described below.

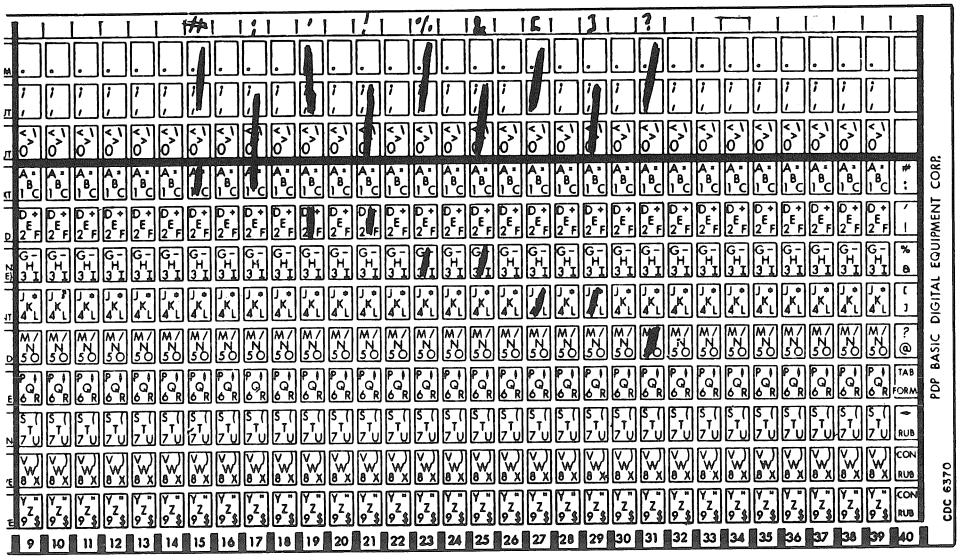
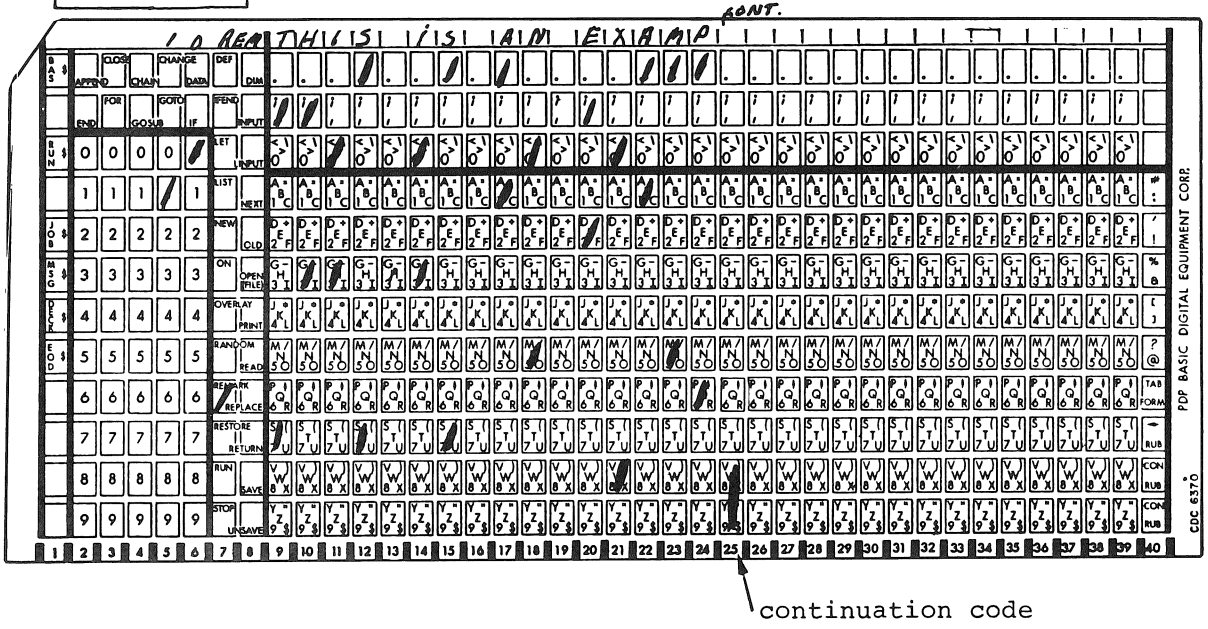


Figure 2-11
Selecting Special Characters in Card Column 40

Marking the bottom two boxes in any column specifies a continuation of the current statement onto the following card. Although the CON (for continuation) appears only in card column 40, the two bottom boxes in any column between 9 and 40 can be marked. Any marked columns that follow a continuation specification on that card are ignored, so be sure that CON is the last column you use on the card. Figure 2-12 shows an example of card continuation.

First Card



Continuation Card

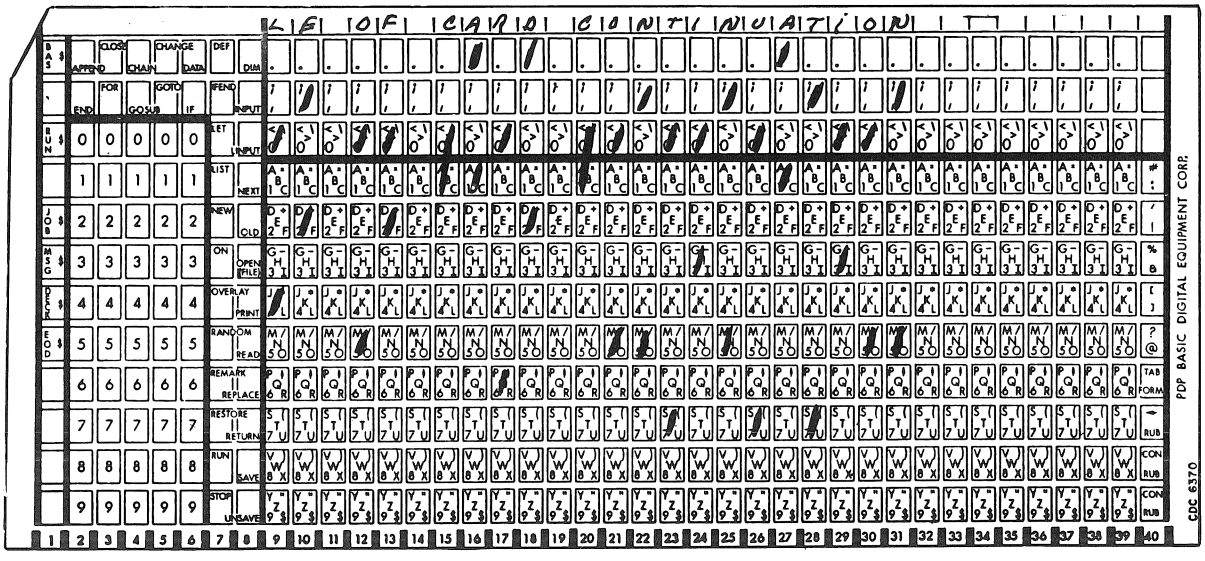


Figure 2-12
Specifying Card Continuation

Incorrectly marked cards can be corrected by using the RUB (for rubout) feature. Marking the bottom three boxes in any column causes the system to ignore any other character previously selected in that column. As with the other characters in column 40, RUB can be used in any column between 9 and 40 simply by using column 40 as a template. See Figure 2-13.

Errors can also be corrected by <u>completely</u> erasing the marks that are in error; the column can then be correctly marked. Although more risky than using the RUB feature, erasing is often more convenient, particularly when the error is in the middle of a fully marked card. Wherever possible, however, RUB should be used to correct errors.

