NASA

Technical Memorandum 80690

(NASA-TM-80690) THE RSZ BASIC PROGRAMMING LANGUAGE MANUAL (NASA) 52 p HC A04/MF A01 CSCL 09B

N80-26053

Unclas
G3/61 22950

The RSZ Basic Programming Language Manual

Raymond J. Stattel, James K. Niswander and Anil K. Kochhar

JUNE 1980

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

TM 80690

THE RSZ BASIC PROGRAMMING LANGUAGE MANUAL

Raymond J. Stattel
James K. Niswander
Anil K. Kochhar

June 1980

GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland

PREFACE

The Sounding Rocket Division Instrumentation Branch has provided PCM telemetry ground support equipment for the sounding rocket project since the introduction of its airborne PCM telemetry system in 1972. The various devices now displaying the PCM telemetry data are:

- CRT analog Bar Graph and Storage Scope displays
- Switch selectable electrostatic plots
- Remote multiple numeric LED displays
- Special purpose mini-computer driven CRT displays

A SRD Ling 1800 mini-computer is used for generating PCM formatted tapes and performing various telemetry data processing tasks.

The balloon project at Goddard Space Flight Center identified a requirement for a small portable computer-based system which would allow observation of many pieces of PCM telemetry data in real time. The Instrumentation Branch had proposed the development of such a system called the Telemetry Data Processor, a Zilog Z80 microprocessor-based microcomputer system which would offer valuable features not otherwise commercially available. After an evaluation of existing commercial systems, the balloon project decided to support the development of the Telemetry Data Processor system.

A prototype Telemetry Data Processor system was designed, built and tested by the Instrumentation Branch. The prototype system has already successfully supported four balloon flight launched at Palestine, Texas. The procurement of Telemetry Data Processor production units is in progress. It is expected that the Telemetry Data Processor system will be used in sounding rocket, balloon, aircraft, and Get-Away-Special and Experiment of Opportunity shuttle projects.

The Telemetry Data Processor System (Figure I) is dedicated to decommutation, processing and display of PCM telemetry data. The system runs in a resident BASIC interpreter which has special enhancement to increase programming efficiency for telemetry data handling and facilitate overall operations. The system is composed of:

- CRT Display
- Keyboard
- Dual Mini-floppy Disk Drives
- Frame Synchronizer
- Telemetry Data Memory
- Versatec Electrostatic Plotter and Printer Controller
- Thirty-two Analog Output Controller
- Dual 16-Bit Parallel Data Output Controller
- Bit Synchronized Remote Controller Interface
- Time Code Reader Interface
- General Purpose Computer Interface

The principal output devices are the CRT display, thirty-two analog outputs, the electrostatic printer/plotter and the two 16-bit parallel data output ports.

The CRT display is used for presenting numeric and graphic information derived from tne telemetry data stream and processed by the RSZ BASIC programs. Typical applications have up to 44 pieces of telemetry data in a numeric display. A magnitude or deviation bar graph display may show 10 pieces of data with a bar resolution of one part in 400.

The analog outputs, 16-bit parallel data outputs, and Versatec electrostatic plotter display data separately and independently from the CRT display. They are each driven by hardware controllers which are loaded once with telemetry data selection and option parameters and immediately begin decommutating, processing and delivering outputs. Each controller can operate at data rates up to 1 x 10^6 bits per second.

The general purpose computer interface is provided so that future special purpose devices can be tied into the system. A Telemetry Data Processor system Block Diagram is shown in Figure II.

