

HANIMEX

Rameses®

VERSION  
1:0

# V1 BASIC INTERPRETER



For use with **Rameses** HVC6502

## INSTRUCTION MANUAL

(Incorporating CREATIVISION BASIC)

## ERRATUM

P.26 LINE 22: 30PRINT 'RESULT' (4/2 \*6) RET'N

P.68 LINE 15: ...20 five times...

P.70 LINE 11: 40 PRINT N,S  
LINE 12: 50 N = N+1

P.82 LINE 6: Example: PRINT TAB(5); 'A', TAB  
(10); 'A'

P.95 LINE 26: Example:>100 REM COLOR DEMONSTRATION

P.103 LINE 7: 5 2 d

P.105 4TH PARAGRAPH SHOULD BE:

Example: 10 PRINT JOY (1)  
20 GOTO 10  
RUN  
3  
.  
.  
.

P.109 LINE 13: Example: POKE 1000, 48  
This makes the content at the address location  
1000 to have the value 48. If you type  
PRINT PEEK (1000)

P.123 AFTER LINE 4 SHOULD ADD:  
0. CASSETTE LOADING ERROR

P.127 LINE 5: 50 IF X,<>Y THEN 30  
LINE 9: ? 3  
? 9  
3

Information in this document is subject to change without notice and does not represent a commitment on the part of Video Technology Ltd. or Hanimex (PTY) Ltd. It is against the law to copy Creativision BASIC on cassette tape, disk, ROM, or any other medium for any purpose without the written consent by Video Technology Ltd.

© Video Technology Ltd.

## **LIMITED WARRANTY**

Neither V. T. Ltd. nor Hanimex Pty. Ltd. shall have any liability or responsibility to purchaser or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by this product, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of this product. This product will be exchanged within six months from date of purchase if defective in manufacture, labelling or packaging, but except for such replacement the sale or subsequent use of this program is without warranty or liability.

This warranty gives you specific legal rights and you may have additional rights and remedies which either cannot be excluded or limited in law or which have not been excluded or limited by the terms of this warranty and which may vary from state to state.

## FIRST EDITION – 1982

All rights reserved. Reproduction or use, without express permission, of editorial or pictorial content, in any manner, is prohibited. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

© Copyright 1982, Video Technology Ltd.

# **TABLE OF CONTENTS**

<b>PREFACE</b>	Page 7
<b>CHAPTER 1</b>	Page 9
<b>INTRODUCTION</b>	
— What is a computer	
<b>CHAPTER 2</b>	Page 13
<b>DEFINITION</b>	
— Programming Concept	
— BASIC Interpreter	
<b>CHAPTER 3</b>	Page 17
Getting Ready with the Rameses Computer System	
<b>CHAPTER 4</b>	Page 23
<b>TO START PROGRAMMING</b>	
— Immediate-Execution Mode	
— Programming Mode	
— BASIC Program Formats	
— Line editing	
— LIST	
— Delete a line from the program	
— LLIST	
— NEW	
<b>CHAPTER 5</b>	Page 31
<b>NUMBER &amp; VARIABLES</b>	
— Numeric Constants	
— Numeric Variables	
— LET Statement	
— REM Statement	
— ABS Function	
— SGN Function	
— RND Function	
<b>CHAPTER 6</b>	Page 39
<b>DATA MANIPULATION</b>	
— Arithmetic	
— Relational	
— Logic	
— Functional	