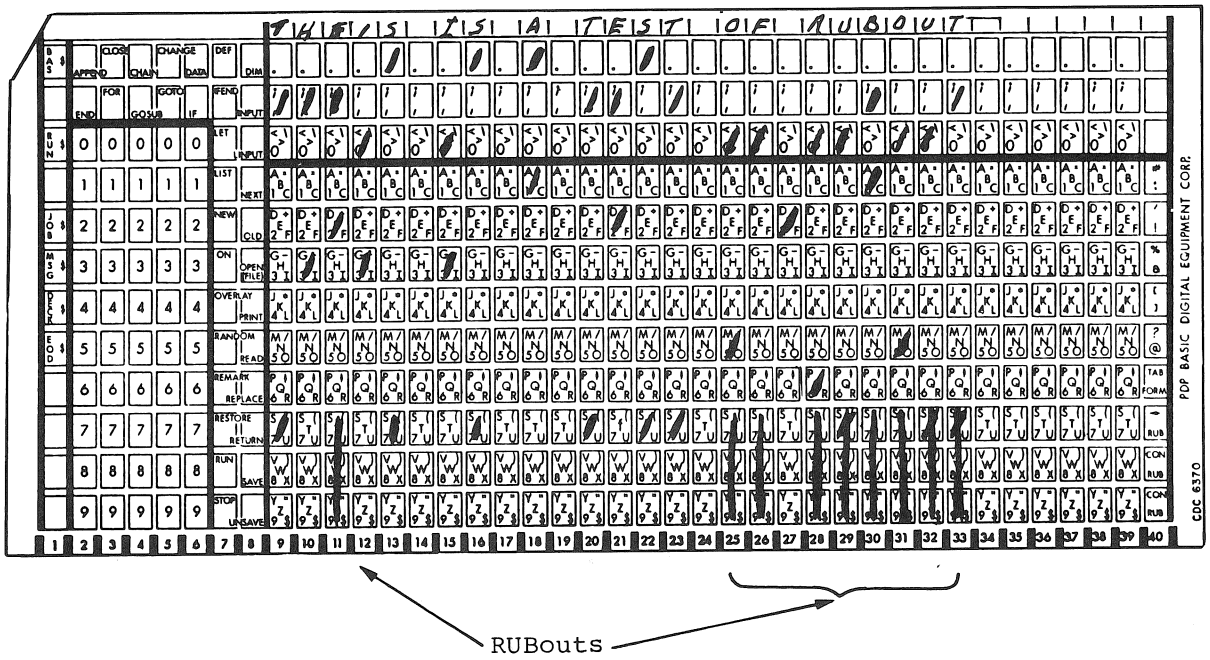


Figure 2-13
Correcting Errors with RUBout

2.2 USING THE CM8E CARD READER

This section describes the operating procedures for the CM8E optical mark card reader. Figure 2-14 shows the CM8E card reader.

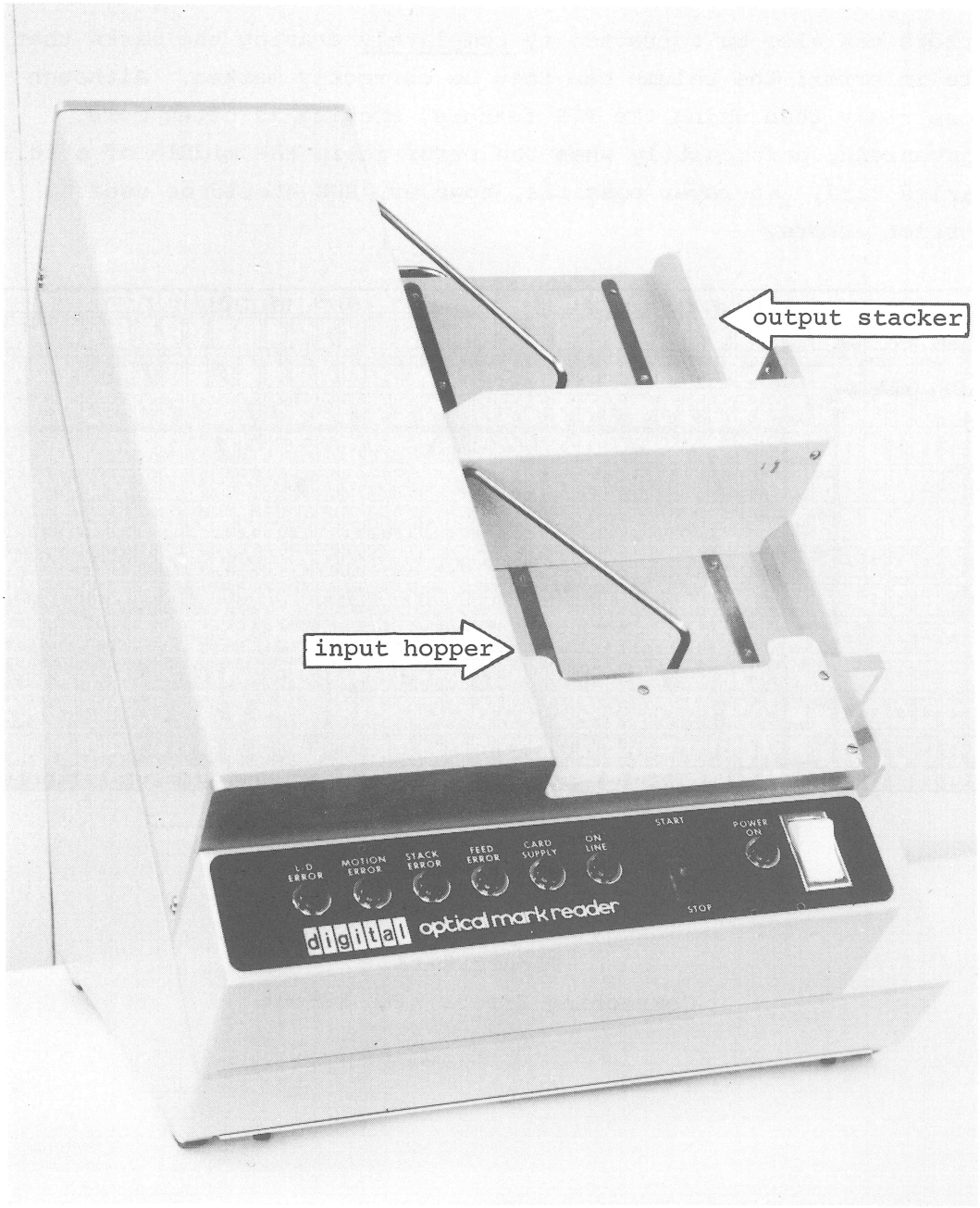


Figure 2-14
CM8E Card Reader

2.2.1 Controls and Indicators

The controls and indicators used to operate the CM8E card reader are shown in Figure 2-15 and listed in Table 2-1.

The control panel (see Figure 2-15), located at the lower edge of the front of the reader, contains the controls and indicators used for card reader operation.

Figure 2-15
CM8E Controls and Indicators

Table 2-1
CM8E Controls and Indicators

Control or Indicator	Type	Function
POWER ON switch	2-position rocker	When upper portion of switch is depressed, applies primary power to all card reader circuits except drive motor. Depressing lower portion of switch removes all power from reader.
POWER ON indicator	green light	When lit, indicates primary ac power has been applied to the reader.

Table 2-1 (Cont.)
CM8E Controls and Indicators

Control or Indicator	Type	Function
START switch	pushbutton	When depressed, starts main drive motor and resets all error indicators provided the error condition has been corrected. Upon releasing switch and allowing motor to come up to speed, the reader can start processing cards. Lights ON LINE indicator.
STOP switch	pushbutton	When depressed, stops reader operation by stopping main drive motor. If this switch is depressed while card reading is in process, the current card cycle is completed before the motor stops. Extinguishes ON LINE indicator.
ON LINE indicator	green light	When lit, indicates the reader is on-line and able to accept commands. This light comes on when the START switch is depressed. This light goes out when the STOP switch is depressed or when an error condition occurs.
CARD SUPPLY indicator	red light	When lit, indicates either the input hopper is empty or that the output stacker is full. Condition must be corrected by the operator before card reading can continue. Stops reader operation and extinguishes ON LINE indicator.

Table 2-1 (Cont.)
CM8E Controls and Indicators

Control or Indicator	Type	Function
FEED ERROR indicator	red light	When lit, indicates the card reader failed to move a card into the read station. Stops card reader operation and extinguishes ON LINE indicator.
STACK ERROR indicator	red light	When lit, indicates a card was not properly delivered to the output stacker after the read operation. Stops card reader operation and extinguishes ON LINE indicator.
MOTION ERROR indicator	red light	When lit, indicates a card jam in the read station. Stops card reader operation and extinguishes ON LINE indicator.
L.D. ERROR indicator	red light	When lit, indicates the card failed to pass the light/dark check at the read station. Stops card reader operation and extinguishes ON LINE indicator.

2.2.2 Card Handling Procedures

The following paragraphs present the recommended procedures for loading the input hopper, unloading the output stacker, and correcting error conditions.

To Load Cards:

Step	Procedure
1	Pull the hopper follower back with one hand and begin loading card decks into the hopper. Make certain that the first card to be read is placed at the front, face down, column 1 to the left.
2	Continue placing cards into the input hopper until it is loosely filled.

> CAUTION
>
> Do not pack the input hopper so full that the air from the blower cannot riffle the cards properly. If the cards are packed too tightly, it impairs proper operation of the vacuum picker.

3	Cards may continue to be loaded while the reader is operating provided tension is maintained on the front portion of the deck as cards are added to the rear. Additional cards should not be loaded, however, until the hopper is approximately 1/2 to 2/3 empty.