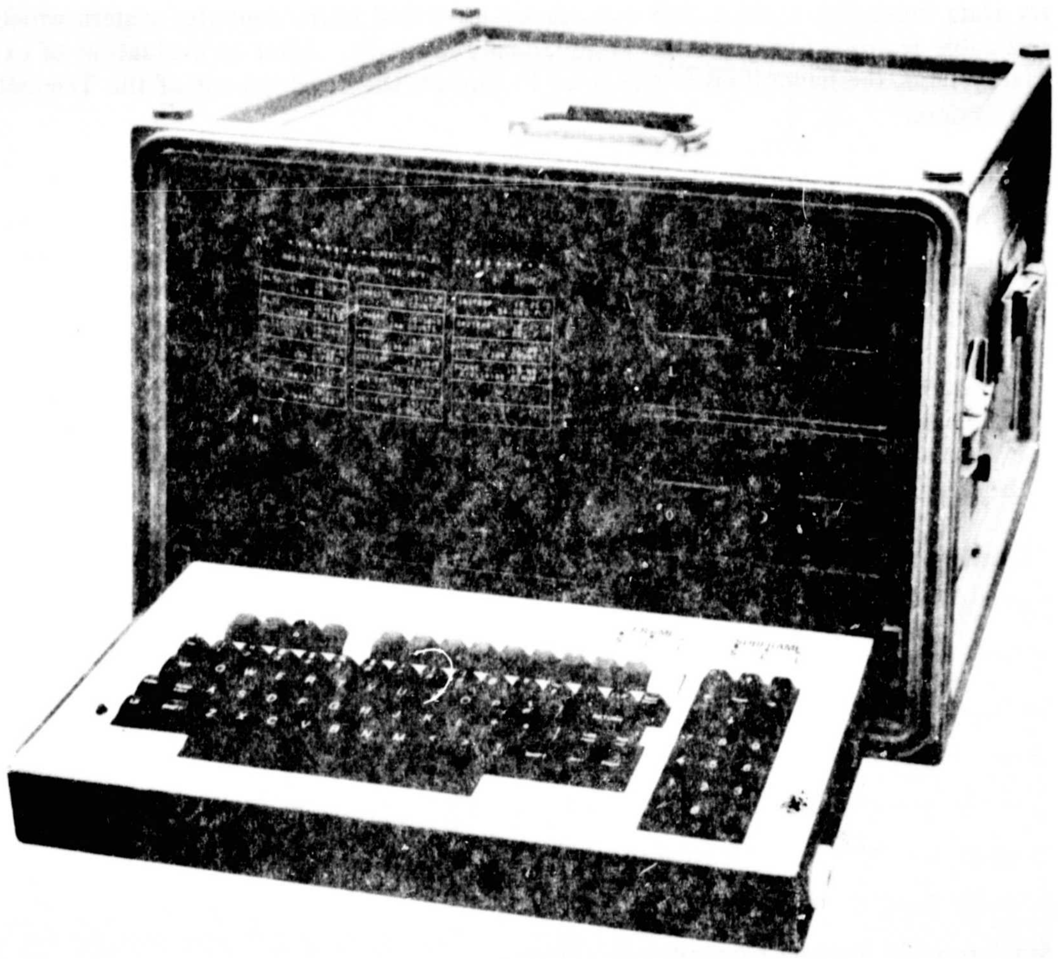


Figure I. Telemetry Data Processor Prototype Front View

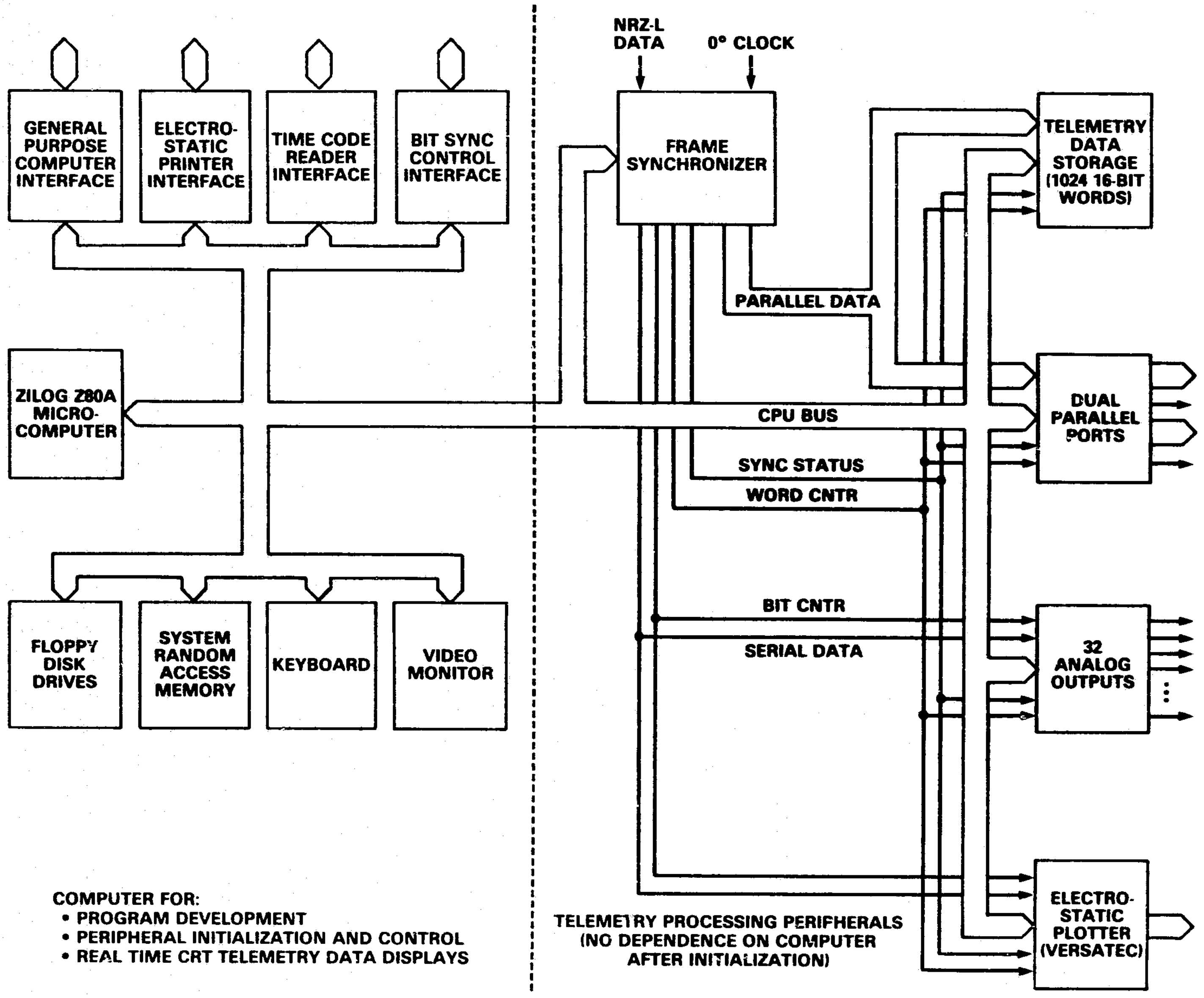

GENERAL PURPOSE COMPUTER INTERFACE
ELECTRO-STATIC PRINTER INTERFACE
TIME CODE READER INTERFACE
BIT SYNC CONTROL INTERFACE
ZILOG Z80A MICRO-COMPUTER
FLOPPY DISK DRIVES
SYSTEM RANDOM ACCESS MEMORY
KEYBOARD
VIDEO MONITOR
COMPUTER FOR:
• PROGRAM DEVELOPMENT
• PERIPHERAL INITIALIZATION AND CONTROL
• REAL TIME CRT TELEMETRY DATA DISPLAYS
NRZ-L DATA
0° CLOCK
FRAME SYNCHRONIZER
PARALLEL DATA
CPU BUS
SYNC STATUS
WORD CNTR
BIT CNTR
SERIAL DATA
TELEMETRY DATA STORAGE (1024 16-BIT WORDS)
DUAL PARALLEL PORTS
32 ANALOG OUTPUTS
ELECTRO-STATIC PLOTTER (VERSATEC)
TELEMETRY PROCESSING PERIPHERALS (NO DEPENDENCE ON COMPUTER AFTER INITIALIZATION)

Figure II. Telemetry Data Processor Block Diagram

ACKNOWLEDGMENT

This document was developed by Systex Inc., Beltsville, MD in collaboration with the authors. Anil Kochhar is a member of senior technical staff of Systex Inc.

ABSTRACT

This document describes the RSZ BASIC, an interactive language developed by the Sounding Rocket Division Instrumentation Branch. The BASIC interpreter is a resident in the Telemetry Data Processor System. The system is dedicated to processing and displaying PCM telemetry Data. A series of working examples teaches the fundamentals of RSZ BASIC and shows how to construct, edit and manage storage of programs.

CONTENTS

CONTENTS (Continued)

LIST OF TABLES

CONTENTS (Continued)

LIST OF ILLUSTRATIONS

Chapter 1
INTRODUCTION

1.1 PURPOSE

This document presents the information needed by a user to implement programs written in RSZ BASIC for the Telemetry Data Processor System. Each RSZ BASIC statement is described in detail. Applying a programmed series of working examples, this document shows how to construct, edit and manage the BASIC programs. As the RSZ BASIC statements language is enhanced, revision to this document will be generated. The user should note that, if necessary, additional BASIC statements can be implemented on the Telemetry Data Processor System tailored to user's need. The RSZ BASIC Programming Language Manual is the first of four planned documents to be released by the Instrumentation Branch on the Telemetry Data Processor System. The other three documents planned to be released are:

- Telemetry Data Processor Hardware Manual
- Telemetry Data Processor RSZ BASIC Application Manual
- Z80 Assembly Language for RSZ BASIC

1.2 GETTING STARTED

To start the Zilog Z80 microprocessor based microcomputer system, simply turn on the power; the switch is located at the rear of the Telemetry Data Processor system (Figure 1-1). The message "RSZ BASIC 1.3" will appear in the upper left corner of the display screen followed by a system prompt character ">" in the next line. The user interacts with the system by entering data through the keyboard.

```
>?1+2
  3
>?1/2
  0.5
>?1*2
  2
```

As the illustration shows, the terminal can be used as a desk calculator. We can assign a value to a variable and ask the system to execute the RSZ BPL Statement.

```
>X = COS (45)
>PRINT X
  0.25
```

The system responds with the value of X.

The system is comprised of three levels; Command, Edit and Execution.

The Command mode is invoked by turning on the power switch. It is also entered upon completion of Edit or the Execution mode. In the Command mode the system responds to System and

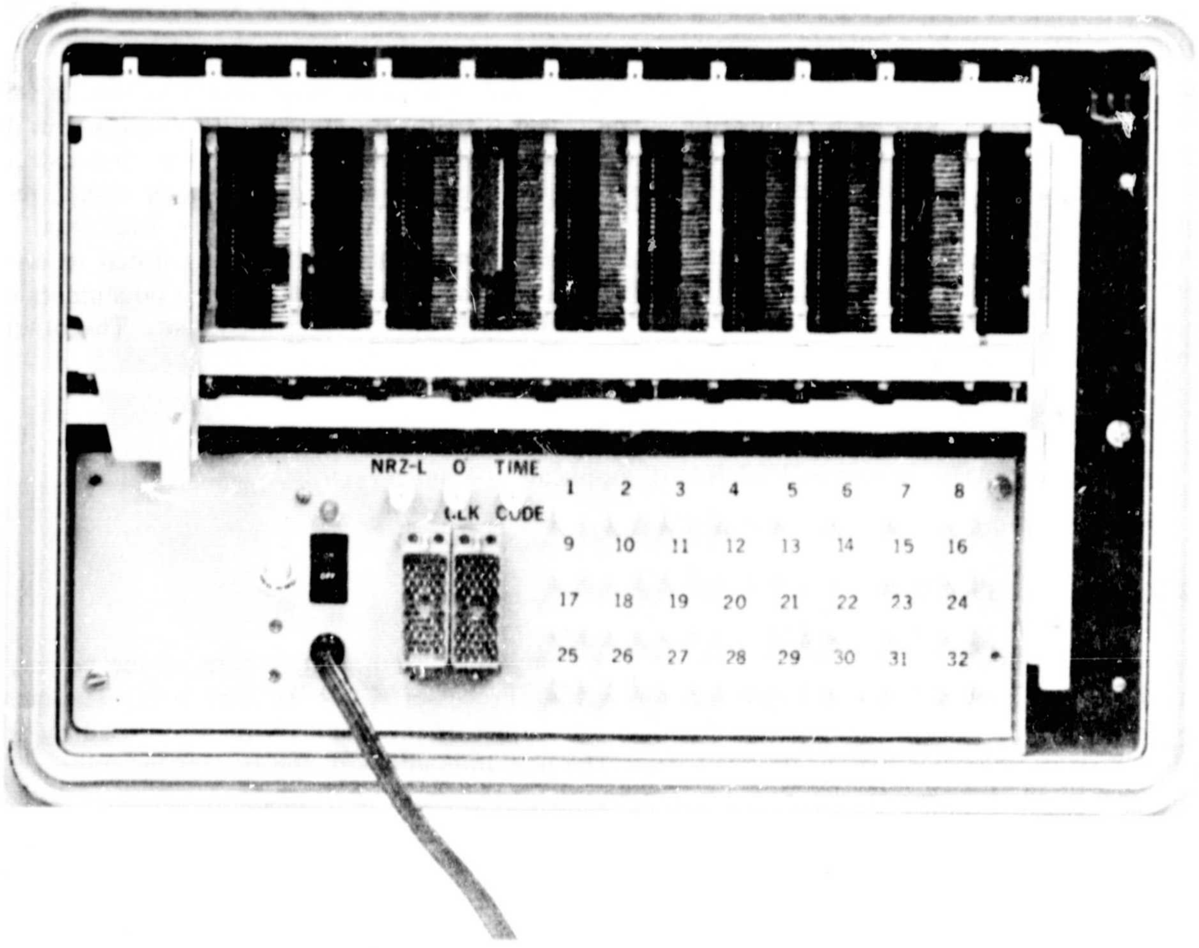


Figure 1-1. Telemetry Data Processor Prototype Rear View

Terminal commands and the RSZ BASIC statements. A list of all Systems and Terminal commands and their description is provided in appendices B and C respectively.