<b>CHAPTER 7</b>	<b>Page 49</b>
<b>STRING FUNCTIONS</b>	
– Character strings	
String Constants	
String Variables	
– String Manipulation	
– String Functions:	
LEFT \$	
RIGHT \$	
MID \$	
CHR \$	
STR \$	
LEN	
VAL	
ASC	
– String Comparison	
<b>CHAPTER 8</b>	<b>Page 59</b>
<b>SYSTEM CONTROL</b>	
– STOP	
– END	
– CONT	
– CNTL/C	
– RESET	
<b>CHAPTER 9</b>	<b>Page 65</b>
<b>LOOPING &amp; CONDITION BRANCHING</b>	
– IF ... THEN	
– FOR ... NEXT	
– GOSUB/RETURN	
– GOTO	
<b>CHAPTER 10</b>	<b>Page 73</b>
<b>ARRAYS</b>	
– DIM	
<b>CHAPTER 11</b>	<b>Page 77</b>
<b>INPUT/OUTPUT COMMANDS</b>	
– INPUT	
– PRINT	
– LPRINT	
– TAB	
– READ/DATA	
– RESTORE	

<b>CHAPTER 12</b>	
TAPE STORAGE	Page 85
– CSAVE	
– CLOAD	
– CRUN	
<b>CHAPTER 13</b>	
GRAPHIC & SOUND FUNCTION	Page 91
– CLS	
– COLOR	
– CHAR	
– PLOT	
– SOUND	
– JOY	
<b>CHAPTER 14</b>	
SYSTEM MEMORY ACCESS	Page 107
– PEEK	
– POKE	
<b>CHAPTER 15</b>	
RAMESES SYSTEM EXPANSION	Page 111
<b>APPENDIX:</b>	
(A) TABLE OF STATEMENT	Page 115
(B) ERROR MESSAGE	
(C) TYPICAL PROGRAM EXAMPLES	
(D) ASCII Code	





## Preface

This manual is intended to serve the beginner as an introduction to programming in BASIC on Rameses Computer System. Starting at the most elementary level, it introduces the reader to the fundamentals of BASIC, and the procedures of creating programs on the Rameses Computer System. It covers all the concepts and statements needed for fundamental programming in BASIC and presumes no prior knowledge of BASIC or of programming in general.

The reader should try out new statements and programming concepts as they are introduced by keying in and executing the example programs. One should experiment with the example programs by making changes to them, predicting the effects of the changes, and then confirming or correcting one's knowledge based on the observed effects. As soon as possible, one should begin to write programs that solve simplified problems within one's particular area of interest. Only in this manner can the concepts and capabilities introduced here become practical knowledge.

There is nothing that can be done from the keyboard that can damage a Rameses Computer System. Cautions are included in the text when statements that might destroy data files are introduced. Thus, the reader should feel free to experiment with the Rameses Computer System at every stage of learning.

In general, the reader should follow the sequence of presentation in the text; however, there are other alternatives. Chapter 12 which discusses the use of tape storage may be read at anytime when the reader wishes to save a program on the cassette tape. Although this manual is specially written for one who wishes to learn to program in BASIC on Rameses Computer System, it can also serve as a general introduction to programming in BASIC on any system. However, the reader must be aware that the BASIC language has many forms. There are slight differences between one implementation of BASIC and another.

Finally, we hope that this manual will help you to learn the fundamentals of BASIC and the fundamental concepts in computer programming.

**HAVE FUN WITH YOUR RAMESES COMPUTER SYSTEM!!**



# **CHAPTER 1**

## **INTRODUCTION**

**— What is a computer**



## What is a computer?

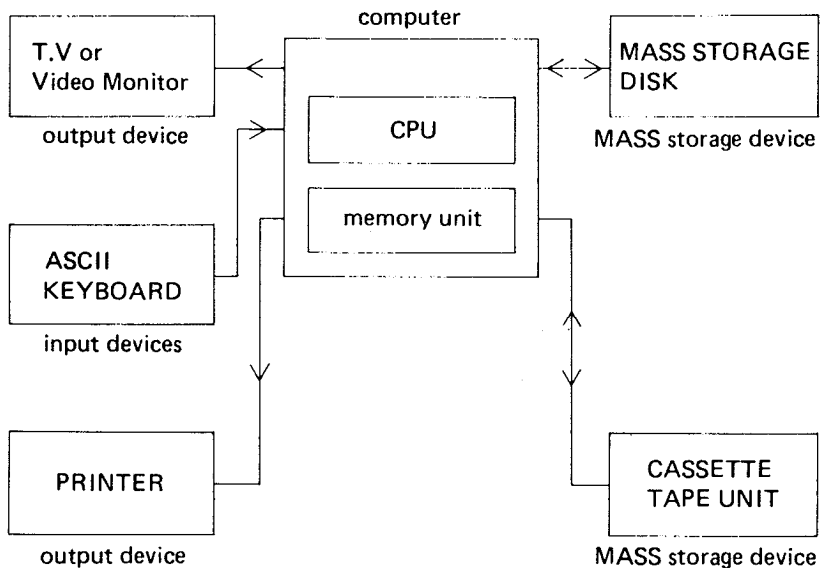
Computers are devices which perform various operations based on instructions which have been given by people who use them.