> CAUTION
>
> When maintaining tension on the card deck, use just enough pressure to maintain the riffle action to prevent card damage and jamming of the reader.

4	Normally, all cards are moved through the reader into the stacker. If, however, it is necessary to remove cards from the input hopper, simply pull back the follower and remove the card deck.

To Unload Cards:

To unload cards from the output stacker, pull the stacker follower back with one hand and remove the card deck from the stacker, being careful to maintain the order of the deck. The stacker may be unloaded while the cards are being read.

2.2.3 On-Line Operation

The model CM8E reader has no capability for off-line operation. Verification of satisfactory operation is determined by successful on-line operation.

Step	Procedure
1	Make certain that the card reader is ON; the POWER ON indicator should be lit.
2	Make certain the output stacker is empty. Load the input hopper with the cards to be read.
3	Press the START switch. Observe that the green ON LINE indicator lights. The card reader is now on-line.
4	If the card reader goes off-line due to an error, it can be placed back on-line by correcting the error and then depressing the START switch.
5	To go off-line, simply depress the STOP switch.

2.3 USING THE CM8F CARD READER

This section describes the operating procedures for the CM8F optical mark card reader. Figure 2-16 shows the CM8F card reader.

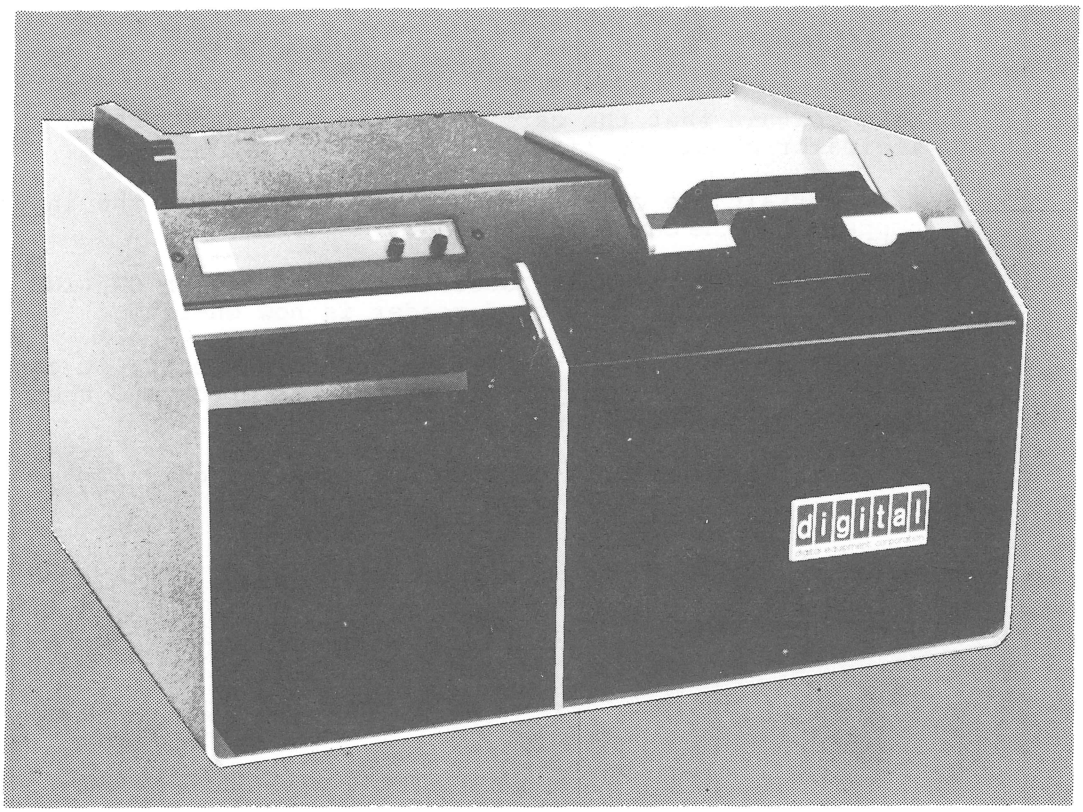


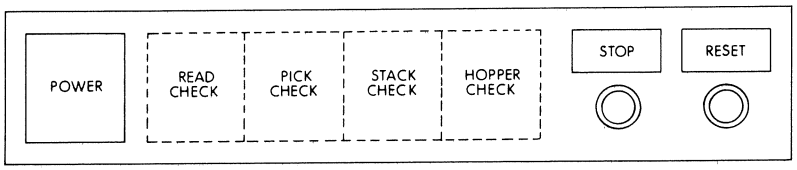
Figure 2-16
CM8F Card Reader

2.3.1 Controls and Indicators

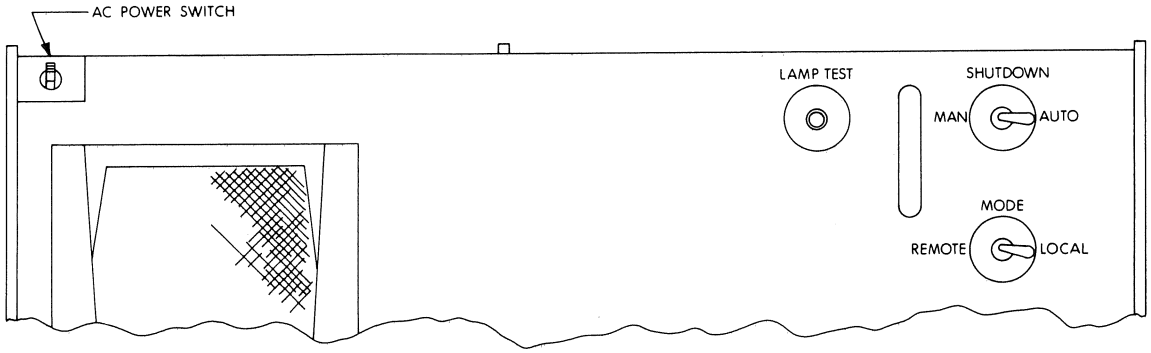
The controls and indicators used to operate the CM8F card reader are shown in Figure 2-17 and listed in Tables 2-2 and 2-3.

The control panel (see Figure 2-17), located at the upper left-hand corner of the front of the reader, contains the controls and indicators used for normal on-line/off-line operation of the card reader.

The rear panel, located at the upper right-hand corner of the back of the reader, is used for initial setup of the system and for maintenance purposes.



FRONT CONTROL PANEL



REAR PANEL

Figure 2-17
CM8F Controls and Indicators

Table 2-2
CM8F Front Panel Controls and Indicators

Control or Indicator	Type	Function
POWER switch	alternate-action pushbutton/indicator switch	Controls application of all power to the card reader. When indicator is off, depressing switch applies power to reader and causes associated indicator to light. When indicator is lit, depressing switch removes all power from reader and causes indicator to go out.
READ CHECK indicator	white light	When lit, indicates the card just read may be torn on the leading or trailing edges. Because READ CHECK indicates an error condition, whenever this indicator is lit, it causes the card reader to stop operation and extinguishes the RESET indicator.
PICK CHECK indicator	white light	When lit, indicates the card reader failed to move a card into the read station after it received a READ COMMAND from the controller. Stops card reader operation and extinguishes RESET indicator.
STACK CHECK indicator	white light	When lit, indicates the previous card was not properly seated in the output stacker and therefore may be badly mutilated. Stops card reader operation and extinguishes RESET indicator.

Table 2-2 (Cont.)
CM8F Front Panel Controls and Indicators

Control or Indicator	Type	Function
HOPPER CHECK indicator	white light	When lit, indicates either the input hopper is empty or the output stacker is full. In either case, the operator must manually correct the condition before card reader operation can continue.
STOP switch	momentary pushbutton/indicator switch (red light)	When depressed, immediately lights and drops the READY line, thereby extinguishing the RESET indicator. Card reader operation then stops as soon as the card currently in the read station has been read. This switch has no effect on the system power, it only stops the current operation.
RESET switch	momentary pushbutton/indicator switch (green light)	When depressed and released, clears all error flip-flops and initializes card reader logic. Associated RESET indicator lights to indicate that the READY signal is applied to the controller. The RESET indicator goes out when the STOP switch is de-pressed or when an error indicator (READ CHECK, PICK CHECK, STACK CHECK, or HOPPER CHECK) lights.