The Edit mode is invoked by entering the DEF statement. The RSZ BASIC editor program is described in Chapter 8. The Edit mode is terminated by pressing the ESC key.

The Execution mode is invoked by entering the CALL statement. The CALL statement is described in Chapter 7. It allows us the ability to call upon functions previously stored in the system workspace. The Execution mode is terminated by either a RETURN from the main function or by pressing the ESC key.

1.3 LINE NUMBERS

In the standard BASIC language program, the assignment of a line number to each statement is done by the program author. However, in the RSZ BASIC language, this assignment is a computer function.

1.4 THE REMARK STATEMENT (REM)

The REM Statement enables the program author to insert explanatory comments into the program. Such remarks can and should be used freely throughout a program. Not only do they inform others about the intent of the program but they may also help the programmer if he attempts to modify the program some time after writing it. Each remark statement consists of REM followed by the remark itself.

```
REM THIS IS A REMARK STATEMENT
```

The computer will ignore any information that follows REM. The single quote character (') may be used in place of the key word REM when typing in the remark statement. Note that the remarks describing the action of the various steps can be included along with the program statements. RSZ BASIC language allows for more than one statement on a line with the use of a colon (:) between statements.

```
LET C1 = 0 : REM INITIALIZE COUNTER
```

1.5 LET STATEMENT

The principal computational statement is the LET statement, which takes the general form

```
LET <Variable> = <Expression>
```

The = symbol is not a mathematical equality sign. It means "replaced by". Therefore, this statement is interpreted to mean, "the value of the arithmetic expression on the right of the (=), or "replaced by" sign, replaces the value of the variable on the left.

```
LET N = N + 1
```

This statement results in the value 1 being added to the value N in storage. The new sum replaces the original value of N. Note that the key word LET is optional.

Chapter 2
VARIABLES, TYPES, OPERATORS AND EXPRESSIONS

2.1 CONSTANTS

A constant is a number like 2, 5, 19, 10.3E - 3, and does not change in value during program execution. There are two types of constants: Integer constants and Real (floating point) constants. An integer constant is a string of digits which does not contain a decimal point, for example, 9999. The range of an integer constant is ±32767. A real constant is a decimal number with precision to seven digits, as represented in one of the following forms:

±Int, ±.Fra, ±Int.FraE±exp

where Int, Fra, exp are each a string of digits representing integers, fractions and exponents respectively. The exponent value range is ±127.

2.1.1 Hexidecimal Constant

A hexidecimal constant is represented in the form:

#n

where n is a string of hexidecimal characters. The hexidecimal characters include digits 0 through 9, and A through F. A hexidecimal constant may be used in an assignment statement or expression. The constant is right justified and zero filled,

for example

```
A! = #002F
```

can be written as

```
A! = #2F
```

2.1.2 Binary Constant

A binary constant is represented in the form:

##n

where n is a string of zero's and one's. A binary constant may be used in an expression or assignment statement. The constant is right justified and zero filled. For example,

```
A# = ##00011010
```

can be written as

```
A# = ##11010
```

2.2 VARIABLES

A variable represents a storage unit that may take on a different value during the execution of a program. In the RSZ BASIC, the variable name is restricted to a single letter or a letter and a digit, for example:

A, B0, G, S2, X9

Note that in the RSZ BASIC, a single letter variable name, and a single letter plus digit 0 variable name with the same single letter, share the same storage unit. For example, B and B0 are interchangable within a program. Variables are of four data types as shown in Table 2-1.

Table 2-1
Variable Type

Type	Format and Description
Integer Byte	A variable name followed by a #, for example, A1#. It is a 8-bit unsigned integer, negative numbers are undefined.
Integer	A variable name followed by a !, for example, A1!. It is a 16-bit signed integer. Negative numbers are represented in two's complement.
String	A variable name followed by a $, for example, A1$. A string variable takes on a value of a string enclosed in quotation marks, for example, A1$ = "THIS IS A STRING". There is no limit on the length of a string.
Real	A variable name not followed by #, ! or $ is a real variable, for example, A1, B9, Z.

2.3 ARITHMETIC OPERATORS

The RSZ BASIC has four arithmetic operators. They are as follows:

+	Addition
-	Subtraction
*	Multiplication
/	Division

Note that the current version of RSZ BASIC does not have an exponentiation operator.

2.4 RELATIONAL AND LOGICAL OPERATORS

The relational operators are:

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

The logical operators are:

AND	bitwise AND
OR	bitwise OR
NOT	one's complement of a 16-bit integer
BNOT	one's complement of a 8-bit integer
XOR	bitwise Exclusive OR

2.5 EXPRESSIONS

An expression consists of constants and variables connected by operators. There are three types of expressions: Arithmetic expressions, Logical expressions and Relational expressions. An arithmetic expression consists of constants and variables separated by arithmetic operators, for example:

```
A + 4 * B ! - 16.3E-3*C2 + D * (C2!/C3!)
```

A logical expression consists of constants and variables separated by logical operators, for example:

```
NOT ( (A! AND (B! OR C!) ) XOR D! )
```

A relational expression consists of constants and variables separated by relational operators, for example:

```
X1! > 20
X1! > X2!
```

Chapter 3
ARRAYS AND SUBSCRIPTED VARIABLES

3.1 THE DIM STATEMENT

An array is a group of values of the same type. There are three types of arrays in RSZ BASIC. A one-dimensional array, a two-dimensional array and a three-dimensional array. In mathematics, an element of an array is denoted by a lowercase subscript attached to the symbol for the array. In the RSZ BASIC the subscripts are written in parentheses following the variable name for the array. For example, X(N) in the RSZ BASIC is equivalent to the mathematical notation X_n denoting element n of the array X. Subscripted variables may be used in the same way that a non-subscripted variable is used in the RSZ BASIC. Since a subscripted variable refers to many storage locations, we need a statement which will let us define the size of this array. The DIM statement does this. The general form of DIM statement is

$$\text{DIM Name}_1\ (d_1, --, d_3), -------, \text{Name}_n\ (d_1, --, d_3)$$

where the name is a subscripted variable name (integer, real, or string) and the d's are positive integer constants which indicates the number of subscripts to be used in the program. For example,

```
DIM A1$ (2), B9! (3), A2# (2, 3, 2)
```

In the RSZ BASIC, the array subscript always starts at zero, so the elements of A1$ are A1$ (0), A1$ (1). Note, in the RSZ BASIC the number of subscripts are limited to a maximum of three and the subscripted value to 255. All subscripted variables must be defined in a DIM statement prior to their first use in a program. An array dimension with a zero value in a DIM statement will generate a syntax error. For example,

```
DIM A1(0)
```

is not allowed.

3.2 STORAGE ALLOCATION FOR ARRAYS

In RSZ BASIC arrays are stored by rows, that is with the second of their subscripts varying most rapidly; for example, a two-dimensional array X1(2, 2), would be stored in ascending locations as follows:

X1 (0, 0) X1 (0, 1) X1 (1, 0) X1 (1, 1)

$\xrightarrow{\hspace{4cm}}$

Storage Addresses

A three-dimensional array X2 (2, 2, 2), would be stored in ascending locations as follows:

X2 (0, 0, 0) X2 (0, 1, 0) X2 (1, 0, 0) X2 (1, 1, 0)

X2 (0, 0, 1) X2 (0, 1, 1) X2 (1, 0, 1) X2 (1, 1, 1)

$\xrightarrow{\hspace{8em}}$

Storage Addresses

3.3 INITIALIZATION OF ARRAYS

In RSZ BASIC, an element of an array or the entire array may be initialized by using the assignment statement (LET Statement). Let us initialize a previously specified 3 element array B9! (3) as follows:

```
B9! (0) = 10
B9! (1) = 20
B9! (2) = 30
```

This can be written instead as

```
B9! ( ) = 10, 20, 30
```

Two comments should be made regarding the initialization of an array. First, if there are not enough values in the assignment statement to initialize the entire array, the system goes back to the start of the first assigned value on the right of "=" in the assignment statement and starts over. For example, an array A (2, 3) initialized as follows

```
A ( ) = 1, 2
```

will result in

```
A (0, 0) = 1 A (0, 1) = 2 A (0, 2) = 1
A (1, 0) = 2 A (1, 1) = 1 A (1, 2) = 2
```

If the array A (2, 3) were initialized as

```
A ( ) = 0
```

then each array element is filled with zero.

Second, if there are too many numbers of values assigned to an array, only those needed are used and the rest are ignored. For example, an array B (2, 2) initialized as follows:

```
B ( ) = 0, 1, 2, 3, 4
```

will result in

```
B (0, 0) = 0  B (0, 1) = 1
B (1, 0) = 2  B (1, 1) = 3
```

Chapter 4
CONTROL STATEMENTS

The control statements of a language determine the flow of a program. In this chapter we will discuss the several statements that allow the programmer to specify the sequence of program execution.

4.1 GO TO STATEMENT

This statement causes transfer of control to some designated statement (line number) at any point in the program. The statement has the general form of

GO TO <LINE NUMBER>

Example of the GO TO statement follows:

```
1 INPUT X
2 LET Y = X * 2 + 3
3 PRINT Y
4 GO TO 1
```

After printing the value of Y, the control is transferred to statement 1.

4.2 IF-THEN-ELSE STATEMENT

The IF-THEN-ELSE is used to make decisions. The statement has the general form

IF <Expression> THEN <Statement_1> ELSE <Statement_2>

where the ELSE part is optional. The expression is then evaluated, and if it is "true", statement_1 is executed. If the expression is "false" and if there is an ELSE part, Statement_2 is executed, otherwise, the next statement in the program is executed.

The statement can also be written without the key word THEN as follows:

IF <Expression> <Statement_1> ELSE <Statement_2>

Examples of this statement and their meanings are as follows:

IF X > = 0 THEN GO TO 10	If the value of X is positive, transfer program control to line number 10. If not, continue with next in-line statement.
IF X < 0 RETURN ELSE X = X - 1	If the value of X is negative, transfer control to the calling function. If not, decrement value of X and continue with the following statement.

4.3 FOR AND NEXT STATEMENTS

The FOR and NEXT Statements are used in constructing a program loop. The general format of the FOR statement is

```
FOR <variable> = <a> to <b> STEP <c>
```

where the variable is the index, a is the initial value of the index, b is the terminal value of the index, and c is the positive or negative value by which the index is to be modified for each iteration. When c is 1, the general form may be simplified to

```
FOR <variable> = <a> to <b>
```

The general form of the NEXT statement is

```
NEXT <variable> or
NEXT <list of variables separated by commas>
```

where the variable must be the same as that in the corresponding FOR statement.

Note that every FOR statement must have an associated NEXT statement; that is, each FOR and NEXT Statement must form a pair.

A general format for the loop using FOR and NEXT follows:

```
FOR I = 1 TO 100
   Statement_1
      .
      .
      .
      .
   Statement_N
NEXT I
```

Examples of FOR and NEXT Statement:

```
1  FOR N = 1 TO 5  STEP 0.5
2  PRINT N;
3  NEXT N
```

prints the following

```
1  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0
```

```
1  FOR N = 1 TO 5
2  PRINT N;
3  NEXT N
```

prints the following values of N

```
1 2 3 4 5
```

```
1  FOR I = X to Y
2  FOR J = A to B
3  T# (I, J) = R# (I, J)/25
4  NEXT J, I
```

executes both loops with one NEXT statement.

Chapter 5
INPUT AND OUTPUT

The RSZ BASIC provides the programmer with INPUT and PRINT Statements for sending information to and from the processor.

5.1 INPUT STATEMENT

The general form of this statement is:

INPUT <@(Expr_1, Expr_2)> <"Any String of Characters"> <Var_1, . . . , Var_n>

The first argument positions the cursor to a row and column specified by Expression_1 and Expression_2. The value of Expression_1 and Expression_2 must not exceed 31 and 63 respectively. The second argument provides a message identifying the information the user is to input, and is comprised of any string of characters in quotes. The third argument is a list of variables of any type. The first and/or second argument in the statement is optional. For example, the statement

INPUT "NEXT VALUES OF TEMPERATURE T1, T2" T1, T2

would prompt the user as follows:

NEXT VALUES OF TEMPERATURE T1, T2?

Now the user may enter the temperature values requested, followed by RETURN. Each value input must be separated by a comma. A colon is used to separate entries into an array. Note that the question mark was provided by the execution of the INPUT statement.

5.2 PRINT STATEMENT

The general form of this statement is:

PRINT <@(Expr_1, Expr_2)> <% Format specification %> <Var_1, . . . , Var_n>

The @(Expr_1, Expr_2) positions the cursor to a row and column specified by Expr_1 and Expr_2. The format specification between the characters % defines the Output format of the variables. The format specifications and their descriptions are as follows:

<u>I – Format Specification</u>

Form: Iw.d

It may be used for printing an integer or real data. w is an integer constant defining the field width in number of digits excluding any decimal point. d is an integer constant specifying the number of digits appearing to the right of the decimal point. For example,

Format Specification	Internal Value	Output (b = blank)
I2	15	b15b
I3.1	15.1	b15.1b
I4.2	15.1	b15.10b
I10.2	-678.4	-bbbbb678.40b
I2	100	b***b

E - Format Specification

Form: Ew

Real data is printed using this conversion. w is again an integer constant defining the field width. For example,

Format Specification	Internal Value	Output (b = blank)
E3	30.5	b3.05E1b
E2	30.5	b3.1E1b
E4	-35.53	-3.553E1b

X - Format Specification

Form: Xw

It causes all types of data to be printed in hexidecimal. w is an integer constant specifying number of hexidecimal digits to be printed. For example,

Format Specification	Decimal Value	Type	Output (b = blank)
X4	22358	Integer	b5756b
X4	-35.53	Real	bFEDCb
X2	22358	Integer	b56b

If the specified field width is smaller than the number of hexidecimal digits that the variable holds, the leftmost hexidecimal digits are truncated. Line 3 is an example of this data loss. Note that if the field width is not defined, the default value is 4.

B - Format Specification

Form: Bw

It causes all types of data to be printed in binary. w defines the field width. For example,

Format Specification	Decimal Value	Type	Output (b = blank)
B4	15	Integer	b1111b
B16	22358	Integer	b0101011101010110b
B8	22358	Integer	b01010110b
B	-35.53	REAL	b1111111111011100b

If the specified field width is smaller than the number of binary digits that the variable holds, the leftmost binary digits are truncated. Line 3 is an example of this data loss. Note that if field width is not defined, the default value is 16.

H – Format Specification

Form: H

It causes the output of the print statement to be printed on the line printer. Note that the line printer has 128 columns per line (twice as many as the CRT).

F – Format Specification

Form: F

There are twenty-six special characters stored in the system in addition to the standard ASCII character set. The F format specification, followed by an alpha(s) and/or @ character(s) in quotation marks, will output a corresponding special character(s). Thus, alpha characters and the @ character in quotes access an additional set of 26 special characters. For example,

PRINT % F % "A"

will output

┐

PRINT % F % "@C"

will output

┌┘

The @ and alpha characters and their corresponding special characters are listed below.

ASCII Character	Special Character
@	┌
A	┐
B	─
C	┘
D	└

ASCII Character	Special Character
E	│
F	├
G	┬
H	┴
I	┤
J	┼
K	▏
L	▎
M	▍
N	▌
O	▋
P	▊
Q	▉
R	█
S	▏
T	▎
U	▍
V	▌
W	▋
X	▊
Y	▉
Z	Not Defined

Note that the Z character does not have corresponding special character.

Aside from the H declaration, a previously defined format specification remains in effect until redefined by another format specification in a PRINT statement. The % % characters without the format specification in a PRINT statement results in data being output as follows:

- An integer data value is output as an integer.
- A real data value is output as exponential.
- String data is printed as alphanumeric.