Table 2-3
CM8F Rear Panel Controls

Control	Type	Function
LAMP TEST switch	pushbutton	When depressed, illuminates all indicators on the front control panel to determine if any of the indicator lamps are faulty.
SHUTDOWN switch	2-position toggle	Controls automatic operation of the input hopper blower. MAN position - blower operates continuously. AUTO position - causes the blower to shut down automatically when the input hopper is emptied. Blower automatically restarts when cards are loaded into the hopper and the RESET switch is depressed. Blower activates approximately three seconds after RESET is depressed.
MODE switch	2-position toggle	Permits selection of either on-line or off-line operation. LOCAL position - removes the READ COMMAND input from the controller to allow the operator to run the reader off-line by using the RESET and STOP switches on the front control panel. REMOTE position - enables the READ COMMAND input from the controller to allow normal on-line operation under program control once RESET is depressed.

2.3.2 Card Handling Procedures

The following paragraphs present the recommended procedures for loading the input hopper, unloading the output stacker, and correcting error conditions.

To Load Cards:

Step	Procedure
1	Pull the hopper follower back with one hand and begin loading card decks into the hopper. Make certain that the first card to be read is placed at the front, face down, column 1 to the left.
2	Continue placing cards into the input hopper until it is loosely filled.

> CAUTION
>
> Do not pack the input hopper so full that the air from the blower cannot riffle the cards properly. If the cards are packed too tightly, it impairs proper operation of the vacuum picker.

3	Cards may continue to be loaded while the reader is operating provided tension is maintained on the front portion of the deck as cards are added to the rear. Additional cards should not be loaded, however, until the hopper is approximately 1/2 to 2/3 empty.

> CAUTION
>
> When maintaining tension on the card deck, use just enough pressure to maintain the riffle action to prevent card damage and jamming of the reader.

4	Normally, all cards are moved through the reader into the stacker. However, if it is necessary to remove cards from the input hopper, simply pull back the follower and remove the card deck.

To Unload Cards:

To unload cards from the output stacker, pull the stacker follower back with one hand and remove the card deck from the stacker, being careful to maintain the order of the deck. The stacker may be unloaded while the cards are being read.

2.3.3 Off-Line Operation

Off-line operation of the CM8F card reader is used primarily for setting up and checking reader operation prior to switching to on-line use; for correcting error conditions; and for maintenance tests. When placed off-line, the reader can be operated locally from the control panels. The following procedure is used to energize the reader and check off-line operation prior to switching to on-line operation.

Step	Procedure
1	Make certain the ac power cord is plugged in and the circuit breaker on the rear base panel of the reader is in the ON position.
2	Set MODE switch to LOCAL position.
3	Set SHUTDOWN switch to AUTO position.
4	Depress POWER switch to energize reader. Note that POWER indicator lights but blower does not come on.
5	Depress LAMP TEST switch and observe that all front panel indicators are lit.
6	Load a card deck into the input hopper.
7	Depress RESET switch and observe that associated green indicator comes on. After approximately 3 seconds, cards should be picked and moved through the read station into the output stacker.
8	When the input hopper is empty, observe that the HOPPER CHECK indicator lights, the green RESET indicator goes out, and the red STOP indicator lights.
9	The card reader may now be operated locally or switched to on-line operation.

2.3.4 On-Line Operation

Step	Procedure
1	Make certain that the card reader is operational by performing the procedure given in Section 2.3.3.
2	Make certain the output stacker is empty. Load the input hopper with the cards to be read.
3	Set the MODE switch to REMOTE.
4	Depress the RESET switch and observe that the associated green indicator lights. The card reader is now on-line.
5	If the card reader goes off-line because of an error alarm, it can be placed back on-line by correcting the error and then depressing the RESET switch.
6	To go off-line at any time, depress the STOP switch.

2.4 RUNNING PROGRAMS WITH THE CARD READER

This section explains how to run programs with one of the card readers described in previous sections. Information on entering data from the card reader is also supplied.

Finally, this section assumes that you are familiar with marking educational mark sense cards; they should be used, one statement per card, in the examples shown in this section.

2.4.1 CDR: Assigning the Card Reader

You can instruct the card reader, from the terminal, to begin reading cards. First be sure power is supplied to the card reader (the light marked POWER should be on). In immediate mode, simply type the command,

```
CDR↵
READY
```

This command transfers control to the card reader. Be sure no one else is using the card reader, or you may lose program control to another user. When CDR is executed again (this time from a marked card in the card reader), it transfers control back to the terminal.

Once you have executed CDR from the terminal, load a program on marked cards onto the card reader. You can also include cards with immediate mode statements; simply leave the line number field on these cards blank. For example, mark cards in the following manner:

```
SCRATCH

10 PRINT "THIS IS A TEST"
20 A = 33
30 PRINT A
40 END

LIST

RUN

DELETE 10

LIST

RUN

SCRATCH

LIST

CDR
```

Now press the START button on the card reader. The following listing shows what is printed on the terminal.

Terminal Output

```
SCRATCH

READY
10 PRINT "THIS IS A TEST"
20 A=33
30 PRINT A
40 END
LIST

  10 PRINT "THIS IS A TEST"
  20 A=33
  30 PRINT A
  40 END

READY
RUN
THIS IS A TEST
 33

READY
DELETE10

READY
LIST

  20 A=33
  30 PRINT A
  40 END

READY
RUN
 33

READY
SCRATCH

READY
LIST


READY
CDR

READY
```

In the above example, notice that the card reader echoes the information on each card as it is read. It executes the statement(s) on each card completely before reading the next card. READY is automatically printed after each immediate mode statement is executed. Once all program lines are read by the card reader, the program is in computer memory. Subsequent immediate mode statements executed from the card reader or the terminal (if it has control) can affect the program. In the above case, for example, the immediate mode statement, DELETE 10, actually erases line 10 from computer memory. This is verified by a subsequent LIST and RUN; line 10 does not appear and is not executed.

Finally, notice that a CDR statement is on the last card. This transfers control back to the terminal. If you forget to include this card at the end of the card deck (i.e., after the last statement you want executed), simply type CTRL/C from the terminal. CTRL/C automatically deassigns the card reader and transfers control back to the terminal. CTRL/C can also be executed from a card, by marking the top left box (the one labeled BAS) on a card. Execution of this CTRL/C is identical to immediate mode execution from the terminal.

In summary, there is only one way to transfer control to the card reader: executing CDR from the terminal. There are three ways to transfer control back to the terminal: executing CDR from a marked card in the card reader; executing CTRL/C from a marked card in the card reader; executing CTRL/C from the terminal.

2.4.2 Entering Data from Cards

Normally, string and numeric data can be entered to a program with the use of DATA statements (see Section 1.3.13) within the program. INPUT statements are more convenient when using the card reader because one program can be run over and over again using different input values.

For example, consider the following program, written on a terminal:

```
10 REMARK  THIS PROGRAM AVERAGES FIVE NUMBERS
20 T = 0
30 FOR M = 1 TO 5
40 INPUT X
50 T = T + X
60 NEXT M
70 PRINT "AVERAGE IS" T/5
80 END
```

Now place six cards (or only two cards, if you use commas between data elements) in the input hopper of the card reader. These six cards should have the information shown below, in this order:

```
RUN
3
6
8
-1
9
```

Execute CDR from the terminal and press the START button on the card reader.

```
CDR
READY
RUN
? 3
? 6
? 8
? -1
? 9
AVERAGE IS 5

READY
```

Then type CTRL/C at the terminal to return control to the terminal.

Another method by which you can enter data elements from the card reader is by programming CDR commands. (Use line numbers for this purpose.) Be sure to place one CDR statement immediately before the INPUT and also one CDR statement immediately after the INPUT in your program. For example, consider the following program, written at a terminal:

```
10 REMARK  THIS PROGRAM AVERAGES FIVE NUMBERS
15 CDR
20 INPUT A,B,C,D,E
30 CDR
40 PRINT "AVERAGE IS" (A+B+C+D+E)/5
50 END
```

Now place five cards (or only one card, if you use commas between data elements) in the input hopper of the card reader and press the START button. Use the same data elements in this example as in the previous one:

```
3
6
8
-1
9
```

Execute RUN from the terminal.

```
RUN
? 3,6,8,-1,9
AVERAGE IS 5

READY

CDR

READY
```

In the above example listing, the five data elements were entered on one card, separated by commas.

2.4.3 Entering Data from the Terminal

You can instruct the card reader to read a program on cards and execute it, as discussed in Section 3.4.1. It is often convenient to instruct the computer to accept data from the terminal, although the program itself is entered by cards. To halt program execution until data values are entered by the terminal, use programmed CDR commands as shown below. Each statement is marked on a separate card. Once you have marked these cards, place them in the input hopper of the card reader and press the START button.

```
10 REMARK  ENTERING DATA FROM A TERMINAL
20 PRINT "WHAT IS YOUR NUMBER";
30 CDR
40 INPUT Y
50 CDR
60 PRINT "THANK YOU." Y "IS MY FAVORITE"
70 END

RUN
```

Notice the CDR commands on both sides of line 40, INPUT Y. The first CDR command transfers control to the keyboard and the second returns control to the card reader.