5.3 SOME ADDITIONAL INFORMATION ON PRINTING AND INPUTING DATA

At sign–on the RSZ BASIC system sets the printer tab at columns 16, 32 and 48. A comma in the PRINT statement signals to the system that the preset tab is to be used for printing values. For example, the statement

```
PRINT "NUMBER", 15, 25, -35
```

will output in columns 0, 16, 32, and 48 respectively (b = blank)

```
NUMBER          b15b            b25b            -35b
```

If the statement contains more than four values to be printed, then the next four values will be printed on the next line. Suppose we want to print values next to each other. The use of a semicolon in the PRINT statement signals the system to print the actual number of digits in the values, each separated by a blank, for example, the statement

```
PRINT "NUMBER"; 15; 25; -35
```

will output (b = blank)

```
NUMBERb15bb25b-35b
```

Let us illustrate a simple program in several variations to show the results of writing a PRINT statement in different ways.

```
1 FOR I = 0 TO 30
2 PRINT I
3 NEXT I
```

This program will print numbers 0 through 30, each number starting at column 0 on a separate line. If we modify line 2 to

```
2 PRINT I,
```

The program will print

```
0         1         2         3
4         5         6         7
8         9         10        11
12        13        14        15
16        17        18        19
20        21        22        23
24        25        26        27
28        29        30
```

If we modify line 2, this time to

```
2 PRINT I;
```

The output is (b = blank)

```
b0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
b18 19 20 21 22 23 24 25 26 27 28 29 30
```

All characters in the field bounded by quotation marks are recognized as string data by the RSZ BASIC interpreter. The upper case @ (SHIFT-@) and / (SHIFT-/) display a quotation mark on CRT. How is this quotation mark different from the standard quotation mark? It is not recognized by the RSZ BASIC interpreter because of different value, thus can be part of a string of characters bounded by standard quotation mark. The exception is the SHIFT-@ quotation mark which is recognized by the RSZ BASIC interpreter when it is read from CRT memory block. For example, the statements

```
PRINT @ (1, 16) " "X" "
INPUT @ (1, 1) "CONTINUE "Y/N" " A$
```

(note that the quotation marks embedded into the PRINT AND INPUT statements were generated by entering SHIFT-@ and SHIFT-/ respectively) would output as follows:

```
CONTINUE "Y/N"?"X"
```

Now the user positions the cursor at X and enters Y or N followed by RETURN. The RSZ BASIC interpreter recognizes the quotation mark after the question mark and accepts the entered character Y or N as string data. Note that this quotation mark was generated by entering SHIFT-@ in the PRINT statement above.

5.4 THE TAB STATEMENT

The general form of the tab statement is

TAB C_1, ... , Cn

where C is an integer constant specifying the position in the line where the value is to be printed. For example, the statement

```
TAB 10, 20, 30, 40
```

would set 4 tabs at a specified column. Thus a comma in the PRINT statement will print the values starting at columns 10, 20, 30, and 40.

Chapter 6
SOME ADDITIONAL RSZ BASIC STATEMENTS

6.1 CLEAR STATEMENT

The CLEAR statement is used for initializing the symbol table in the user's workspace. The general form of CLEAR statement is

```
CLEAR
```

Whenever the CLEAR statement is encountered, the computer removes all variables listed in the symbol table.

6.2 DROP STATEMENT

The DROP statement is used for freeing memory space in the user's workspace by removing the subscripted and/or unsubscripted variables from the symbol table. The general format of DROP statement is

DROP <Variable_1, - - - - - - -, Variable_N>

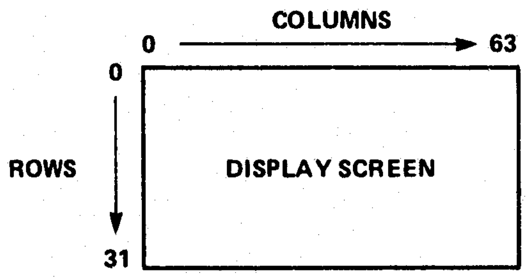
Example of DROP Statement

```
LET S = 0
FOR N = 1 TO 100
S = S + N
NEXT N
DROP S, N
```

when the DROP statement is encountered in the program, the computer removes variables S and N from the symbol table.

6.3 FIELD AND CFIELD STATEMENTS

The system CRT display device is 32 rows by 64 characters with the origin at the top left corner of the display screen.



The display screen can be set to two levels of brightness, white on black (Normal) and black on white (Reverse), with or without the cursor. Each level of brightness, with or without the cursor, can be set by the FIELD statement. The general form of FIELD statement is

$$\text{FIELD} \left\langle \begin{matrix}\text{Field}\\ \text{Identifier}\end{matrix} \right\rangle , \left\langle \begin{matrix}\text{First Row}\\ \text{Number}\end{matrix} \right\rangle , \left\langle \begin{matrix}\text{Last Row}\\ \text{Number}\end{matrix} \right\rangle , \left\langle \begin{matrix}\text{Field}\\ \text{Type}\end{matrix} \right\rangle$$

where the Field Identifier is a positive integer value from 1 to 4. The First and Last Row Numbers are positive integer values from 0 to 31 defining the field width. The Field Type is a positive integer value from 1 to 4, and these field types are:

1 – Normal with cursor

2 – Reverse with cursor

3 – Normal without cursor

4 – Reverse without cursor

After issuing a FIELD statement, control is passed to the defined field, i.e., any subsequent output of a PRINT statement will appear in the last defined field. For example, let us define field type 2 as rows 16 to 31.

```
FIELD 2, 16, 31, 2
```

The output of the next PRINT statement

```
PRINT A
```

would appear at the next available line of FIELD 2.

Suppose we want to direct the output of the PRINT statement to a specific row (18) and column (10) number of display screen. The reference to a row number in the PRINT statement must be relative to the first row number of the last defined field. Since the last defined field (FIELD 2) starts at row 16, and we want to output to row 18, the referenced row number in the PRINT statement will now be 2, or the difference between 18 and 16. As a general rule, we can always determine the row number of the PRINT statement as follows:

Row Number of PRINT Statement =
Row Number of Output – First Row Number of Defined Field

The output of the following PRINT Statement

```
PRINT @ (2, 10) A
```

would appear at row 18 and column 10 of the display screen. Note that field identifier 1 can only be type 1 (Normal with cursor).

When several fields are defined, control can be passed from one field type to another by the CFIELD statement. The general form of CFIELD statement is

CFIELD <Field Identifier>

where Field Identifier is a positive integer value from 1 to 4. For example,

FIELD 3, 10, 20, 3
CFIELD 2

Would cause the output of subsequent PRINT statements to appear in FIELD 2.

6.4 REVERSE STATEMENT

The REVERSE statement is used for generating a mirror image of a variable (8 or 16 bit length). The general form of the REVERSE statement is

REVERSE <VARIABLE>

Examples of the REVERSE statement and its meaning

REVERSE X1! $\boxed{0000000000111010}$ X1! Before Reverse

$\boxed{0101110000000000}$ X1! After Reverse

6.5 ROTATE STATEMENT

The ROTATE is used for shifting the contents of variable (8 or 16 bit length) without losing the most significant bits (MSB) or the least significant bits (LSB) (depending on the direction of the shift). The general format of the ROTATE statement is

ROTATE <Number of Bit Positions> <R or L> <Variable>

where R is for right and L is for left shift. The number of bit positions to rotate is an integer number and is taken module 8 or 16 depending on the type of variable. Examples of the ROTATE statement and its meaning

ROTATE 13L A1! $\leftarrow\boxed{1000000000000101}\leftarrow$ A1! Before Rotate

$\boxed{1011000000000000}$ A1! After Rotate

6.6 SHIFT STATEMENT

The SHIFT statement is used for shifting the contents of a variable. Unlike the ROTATE statement the MSB's or LSB's are lost during the shift (depending on the direction of the shift). The general format of SHIFT statement is

SHIFT <Number of Bit Positions> <R or L> <Variable>

where R is for right and L is for left shifts. The number of bit positions to shift is an integer number, and is taken modulo 8 or 16 depending on the type of variable. Examples of the SHIFT statement and its meaning

SHIFT 5R X1# $\boxed{10100011}$ X1# Before Shift

$\boxed{00000101}$ X1# After Shift

6.7 XDIM STATEMENT

Suppose we are interested in making a data transaction between memory or I/O devices. The XDIM statement assigns a subscripted variable to a location in memory. The general form of XDIM statement is

XDIM <(Name (d_1, ... , d_3)> <Integer Constant>

where the name is a subscripted variable name, and d's are positive integer constants which indicate the number of subscripts. The Integer Constant is the memory location of the first element of the array. Let us illustrate a simple program to transfer a block of data between memory locations:

```
EXAMPLE [   ]

1  XDIM A# (256) #400
2  XDIM A1# (256) #600
3  FOR I = 0 to 255
4  A1# (I) = A#(I)
5  NEXT I
END
```

This program would transfer a block of data from memory locations #400 - #4FF to #600 - #6FF.

6.8 PSCRN STATEMENT

The PSCRN statement will output on the line printer a copy of what is on the display screen. The general firm of PSCRN statement is

PSCRN

Note that since the display screen only takes up half a page, two copies of the display screen may be printed on a single page.

6.9 EJECT STATEMENT

The EJECT statement causes the line printer to skip to the top of the next page. The general form for the EJECT statement is

EJECT

Chapter 7
FUNCTION DEFINITION

7.1 DEF STATEMENT

In RSZ BASIC, a function is equivalent to a subroutine in FORTRAN or a procedure in PL/I, Pascal, etc. A function is defined by the DEF statement. The general form of DEF statement is

DEF <Function - Name> [<Argument list>]

where function name may be any alphanumeric characters. The argument list allows communication of data between functions. Note that the argument list is not necessary, but the function name must be followed by "[ ]". Any function may be ended with the statement

END

However, the END statement is not necessary since by pressing the ESC key the system automatically inserts an END at the physical end of each function. Let us illustrate the mechanics of function definition by writing a function AVERAGE [A, A1( ), N] which computes an average of N numbers in array A1( ) and return the average value in A. The following functions, AVERAGE and MAIN program, excercise the above computation.

```
DEF MAIN [  ]

1  DIM X (100)
2  FOR I = 0 to 99
3  X(I) = I
4  N = I
5  NEXT I
6  CALL AVERAGE [A, X( ), N]
7  PRINT "AVERAGE = ", A
8  DROP A, X( ), N
9

DEF AVERAGE [A, A1( ), N]

1  S = 0
2  FOR I = 0 to N - 1
3  S = S + A1(I)
4  NEXT I
5  A  =  S/N
6  RETURN
7
```

The CALL statement in the MAIN program passes the program control from function MAIN to AVERAGE. The general form of this statement is:

CALL <Function - NAME> [<Argument List>]

The number of parameters in the argument list must agree with that of the called function, and the parameters in the argument list must be variables. If no argument list is needed, the function name must be followed by "[ ]".

The RETURN statement in the above function causes control to be returned to the calling program. The general form of this statement is

RETURN

Unlike FORTRAN, in RSZ BASIC a function (Subroutine) need not be ended by entering a RETURN statement, since the system simply inserts a RETURN at the physical end of each function.

7.2 LIBRARY FUNCTIONS

The RSZ BASIC has ten library functions (see Table 7-1). A library function takes the general form,

Function - Name (Argl, ... , Arg N)

Where function name is that defined in Table 7-1. To make use of a library function one needs only to write the name of the function followed by parentheses enclosing the arguments. For example,

Y = SQR (X + 5)

will cause square root of X + 5 to be computed and the result to be assigned to Y.

Table 7-1
Library Functions

Function	Description
SQR (arg_1)	Square Root of arg_1. Argument value must be positive.
MOD (arg_1, arg_2)	Remainder of Arg_1 divided by Arg_2.
FIX (arg_1)	The fraction part of the real number is truncated. FIX (2.7) = 2; FIX (-2.7) = -2.
CINT (arg_1)	Ceiling, the next largest integer CINT (2.7) = 3; CINT (-2.7) = -2.
INT (arg_1)	Floor, the next smallest integer INT (2.7) = 2; INT (-2.7) = -3.
LEN (arg_1)	Length of the character string in arg_1.
SIN (arg_1)	Sine of arg_1.
COS (arg_1)	Cosine of arg_1.
TAN (arg_1)	Tangent of arg_1.
ARCTAN (arg_1)	Arctangent of arg_1.

7.3 ANGLE STATEMENT

The measure of an angle, represented by numerical value of arg_1 in the library trignometry functions, can be expressed in degrees, radians or grads. The choice of units is selected by the ANGLE statement. The general form of ANGLE statement is

ANGLE <D or R or G>

where D, R and G are symbols expressing the measure of an angle in degrees, radians or grads respectively. For example,

ANGLE D

will signal to the system that all subsequent angle values in the trignometry functions are expressed in degrees. Note that in RSZ BASIC the system default measure of angle value is expressed in radians. Also, the R in

ANGLE R

is optional; that is

ANGLE is equivalent to ANGLE R.

Chapter 8
FUNCTION EDITING

In the previous chapters we examined how we can write a function in the RSZ BASIC. We will now consider how we might change a function after it has been entered into the workspace. The RSZ BASIC editor program provides such capability. To examine the different ways of editing a function using the RSZ BASIC editor program, we will first enter the following function SUM into the workspace:

```
DEF SUM [   ]

1  REM - - - INTEGER SUMMATION PROGRAM
2  DIM K (100)
3  INPUT "NUMBER OF VALUES" N
4  S = 0
5  FOR I = 0 TO N - 1
6  INPUT "NEXT VALUE" K(I)
7  S = S + K(I)
8  NEXT I
9  PRINT "SUM = ", S
```

This function will be used in our discussion of various ways of editing a function throughout this chapter.

8.1 DISPLAYING A FUNCTION

An entire function or only part of a function can be displayed by entering the following command:

```
LIST <NUMBER OF LINES>
```

Let's suppose we want to display the entire function. How is this done? The first step is to open up the function by entering DEF <Function name>, followed by depressing the RETURN key.

```
>DEF SUM [   ]
10
```

Now enter the LIST command without the number of lines.

```
10 LIST
```

The system will display the entire function.

```
SUM [   ]

1  REM - - - INTEGER SUMMATION PROGRAM
2  DIM K (100)
3  INPUT "NUMBER OF VALUES" N
4  S = 0
```

```
5  FOR I = 0 TO N - 1
6  INPUT "NEXT VALUE" K(I)
7  S = S + K(I)
8  NEXT I
9  PRINT "SUM = ", S
END
10
```

Suppose we are interested in only a single line, say, 4? The display command is very simple. Just enter 4 on the current line as follows (the system is currently pointing to line 10):

```
10 4
```

The system will display line 4.

```
4 S = 0
```

To display the next line, simply press RETURN. Suppose we are interested in displaying only a part of a function, say, 3 lines? Simply enter the LIST command with the number of lines followed by a blank as follows: (the system is currently pointing to line 4)

```
4  LIST 3 (Type over the current text)
```

The system will display three lines starting with the current line.

```
4  S = 0
5  FOR I = 0 TO N - 1
6  INPUT "NEXT VALUE" K(I)
```

Notice that the text of line 4 was not altered by the list command.

8.2 ADDING A LINE

The nine lines of the function as presently written gives the sum of integers entered by the user. Let's suppose we have decided to add a tenth line which will give the average value of the integers entered by the user. How is this done? The first step is to open the function by entering DEF SUM [   ] followed by a RETURN.

```
> DEF SUM [   ]
10
```

Notice that the system responds with 10. In general the next available line number will be returned. Now enter:

```
10 A = S/N:  PRINT "AVERAGE" = , A
11
```

and the system has responded with a 11, waiting for the next line input. Since we don't want to add anything further, the ESC key can be depressed to signal the end of the function. A display of the function now shows that line 10 has indeed been added.

```
SUM [   ]
1   REM - - - INTEGER SUMMATION PROGRAM
2   DIM K (100)
3   INPUT "NUMBER OF VALUES" N
4   S = 0
5   FOR I = 0 TO N - 1
6   INPUT "NEXT VALUE" K(I)
7   S = S + K(I)
8   NEXT I
9   PRINT "SUM = "; S
10  A = S/N:  PRINT "AVERAGE = ", A
END
```

8.3 REPLACING A LINE WITH ANOTHER LINE

We have expanded our function to provide the average of the integer values entered by the user. To reflect this change we need to replace our remark on line 1. As before, to replace line 1 we need to open up the function by entering

```
> DEF SUM [   ]
```

The system responds with 11, which we override by entering 1

```
11   1
```

After pressing RETURN, the system displays statement 1

```
1  REM - - - INTEGER SUMMATION PROGRAM
```

Simply press SHIFT - RUBOUT keys; the system erases the current line, and we can now enter the new remark

```
REM - - - INTEGER SUMMATION AND AVERAGING PROGRAM
```