Start the card reader by executing CDR from the terminal.

```
CDR↵

READY
10 REM ENTERING DATA FROMA TERMINAL
20 PRINT "WHAT IS YOUR NUMBER";
30 CDR
40 INPUT Y
50 CDR
60 PRINT "THANK YOU." Y "IS MY FAVORITE"
70 END
RUN
WHAT IS YOUR NUMBER? 2↵
THANK YOU.  2 IS MY FAVORITE

READY
```

Then type CTRL/C at the terminal to return control to the terminal. Notice that the program halts at line 40, awaiting your value for Y to be input from the terminal. Once this is accomplished, line 50 transfers control back to the card reader to continue program execution.

CHAPTER 3

SYSTEM MANAGER INFORMATION

3.1 LOADING THE EDUSYSTEM 20

EduSystem 20 is usually loaded from paper tape, using the Read-In Mode (RIM) Loader (see Section 3.2). If your system includes OS/8, however, you can load EduSystem 20 from an OS/8 device (see Appendix D for instructions on saving EDU20 on an OS/8 device).

Version A of EduSystem 20 is supplied on paper tape, order number DEC-08-ED20C-A-PB1. This version does not include provisions for card reader operation. It requires a minimum of 8K memory.

Version B of EduSystem 20 is supplied on paper tape, order number DEC-08-ED20C-A-PB3. This version allows card reader operation and requires a minimum of 12K memory.

3.2 READ-IN MODE LOADER

The Read-In Mode (RIM) Loader is the first program loaded into an EduSystem computer.[1] This program is loaded by toggling 17 instructions into core memory using the console SWITCH REGISTER (SR). The RIM Loader instructs the computer to receive and store, in core, data punched on paper tape in RIM coded format -- primarily the EduSystem system tapes.

---

[1]The RIM Loader is not needed if the EduSystem has a hardware bootstrap.

There are two RIM Loader programs: one is used when the input is to be from the low-speed (Teletype) paper tape reader; the other is used when input is to be from the high-speed paper tape reader. The locations and corresponding instructions for both programs are listed in Table 3-1. The procedure for loading (toggling) the RIM program into core is illustrated in Figure 3-1. The RIM Loader is loaded into field zero of core.

Table 3-1
RIM Loader Programs

Location	INSTRUCTION Low-Speed	High-Speed
7756	6032	6014
7757	6031	6011
7760	5357	5357
7761	6036	6016
7762	7106	7106
7763	7006	7006
7764	7510	7510
7765	5357	5374
7766	7006	7006
7767	6031	6011
7770	5367	5367
7771	6034	6016
7772	7420	7420
7773	3776	3776
7774	3376	3376
7775	5356	5357
7776	0000	0000

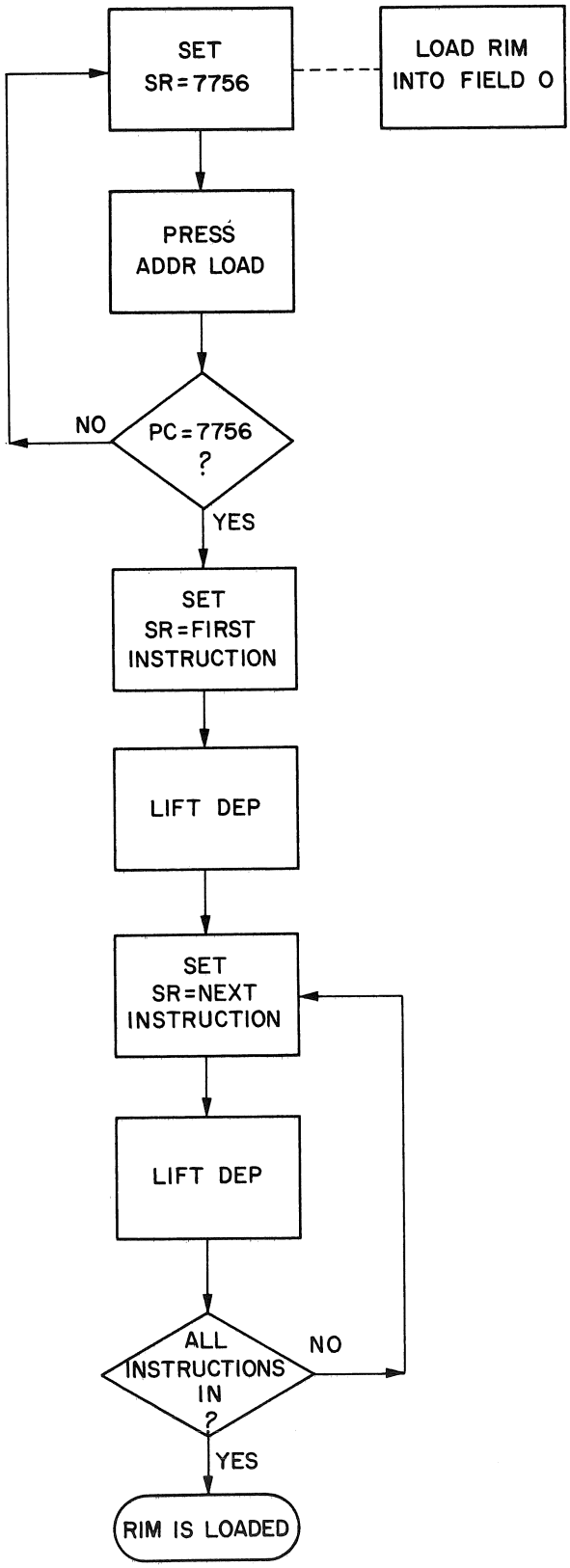


Figure 3-1
Loading the RIM Loader

After RIM has been loaded, it is good programming practice to verify that all instructions were stored properly. This can be done by performing the steps illustrated in Figure 3-2, which also shows how to correct an incorrectly stored instruction.

When loaded, the RIM Loader occupies absolute locations 7756 through 7776. EduSystems do not use the RIM locations; therefore, RIM need not be reloaded unless the contents of the RIM locations have been altered by the system manager.

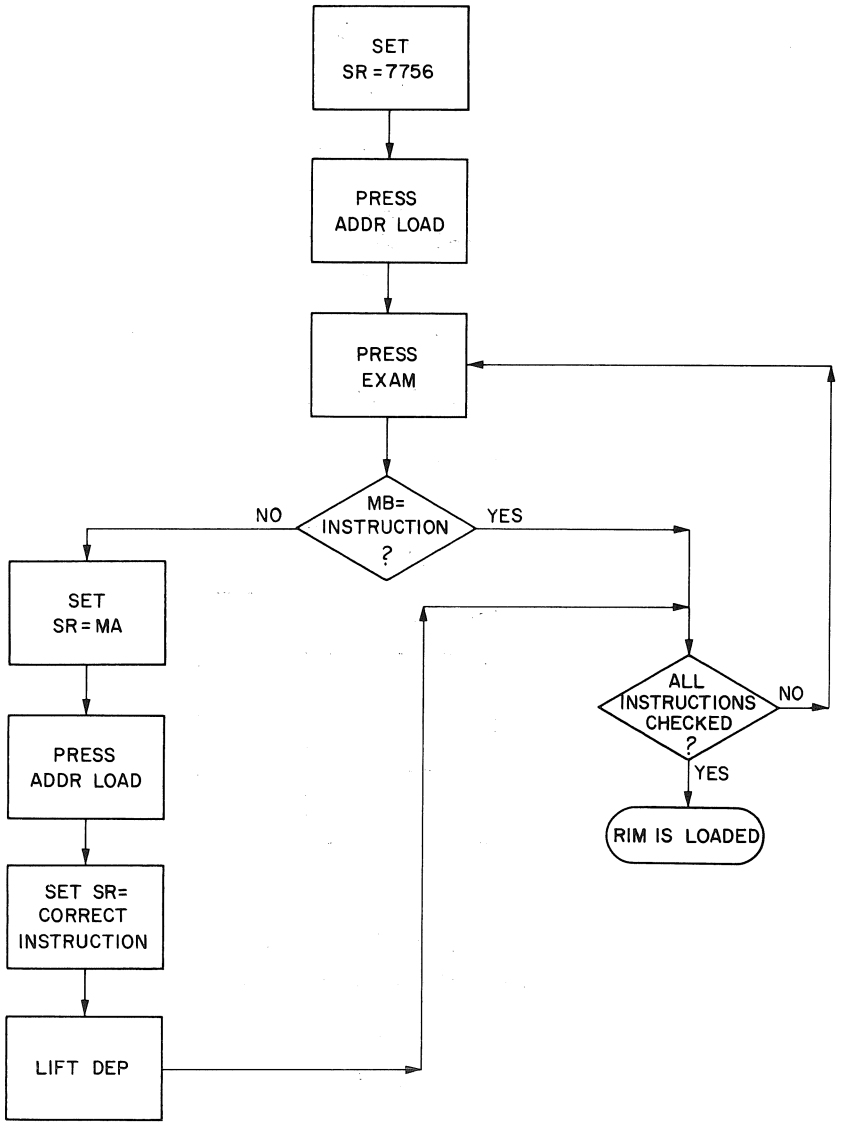


Figure 3-2
Checking the RIM Loader

After RIM has been loaded and verified, insert the paper tape (DEC-08-ED20C-A-PB1 for version A; DEC-08-ED20C-A-PB3 for version B) in the paper tape reader. Leader/Trailer code should be positioned over the reader sensors.