Having accepted the change to line 1, the system replies with the next line. Since we do not want to make any further change, we simply press the ESC key to close out the function. A display of the function now shows that line 1 has indeed been replaced.

```
SUM [   ]
1   REM - - - INTEGER SUMMATION AND AVERAGING PROGRAM
2   DIM K (100)
3   INPUT "NUMBER OF VALUES" N
4   S = 0
```

```
5   FOR I = 0 TO N - 1
6   INPUT "NEXT VALUE" K(I)
7   S = S + K(I)
8   NEXT I
9   PRINT "SUM = ", S
10  A = S/N:  PRINT "AVERAGE = ", A
END
```

8.4 INSERTING A LINE BETWEEN TWO OTHER LINES

Suppose we want to insert between lines 8 and 9 statements whose purpose is to print input values in array K. This can be accomplished in the following way. First open up the function and type in some number between 8 and 9, say 8.1, after response 11:

```
> DEF SUM [   ]
11  8.1
```

Any number will do as long as it falls between the number of the two lines where the insertion is to be made. The system returns 8.1 and we can enter the code, which when encountered during execution will cause the values of array K to be printed.

```
8.1  FOR I = 0 TO N - 1
8.2  PRINT K(I)
8.3  NEXT I
8.4
```

The next line number provided by the system is "one greater" in its last significant digit, to provide for still other entries between line 8 and 9. Since there is to be no additional entry at this time, we close out the function by pressing the ESC key. In response to the ESC key, the line number and references of all GO TO statements are automatically renumbered by the computer, as seen in the following display:

```
SUM [   ]
1   REM - - - INTEGER SUMMATION AND AVERAGING PROGRAM
2   DIM K (100)
3   INPUT "NUMBER OF VALUES" N
4   S = 0
5   FOR I = 0 TO N - 1
6   INPUT "NEXT VALUE" K(I)
7   S = S + K(I)
8   NEXT I
9   FOR I = 0 TO N - 1
10  PRINT K(I)
11  NEXT I
12  PRINT "SUM = ", S
13  A = S/N:  PRINT "AVERAGE = ", A
END
```

8.5 DELETING A LINE

Suppose we want to delete line 12. As usual, we first open the function, and then list line 12.

```
> DEF SUM [    ]
14  12
12  PRINT "SUM = ", S
```

The computer is now waiting for us to do something with line 12. Pressing the SHIFT – RUBOUT keys, followed by a RETURN key removes the text on line 12. Next we ask for the display of the function.

```
LIST
1   REM - - - INTEGER SUMMATION AND AVERAGING PROGRAM
2   DIM K (100)
3   INPUT "NUMBER OF VALUES" N
4   S = 0
5   FOR I = 0 TO N - 1
6   INPUT "NEXT VALUE" K(I)
7   S = S + K(I)
8   NEXT I
9   FOR I = 0 TO N - 1
10  PRINT K(I)
11  NEXT I
12
13  A = S/N:  PRINT "AVERAGE = ", A
END
```

Notice that line 12 is still there, but is blank. To get rid of a blank line, we simply close out the function by pressing the ESC key. The blank lines are removed and the line numbers and references of all line numbers are renumbered. A display of function now shows that the blank line indeed has been removed.

```
SUM [    ]
1   REM - - - INTEGER SUMMATION AND AVERAGING PROGRAM
2   DIM K (100)
3   INPUT "NUMBERS OF VALUES" N
4   S = 0
5   FOR I = 0 TO N - 1
6   INPUT "NEXT VALUE" K(I)
7   S = S + K(I)
8   NEXT I
9   FOR I = 0 TO N - 1
10  PRINT K(I)
11  NEXT I
12  A = S/N:  PRINT "AVERAGE = ", A
END
13
```

8.6 EDITING OF PART OF A LINE

How do we change or add a few characters in a line without having to retype the entire line? For example, suppose we want to print the summation of input integers on line 12. Again, we open the function and then list line 12.

```
> DEF SUM [   ]
13  12
12  A = S/N:  PRINT "AVERAGE = ", A
```

To print the summation of input integers we want to add "SUM = ", S between the PRINT and "AVERAGE = " on line 12, we first move the cursor onto character "

```
12  A  =  S/N:  PRINT "AVERAGE = ", A
```

Next, press BREAK key to move all characters, including the one over the cursor, one position to the right. Press BREAK key several times to open up enough space to enter "SUM = ", S. Now enter

```
12 A = S/N:  PRINT "SUM = ", S, "AVERAGE = ", A
```

Next press RETURN. A display of function now shows that line 12 has indeed been updated.

```
 SUM [   ]
1   REM - - - INTEGER SUMMATION AND AVERAGING PROGRAM
2   DIM K (100)
3   INPUT "NUMBER OF VALUES" N
4   S = 0
5   FOR I = 0 TO N - 1
6   INPUT "NEXT VALUE" K(I)
7   S = S + K(I)
8   NEXT I
9   FOR I = 0 TO N - 1
10  PRINT K(I)
11  NEXT I
12  A = S/N: PRINT "SUM = , S, "AVERAGE = ", A
END
```

8.7 EDITING THE HEADER

The header of a function can be changed in exactly the same way as any other line by using the line number 0. Suppose we want to add an argument to the header. As usual, we open the function and then list line 0.

```
> DEF SUM [   ]
13  0
SUM [   ]
```

Simply follow the same procedure used for editing part of a line,

```
0  SUM [A]
1  REM - - - INTEGER SUMMATION AND AVERAGING PROGRAM
```

and close out the function.

8.8 DELETING A FUNCTION

How do we remove a function from the workspace? The first step is to open up the function

```
> DEF SUM [   ]
13
```

and then enter the DELETE command followed by a RETURN.

```
13 DELETE
```

Now any reference to function SUM will result in an error.

Chapter 9
MANAGING THE WORKSPACE

The user is assigned a clean workspace by the system at sign-on. In this workspace the system maintains a list of all active functions and variables entered after sign-on. In this section, we will follow a series of exercises designed to demonstrate how the workspace can be manipulated by you, the BASIC user. We will accomplish this by the use of system commands. An alphabetical list of all system commands is given in Appendix B.

9.1 WORKSPACE CONTENTS

As we pointed out earlier, the workspace is empty at sign-on. To verify that this workspace is empty, we can use the following commands:

```
> FNS
> VARS
```

and we see there is nothing in the active workspace.

The FNS command produces a listing of functions available in the workspace. The VARS command produces a listing of the active variables in the workspace.

Since the purpose of this chapter is to teach you how to save functions and variables for future use, we will need some example functions. To begin, let's enter the function CIRCLE:

```
> DEF CIRCLE [D]
1  REM - - - COMPUTE AREA OF A CIRCLE
2  A  = 2 * PI * ((D/2) * (D/2))
3  PRINT "AREA OF A CIRCLE = ", A, "OF DIAMETER = ", D
```

Our listing of the functions now shows

```
> FNS
CIRCLE [
```

Let's enter a second function,

```
> DEF RECT [H, W]
1  REM - - - COMPUTE AREA OF A RECTANGLE
2  A1 = H*W
3  PRINT "AREA OF A RECTANGLE = ", A1
```

A new listing of functions,

```
> FNS
CIRCLE [ RECT [
```

now indicates that RECT indeed has been stored.

Now let's set a couple of variables,

```
> W = 10
> H = 20
```

A listing of variable shows that W and H are in storage.

```
> VARS
W0  H0
```

9.2 SAVING AND RECOVERING A WORKSPACE

Suppose we are finished entering functions and variables and now want to preserve them for later use. The system command SAVE accomplishes this. The SAVE command reserves the current workspace under the specified file name on disk 0 or 1, overwriting any previously entered file with the same name. The general form of the SAVE command is

```
SAVE <file name>
```

To save the previously entered functions and variables, we'll use the file name BPL1

```
> SAVE BPL 1
```

Now, we need a list of all the saved workspaces so that we know what we have in our library. The LIB command does this. The LIB command lists the names of all files saved on disk 0 and 1.

```
> LIB
BPL1
```

Note that what we saved on disk is an image of the active workspace.

Let's now get a clear workspace without having to sign-off and then sign-on. This can be accomplished simply by pressing the CLR key. Care should be taken when pressing theCLR key, since everything in the workspace is thus erased.