Be sure the Switch Register contains zeros in every position. Press the key marked EXTD ADDR LOAD.

Set the Switch Register to 7756 and press the key marked ADDR LOAD. Now verify that the high speed reader is ON or the Teletype paper tape reader control knob is set to LINE. Press the keys marked CLEAR and CONT.

If you are using the high speed reader, the tape will begin reading in at this point. On the Teletype paper tape reader, set the reader switch to START. The tape will begin reading in.

When finished loading, the system prints the following message for version A:

```
EDUSYSTEM 2Ø BASIC
ED2ØC-A-VAØ4.ØØ
```

Similarly, for version B, the system prints:

```
EDUSYSTEM 2Ø BASIC
ED2ØC-A-VBØ4.ØØ
```

3.3 ANSWERING SYSTEM DIALOG

When the system has been activated correctly it begins to ask certain questions which you must answer to establish the system configuration. When an invalid response is made to any of the system's questions, an error message (INVALID RESPONSE) is printed and the question is repeated. The carriage return key should not be typed after you type a character, since the system enters your response and continues program execution immediately. The first question is:

```
NUMBER OF USERS (1 TO 8)?
```

Respond with a single digit from 1 to 8, depending on the number of terminals to be used. (The console terminal counts as the single user.)

In Version B, the next question is:

```
MARK SENSE CARD READER USED (Y OR N)?
```

Answering N to the above question causes the system to proceed to the next question. A Y response, however, prompts the system to request the user numbers of legal terminals. For example, the sample dialog shown below informs the system that users 1, 2 and 3 are the only ones allowed to use the card reader.

```
MARK SENSE CARD READER USED (Y OR N)?Y
CAN ALL USERS USE THE CARD READER (Y OR N)?N
LEGAL USER?1 MORE (Y OR N)?Y
LEGAL USER?2 MORE (Y OR N)?Y
LEGAL USER?3 MORE (Y OR N)?N
```

The next question asked is:

```
PDP-8/L COMPUTER (Y OR N)?
```

Type Y if the EduSystem 20 computer is a PDP-8/L, N if not. An N response to this question prompts the next question:

```
STANDARD REMOTE TERMINAL CODES (Y OR N)?
```

BASIC is asking for a PT08 or KL8E device code for each terminal to be used (excluding the console terminal). Standard PT08 or KL8E device codes are 40, 42, 44, 46, 50, 52, and 54. When a system using PT08 or KL8E interface units is first installed, you determine the specific device code for each terminal and label each terminal with its specific device code. If device codes are standard, respond with Y to this question and BASIC assumes the standard device codes and continues the dialog. If device codes are not standard, enter N. The system then asks which codes are to be used for each terminal, in order, up to the number of users you specified in the first question. The console terminal's code cannot be changed. The following example shows the dialog for a five-user system, when terminals' codes are set to 30, 32, 34, and 36 respectively.

```
STANDARD REMOTE TERMINAL CODES (Y OR N)?N
TERMINAL #1 DEVICE CODE?30
TERMINAL #2 DEVICE CODE?32
TERMINAL #3 DEVICE CODE?34
TERMINAL #4 DEVICE CODE?36
```

The system then asks:

```
ANY UNUSED TERMINALS (Y OR N)?
```

Typing N prompts the system to ask the next question, but typing Y allows the system to ask which terminals are unused. The following example shows one unused terminal. Notice that the device codes must be used, rather than the user numbers.

```
DEVICE CODE?44 MORE (Y OR N)?N
```

The next two questions asked are:

```
DO YOU HAVE A HIGH SPEED PUNCH (Y OR N)?
DO YOU HAVE A HIGH SPEED READER (Y OR N)?
```

The next question asked is:

```
STANDARD USER STORAGE ALLOCATION?
```

The above question requires you to decide whether to partition the available core equally among the users on the EduSystem 20. If you respond Y to this question, BASIC determines the size of the core memory available and divides it equally among the users, then ends the dialog. If N is the response, BASIC determines the amount of available core storage and prints the highest core field according to the following:

```
Field 7 - 32K core memory
Field 6 - 28K core memory
Field 5 - 24K core memory
Field 4 - 20K core memory
Field 3 - 16K core memory
Field 2 - 12K core memory
Field 1 -  8K core memory
```

For explanation purposes, the following dialog is written for a 16K, 5-user EduSystem 20. The available core is to be allocated as follows:

```
User 1 - 16 blocks (user 1 is the console terminal)
User 2 -  6 blocks
User 3 -  5 blocks
User 4 -  5 blocks
User 5 -  7 blocks
```

Each core field, except field 1, contains 16 blocks. (Field 1 contains 7 blocks in version B and 10 blocks in version A.) A core field may be divided among several users, but no user may be allotted blocks in more than one core field. To determine the number of blocks, BASIC prints the following dialog and the system manager answers as shown:

```
STANDARD USER STORAGE ALLOCATION?N
FIELD 3
THERE ARE 16 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #1 WILL BE HOW MANY BLOCKS?16
FIELD 2
THERE ARE 16 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #2 WILL BE HOW MANY BLOCKS?6
THERE ARE 10 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #3 WILL BE HOW MANY BLOCKS?5
THERE ARE 05 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #4 WILL BE HOW MANY BLOCKS?5
FIELD 1
THERE ARE 07 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #5 WILL BE HOW MANY BLOCKS?7
```

When all responses have been entered, BASIC prints:

```
IS THE ABOVE CORRECT (Y OR N)?
```

If an incorrect response was made, answer N and BASIC begins the dialog again. A response of Y ends the dialog and BASIC prints:

```
END OF DIALOGUE

READY
```

BASIC prints READY on each of the terminals associated with the specified device codes.

APPENDIX A

ERROR MESSAGES

The EduSystem checks each statement in a program or in immediate mode before executing it. If a statement cannot be executed, the system prints an error code and a line number (if available) in which the error is found, on your terminal. Procedures used to correct errors can be found in Section 1.6.

Message	Explanation
WHAT?	Immediate mode statement or command not understood. It does not begin with a line number and is not a valid system command.
ERROR 1	Log of negative or zero number was requested.
ERROR 2	Square root of negative number was requested.
ERROR 3	Division by zero was requested.
ERROR 4	Overflow, exponent greater than approximately +38.
ERROR 5	Underflow, exponent less than approximately -38.
ERROR 6	Line too long or program too big.
ERROR 7	Characters are being typed too fast; use TAPE command for reading paper tapes.
ERROR 8	System overload caused character to be lost.
ERROR 9	Program too complex or too many variables. (GOSUB, FOR, or user defined function calls are too deeply nested.)
ERROR 10	Missing or illegal operand or double operators.
ERROR 11	Missing operator before a left parenthesis.
ERROR 12	Missing or illegal number.
ERROR 13	Too many digits in number.
ERROR 14	No DEF for function call.
ERROR 15	Missing or mismatched parentheses or illegal dummy variable in DEF.
ERROR 16	Wrong number of arguments in DEF call.
ERROR 17	Illegal character in DEF expression.
ERROR 18	Missing or illegal variable.

Message	Explanation
ERROR 19	Single and double subscripted variables with the same name.
ERROR 20	Subscript out of range.
ERROR 21	No left parenthesis in function.
ERROR 22	Illegal user defined function, not FN followed by a letter and a left parenthesis.
ERROR 23	Mismatched parentheses or missing operator after right parenthesis.
ERROR 24	Syntax error in GOTO.
ERROR 25	Syntax error in RESTORE.
ERROR 26	Syntax error in GOSUB.
ERROR 27	Syntax error in ON.
ERROR 28	Unused
ERROR 29	Syntax error in RETURN.
ERROR 30	RETURN without GOSUB.
ERROR 31	Missing left parenthesis in TAB function.
ERROR 32	Syntax error in PRINT.
ERROR 33	Unavailable device requested.
ERROR 34	Missing or illegal line number.
ERROR 35	Attempt to GOTO or GOSUB to a nonexistent line.
ERROR 36	Missing or illegal relation in IF.
ERROR 37	Syntax error in IF.
ERROR 38	Missing equal sign or improper variable left of the equal sign in LET or FOR.
ERROR 39	Subscripted index in FOR.
ERROR 40	Syntax error in FOR.
ERROR 41	FOR without NEXT.
ERROR 42	Syntax error in LET.
ERROR 43	Syntax error in NEXT.
ERROR 44	NEXT without FOR.
ERROR 45	Too much data typed or illegal character in DATA or the data typed in.

Message	Explanation
ERROR 46	Illegal character or function in INPUT or READ.
ERROR 47	Out of data.
ERROR 48	Unrecognized command: RUN mode.

APPENDIX B

BASIC SYNTAXES

Legend

braces { }	- items enclosed within braces are optional.
character	- a letter, a number or a symbol.
constant	- a number within the range of the computer.
expression	- a constant [like 16.4], a variable [like B or D(6)] or an expression [like 8*A↑2 or A=6].
letter	- an alphabetic character from A through Z.
line number	- an integer from 1 through 2046.
local variable	- a simple variable defined only in relation to the function.
n	- an integer.
text	- a series of characters enclosed within quotation marks.
variable	- a simple variable [like B or B7] or an array variable [like F(9)].

All programming statements must be preceded by a line number. A statement with no line number indicates the statement can be executed only in immediate mode.

BYE
Erases programs currently in memory.

{line number} CDR
Transfers control to the card reader from the terminal or to the terminal from the card reader.

CTRL/C
Stops a running program; prints STOP then READY.

line number DATA constant {,constant,...,constant}
Specifies data for READ statement variables.

line number DEF FN letter (local variable {,variable{,...}}) = expression
Defines a function in one line; a RETURN statement is also needed.

DELETE {1st line number {,2nd line number}}
Erases specified lines from the program currently in memory.

line number DIM array variable {,array variable {, ...}}
Reserves space in memory for an array of fixed length.

EDIT line number
Searches the line for the character typed; then performs editing functions depending on the next character typed.

line number END
Terminates program execution and resets the program line counter to the lowest numbered statement in memory.

{line number} FOR simple variable = expression TO expression {STEP expression}
Executes the program lines between FOR and the corresponding NEXT statement a designated number of times; each time the loop is executed, the simple variable is incremented by 1, unless STEP is specified.

{line number} GOSUB line number
Begins executing the subroutine at the specified line number; must have corresponding RETURN statement.

{line number} GOTO line number
Transfers program execution to the specified line number.

{line number} IF expression THEN line number or executable statement
Expression is evaluated; if it is evaluated as 'true', program execution is transferred to the specified line number.

line number INPUT variable {,variable,...,variable}
Allows values to be assigned to the variables, from the terminal during program execution when a '?' appears on the display.

KEY
Returns to key (normal) mode.

{line number} {LET} variable = expression
Assignment statement; assigns variable a value.

line number LINPUT string variable
Allows a string of alphanumeric characters greater than six characters to be assigned to multiple string variables.

LIST {1st line number {,2nd line number}}
Prints the lines, in numerical order, of the program currently in memory.

{line number} NEXT simple variable
Marks the end of the corresponding FOR loop.

{line number} ON expression GOSUB line number {,line number,...,line number}
Begins executing the subroutine at the first line number if the expression is truncated to 1, at the second line number if the expression is truncated to 2, etc.; must have corresponding RETURN statement.

{line number} ON expression GOTO line number {,line number,...,line number}
Transfers program execution to the first line number if the expression is truncated to 1, to the second line number if the expression is truncated to 2, etc.

{line number} PRINT {any combination of text and expressions}
Allows text and values to be output on the terminal; successive expressions must be separated either by commas (for maximum spacing between successive outputs) or by semicolons (for minimum spacing between successive outputs); if no parameters follow PRINT, the printer performs a carriage return/line feed.

{line number} PTP
Transfers control to the high speed paper punch from the terminal.

{line number} PTR
Transfers control to the high speed paper tape reader from the terminal.

line number READ variable {,variable,...,variable}
Reads, from the DATA statement (beginning at the current data pointer position), values for the specified variables.

line number REMARK {any combination of characters}
Inserts non-executable remarks in a program.

line number RESTORE
Resets data pointer (see READ and DATA) to the first constant in the lowest-numbered DATA statement.

line number RETURN
RETURN is the subroutine exit, transferring program execution to the line following the GOSUB statement.

RUN
Begins execution of the program currently in memory.

SCRATCH
Erases program currently in memory.

line number STOP
Terminates program execution.

TAPE
Reads a program or punches a program via the teletype paper tape reader or punch.

APPENDIX C

ASCII CHARACTER CODES

The ASCII[1] character codes shown in the following table are used by the EduSystem as the argument in the CHR$ function. For each ASCII code a second acceptable form is permitted in CHR$. The second code is obtained by adding 128 to the code given in the following table. For example, CHR$ would print A in response to either 65 or 193 as an argument.

$128_{10} = 200_8$

Character	ASCII Code No. (Decimal)		Character	ASCII Code No. (Decimal)	
linefeed	10	012			
formfeed	12	014			
RETURN	13	015			
space	32	040	@	64	100
!	33		A	65	
"	34		B	66	
#	35		C	67	
$	36		D	68	
%	37		E	69	
&	38		F	70	
'	39		G	71	107
(	40	050	H	72	110
)	41		I	73	
*	42		J	74	
+	43		K	75	
,	44		L	76	
-	45		M	77	
.	46		N	78	
/	47		O	79	117
0	48	060	P	80	120
1	49		Q	81	
2	50		R	82	
3	51		S	83	
4	52		T	84	
5	53		U	85	
6	54		V	86	
7	55		W	87	127
8	56	070	X	88	130
9	57		Y	89	
:	58		Z	90	
;	59		[	91	
<	60		\	92	
=	61		]	93	
>	62		↑	94	
?	63	077	←	95	137

[1]An abbreviation for American Standard Code for Information Interchange.

APPENDIX D

CONFIGURATION TAPES AND OS/8 OPERATION

D.1 CONFIGURATION TAPES

Once EduSystem 20 is loaded and running, you may want to modify the answers to some of the questions asked in the initial dialog (see Section 3.3). For example, you may want to add a terminal to the system or change a terminal code, or allocate more memory to a particular user. To do this, you can stop time sharing operations and reload EduSystem 20 from paper tape (a time consuming process), or you can change the information by using a configuration tape.

The paper tape marked DEC-08-ED20C-A-PB2 is the configuration tape to be used with version A only; the paper tape marked DEC-08-ED20C-A-PB4 can be used only with version B. Of course, card reader support can be added only if the original paper tape contained version B of EduSystem 20. These tapes allow you to change the configuration of your system without reloading the entire EduSystem 20 paper tape.

D.1.1 Version A Reconfiguration

To modify the configuration of a version A system, use the paper tape marked DEC-08-ED20C-A-PB2. Each user should erase the program in memory by typing SCRATCH on his keyboard and then typing the RETURN key. Then every user should turn his terminal OFF.

Press the HALT switch and then raise it.

The RIM loader remains in core, but verify that it is there by performing the procedure illustrated in Figure 3-2, Section 3.2.

Now insert the paper tape in the paper tape reader. Leader/Trailer code should be positioned over the reader sensors.

Be sure the Switch Register contains zeros in every position. Press the key marked EXTD ADDR LOAD.

Set the Switch Register to 7756 and press the key marked ADDR LOAD. Now verify that the high speed reader is ON or the Teletype paper tape reader control knob is set to LINE. Press the keys marked CLEAR and CONT.

If you are using the high speed reader, the tape will begin reading in at this point. On the Teletype paper tape reader, set the reader switch to START. The tape will begin reading in.

When finished loading, the system prints the following message for version A:

```
EDUSYSTEM 2Ø BASIC
ED2ØC-A-VAØ4.ØØ
```

The system immediately asks the questions you answered in the initial dialog. Simply re-enter the answers to reflect the new configuration (see Section 3.3).

D.1.2 Version B Reconfiguration

To modify the configuration of a version B system, use the paper tape marked DEC-08-ED20C-A-PB4. Each user should erase the program in memory by typing SCRATCH on his keyboard and then typing the RETURN key. Then every user should turn his terminal OFF.