Suppose that on a following day we are ready to do some work with CIRCLE and RECT functions. Remember that we have an exact image of our previous day's workspace stored on disk with the name BPL1. To recover this image, enter the system command

```
> LOAD BPL1
```

The LOAD command reads the specified file from disk and replaces the contents of the active workspace with the material in the workspace being loaded. A listing of functions and variables,

```
> FNS
CIRCLE [ RECT [
> VARS
W0  H0
```

shows that our function and variables are still available to us.

Let's check the values of our two variables to verify that they are unaltered:

```
> ? W, H
10    20
```

Note that we can access only one workspace at a time using the LOAD command.

9.3 DELETING A SAVED WORKSPACE

Suppose we want to delete a workspace in our library. The DELETE command does this. The general form of the DELETE command is

```
DELETE <file name>
```

The DELETE command scans the directories of disk 0 and 1 for the specified file name and removes it from disk.

```
> DELETE BPL1
```

BPL1 is now gone, as shown by

```
> LIB
```

9.4 THE WSID COMMAND

Suppose we have forgotten the name of the workspace we started out with. If so, the WSID (workspace identification) command will remind us.

```
> WSID
BPL1
```

Note that the SCRATCH is the default name assigned to the workspace by the computer at sign-on.

9.5 THE SIZE COMMAND

The workspace size at sign-on is 16k. However, the size of the workspace continues to shrink as we enter material into it. Suppose we need to enter a large function into our workspace. The SIZE command will tell us how much available memory remains in the workspace.

```
> SIZE
3F16
```

Note that the number is hexidecimal. Let's pretend that we don't have enough memory to fit the size of our function. We can generate additional workspace by removing some of the functions and variables.

```
> DROP H, W
```

Variables H and W are now eliminated from our workspace

```
> DEF CIRCLE [   ]
1 DELETE
```

and the function CIRCLE is now gone from our workspace, as shown by

```
> FNS
RECT [
```

Thus our workspace has increased in size, as shown by

```
> SIZE
3F93
```

9.6 THE CONTINUE COMMAND

The CONTINUE command is like the SAVE command, except that it always saves the active workspace under the file name CONTINUE. So you can save a workspace with this command, but care must be taken since it always replaces its prior contents.

9.7 THE MEM COMMAND

At sign-on, the lower and upper memory locations of the workspace are defined as #4000 and #7FFF respectively. Suppose we have forgotten or want to redefine the lower and/or upper memory locations of the workspace. The MEM command

```
> MEM
4000  7FFF
FIRST ADDRESS
>
```

will display the current lower and upper memory location of the workspace, which are #4000 and #7FFF respectively, followed by a system message "FIRST ADDRESS." Let's suppose that we want to lower the current upper memory location from #7FFF to #5FFF, and retain the lower memory location #4000 of the workspace. By pressing the RETURN key, the system will retain the current lower limit and will respond with

```
LAST ADDRESS
>
```

Now, enter the new upper memory location value #5FFF, followed by a RETURN key

```
> 5FFF
>
```

The lower and upper memory locations of the workspace are now #4000 and #5FFF respectively, as shown by

```
> MEM
4000   5FFF
```

Note that the lower and upper memory location values must always be between #4000 and #7FFF.

Appendix A
SUMMARY OF RSZ BASIC PROGRAMMING LANGUAGE STATEMENTS

An alphabetical listing of RSZ BASIC Programming Language statements and their general form appears in Table A-1.

Table A-1
Listings of RSZ BASIC Statements

STATEMENT	GENERAL FORM
ANGLE	ANGLE <D or R or G> D – Degree; R – Radians; G – Grads (R is optional)
CALL	CALL <Function-Name> [<Var_1, ... , Var_n>]
CFIELD	CFIELD <Field Identifier>
CLEAR	CLEAR
DEF	DEF <Function-Name> [<Var_1, ... , Var_n>]
DIM	DIM <Variable> (<Int_1, ... , Int_3>)
DROP	DROP <Var_1, ... , Var_n>
EJECT	EJECT
END	END
FIELD	FIELD ⟨Field Identifier⟩, ⟨First Row Number⟩, ⟨Last Row Number⟩, ⟨Field Type⟩
FOR	FOR <Unsubscripted Var.> = <Expression> TO <Expression> STEP <Expression>
GO TO	GO TO <Line Number>
IF-THEN-ELSE	IF <Expression> <Relational> <Expression> THEN <Statement> ELSE <Statement>
INPUT	INPUT <@(<Expr_1, Expr_2>) <"Any String of Characters"> <Var_1, ... , Var_n>
LET	LET <Variable> = <Expression>
PRINT	PRINT <@(Expr_1, Expr_2)> <Format Specification> <Var_1, ... , Var_n>

Table A-1
Listing of RSZ BASIC Statements (continued)

STATEMENT	GENERAL FORM
PSCRN	PSCRN
REM	REM <Any String of Characters>
RETURN	RETURN
REVERSE	REVERSE <Variable> 8 or 16 bit Integer Variable
ROTATE	ROTATE <Integer> <R or L> <Variable> 8 or 16 bit Integer Variable R - Right; L - Left
SHIFT	SHIFT <Integer> <R or L> <Variable> 8 or 16 bit Integer Variable R - Right; L - Left
TAB	TAB <Int_1, ... , Int_n>
XDIM	XDIM <Letter> (<Int_1, ... , Int_3>) <Expression>

Appendix B
SYSTEM COMMANDS

An alphabetical listing of RSZ BASIC system commands and their description is given below:

System Command	Description
CONTINUE	Save active workspace under the file name CONTINUE in library.
CPDISK	Copy the content of disk #0 to disk #1.
DELETE <file name>	Remove the particuiar file from library.
EJECT	Advance line printer to top of next sheet.
FORMAT	Format the disk to allow storage of files. Note all disks must be formatted before use on the RSZ BASIC system.
FNS	List names of all defined functions in the active workspace.
LIB	List the names of workspace in the library.
LOAD <file name>	Replace the current active workspace with the workspace stored under the particular file name in the library.
MEM	Redefine or check the memory boundaries of the active workspace. Depressing the RETURN or the ESC key retains the current limit.
PSCRN	Generate hardcopy of CRT screen.
SAVE <file name>	Store the active workspace under the particular file name in the library.
SIZE	Displays in hexiclemical the available memory remains in the active workspace.
VARS	List the names of active variables in the workspace.
WSID	Displays the name of the active workspace.

Appendix C
TERMINAL COMMANDS

An alphabetical listing of all terminal commands and their description is given below:

Terminal Command	Description
BACKSPACE	Positions the cursor one space to the left.
BREAK	Inserts a blank between characters.
CLR	System initialization.
C/S	Clears CRT display screen.
CNTL-@	Prints a copy of what is on the display screen on the line printer.
CNTL-N	Signals the line printer to skip to the top of the next page.
CNTL-H	Positions the cursor one space to the left.
ESC	Terminates the previously entered command.
HOME	Positions the cursor to the upper left corner of the display screen (row 0, column 0).
LF	Linefeed.
RETURN	Enters current line and positions the cursor at the beginning of the next line.
RUBOUT	Replaces the character with a blank.
REPT	Repeats the character entered in conjunction with REPT key on the current line.
SHIFT-RUBOUT	Deletes the current line.
↑ ← → ↓	Moves the cursor one space in the indicated direction.

Appendix D
RESIDENT CONSTANTS

A list of all resident constants and their value is given below:

Name	Value
PI (π)	3.141593
EPS (ϵ)	2.718282

BIBLIOGRAPHIC DATA SHEET

1. Report No. TM 80690	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle The RSZ Basic Programming Language Manual		5. Report Date June 1980
		6. Performing Organization Code
7. Author(s) R. J. Stattel, J. K. Niswander, A. K. Kochhar		8. Performing Organization Report No.
9. Performing Organization Name and Address		10. Work Unit No.
		11. Contract or Grant No.
		13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address NASA/Goddard Space Flight Center Code 743.4 Greenbelt, MD 20771		
		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

The RSZ Basic interactive language developed by NASA/GSFC Sounding Rocket Division Instrumentation Branch is described. The RSZ Basic interpreter is resident in the Telemetry Data Processor, a system dedicated to the processing and displaying of PCM telemetry data. A series of working examples teaches the fundamentals of RSZ Basic and shows how to construct, edit and manage storage of programs.

17. Key Words (Selected by Author(s)) ground support equipment, BASIC, interpreter, telemetry, micro-computer, data processing		18. Distribution Statement	
19. Security Classif. (of this report) Unclas.	20. Security Classif. (of this page)	21. No. of Pages 50	22. Price*

*For sale by the National Technical Information Service, Springfield, Virginia 22151.

GSFC 25-44 (10/77)