Press the HALT switch and then raise it.

The RIM loader remains in core, but verify that it is there by performing the procedure illustrated in Figure 3-2, Section 3-2.

Now insert the paper tape in the paper tape reader. Leader/Trailer code should be positioned over the reader sensors.

Be sure the Switch Register contains zeros in every position. Press the key marked EXTD ADDR LOAD.

Set the Switch Register to 7756 and press the key marked ADDR LOAD. Now verify that the high speed reader is ON or the Teletype paper tape reader control knob is set to LINE. Press the keys marked CLEAR and CONT.

If you are using the high speed reader, the tape will begin reading in at this point. On the Teletype paper tape reader, set the reader switch to START. The tape will begin reading in.

When finished loading, the system prints the following message for version B:

```
EDUSYSTEM 2Ø BASIC
ED2ØC-A-VBØ4.ØØ
```

The system immediately asks the questions you answered in the initial dialog. Simple re-enter the answers to reflect the new configuration (see Section 3.3).

D.2 SAVING EDUSYSTEM 20 ON OS/8

If your system includes OS/8, EduSystem 20 can be stored on an OS/8 device. The following two sections describe saving procedures for versions A and B on the system device.

D.2.1 Version A Storage

EduSystem 20 can be saved under OS/8 by performing the procedure shown below. Remember: Version A does not support the card reader.

Once OS/8 is loaded in the monitor, place the EduSystem 20 binary paper tape (DEC-08-ED20C-A-PB1) in the paper tape reader. Be sure the tape is positioned so that the three Leader/Trailer codes after the RIM-formatted binary loader and before the binary-formatted EDU20 are placed over the sensors.

Type in the characters shown below from your terminal.

```
.R PIP↵
*ED20A.BN<PTR:/B↵
↑*^C    Type space after up arrow and CTRL/C after asterisk

.R ABSLDR↵
*ED20A.BN$ ←altmode key
.SAVE SYS EDU20 0-7577,10000-17577,12401↵
```

EDU20 can now be run by typing:

```
.R EDU20↵
```

The system prints the following message.

```
TO BOOTSTRAP BACK OS/8 MONITOR:
  LOAD ADDRESS 07600
  AND START

EDUSYSTEM 20  BASIC
ED20C-A-VA04.00
```

The system immediately begins the system dialog shown in Section 3.3. Refer to that section for information on answering these questions.

D.2.2 Version B Storage

Version B (card reader support) of EduSystem 20 can be saved under OS/8 by performing the procedure shown below.

Once OS/8 is loaded in the monitor, place the EduSystem 20 binary paper tape (DEC-08-ED20C-PB3) in the paper tape reader. Be sure the tape is positioned so that the three Leader/Trailer codes after the RIM-formatted binary loader and before the binary-formatted EDU20 are placed over the sensors.

Type in the characters shown below from your terminal.

```
.R PIP↵
*ED20B.BN<PTR:/B↵
↑*^C    type space after up arrow and CTRL/C after asterisk

.R ABSLDR
*ED20B.BN$ ←altmode key
.SAVE SYS EDU20 0-7577,10000-17577,20000-27777:22401↵
```

EDU20 can now be run by typing:

```
.R EDU20
```

The system prints the following message.

```
TO BOOTSTRAP BACK OS/8 MONITOR:
  LOAD ADDRESS 07600
  AND START

EDUSYSTEM 20  BASIC
ED20C-A-V804.00
```

The system immediately begins the system dialog shown in Section 3.3. Refer to that section for information on answering these questions.

D.3 ASSEMBLING EDUSYSTEM 20 UNDER OS/8

If you have the EduSystem 20 source and wish to assemble it using OS/8, perform the following procedure.

When OS/8 is loaded in the monitor, type the following characters from your terminal.

```
.R PAL8
*EDU20.BN,LPT:<EDU20.PA/K
```

Assembling EDU20 requires at least 16K of core. The /K option shown above is required since there are too many symbols in EDU20 to assemble in 8K of core. The option /C is not used because EDU20 is too large to CREF on OS/8.

The list file is very large and does not fit on a device which contains many other files. For this reason, the list file should be output directly to the line printer.

INDEX

HOW TO OBTAIN SOFTWARE INFORMATION

SOFTWARE NEWSLETTERS, MAILING LIST

The Software Communications Group, located at corporate headquarters in Maynard, publishes software newsletters for the various DIGITAL products. Newsletters are published monthly, and keep the user informed about customer software problems and solutions, new software products, documentation corrections, as well as programming notes and techniques.

There are two similar levels of service:

- The Software Dispatch
- The Digital Software News

The Software Dispatch is part of the Software Maintenance Service. This service applies to the following software products:

PDP-9/15
RSX-11D
DOS/BATCH
RSTS-E
DECsystem-10

A Digital Software News for the PDP-11 and a Digital Software News for the PDP-8/12 are available to any customer who has purchased PDP-11 or PDP-8/12 software.

A collection of existing problems and solutions for a given software system is published periodically. A customer receives this publication with his initial software kit with the delivery of his system. This collection would be either a Software Dispatch Review or Software Performance Summary depending on the system ordered.

A mailing list of users who receive software newsletters is also maintained by Software Communications. Users must sign-up for the newsletter they desire. This can be done by either completing the form supplied with the Review or Summary or by writing to:

Software Communications
P.O. Box F
Maynard, Massachusetts 01754

SOFTWARE PROBLEMS

Questions or problems relating to DIGITAL's software should be reported as follows:

North and South American Submitters:

Upon completion of Software Performance Report (SPR) form remove last copy and send remainder to:

Software Communications
P.O. Box F
Maynard, Massachusetts 01754

The acknowledgement copy will be returned along with a blank SPR form upon receipt. The acknowledgement will contain a DIGITAL assigned SPR number. The SPR number or the preprinted number should be referenced in any future correspondence. Additional SPR forms may be obtained from the above address.

All International Submitters:

Upon completion of the SPR form, reserve the last copy and send the remainder to the SPR Center in the nearest DIGITAL office. SPR forms are also available from our SPR Centers.

PROGRAMS AND MANUALS

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center.

Digital Equipment Corporation
Software Distribution Center
146 Main Street
Maynard, Massachusetts 01754

Digital Equipment Corporation
Software Distribution Center
1400 Terra Bella
Mountain View, California 94043

Outside of the United States, orders should be directed to the nearest Digital Field Sales Office or representative.

USERS SOCIETY

DECUS, Digital Equipment Computers Users Society, maintains a user exchange center for user-written programs and technical application information. The Library contains approximately 1,900 programs for all DIGITAL computer lines. Executive routines, editors, debuggers, special functions, games, maintenance and various other classes of programs are available.

DECUS Program Library Catalogs are routinely updated and contain lists and abstracts of all programs according to computer line:

- PDP-8, FOCAL-8, BASIC-8, PDP-12
- PDP-7/9, 9, 15
- PDP-11, RSTS-11
- PDP-6/10, 10

Forms and information on acquiring and submitting programs to the DECUS Library may be obtained from the DECUS office.

In addition to the catalogs, DECUS also publishes the following:

DECUSCOPE	-The Society's technical newsletter, published bi-monthly aimed at facilitating the interchange of technical information among users of DIGITAL computers and at disseminating news items concerning the Society. Circulation reached 19,000 in May, 1974.
PROCEEDINGS OF THE DIGITAL EQUIPMENT USERS SOCIETY	-Contains technical papers presented at DECUS Symposia held twice a year in the United States, once a year in Europe, Australia, and Canada.
MINUTES OF THE DECsystem-10 SESSIONS	-A report of the DECsystem-10 sessions held at the two United States DECUS Symposia.
COPY-N-Mail	-A monthly mailed communique among DECsystem-10 users.
LUG/SIG	-Mailing of Local User Group (LUG) and Special Interest Group (SIG) communique, aimed at providing closer communication among users of a specific product or application.

Further information on the DECUS Library, publications, and other DECUS activities is available from the DECUS offices listed below:

DECUS
Digital Equipment Corporation
146 Main Street
Maynard, Massachusetts 01754

DECUS EUROPE
Digital Equipment Corp. International
(Europe)
P.O. Box 340
1211 Geneva 26
Switzerland

Please cut along this line.

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form (see the HOW TO OBTAIN SOFTWARE INFORMATION page).

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you do not require a written reply, please check here. ☐

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

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

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

Please cut along this line.

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form (see the HOW TO OBTAIN SOFTWARE INFORMATION page).

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you do not require a written reply, please check here. ☐

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

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

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

digital

digital equipment corporation