A computer system consists of the following units in general:

- i) central processing unit (CPU) — it performs operations specified in the instruction, such as arithmetic and logic operations; It can be considered as the brain of the computer system.
- ii) memory unit — it stores instructions and informations generated by the computer or given by the user. This unit is resident inside the computer, and the CPU gets information from it directly.
- iii) mass storage unit — it stores instructions and informations generated by the computer or given by the user except that this unit is resident outside the computer. For example, the tape storage unit and the floppy disk unit are memory storage units. Informations stored in these units have to be transferred to the memory unit before the CPU can process them.
- iv) input device — it allows the user to enter instructions or informations to the computer, e.g. keyboard.
- v) output device — it receives information or results sent from the computer, e.g. printer.

The input and output devices together act as a two-way communication channel between the computer user and the computer system.

Although the size of computer systems varies from one to another, all practical computer systems require the above mentioned units in general.



**Fig. 1** Configuration of a Computer System in general.

# CHAPTER 2

## **DEFINITION**

- **Programming Concepts**
- **BASIC Interpreter**





## PROGRAMMING CONCEPTS

The process of specifying a set of instructions for a computer is called programming, and the set of instruction is called a program. The individual preparing a program is called a programmer.

There are two steps involved in preparing a program for a computer. First, the programmer must know what instructions to specify, and the order in which to specify them. Second, he must be able to communicate his specifications to the computer. Communication is accomplished by means of a programming "language" which the programmer writes, and the computer "reads" to decide what to do.

There are many programming languages in use today. Some are designed for very specialized applications, and some are designed for more general use. BASIC is a language in the latter category.

## BASIC INTERPRETER

BASIC, an acronym for Beginner's All-purpose Symbolic Instruction Code, has a simple English vocabulary and few grammatical rules while resembling ordinary mathematical notation. Although the language is basically simple, it is a powerful language, providing arithmetic capabilities, logic comparison, subscripting, common trigonometric functions, lists, arrays, and alphanumeric character string manipulation.

Programs written in BASIC are translated by a language translation program into a language that the central processor unit understands. This language translation program is called the BASIC Interpreter, and is resided in the BASIC cartridge provided.

The next chapter will tell you how to set up your Rameses Computer System, and the following Chapters will explain the BASIC statements with examples provided to help you to understand them.



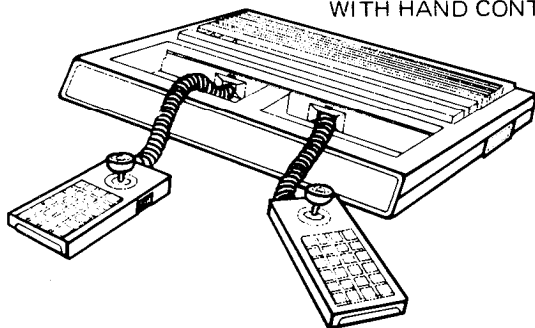
# **CHAPTER 3**

## **Getting Ready with the Rameses Computer System.**

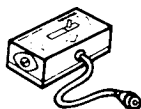


## OPEN THE GIFT BOX AND YOU WILL FIND THESE ITEMS

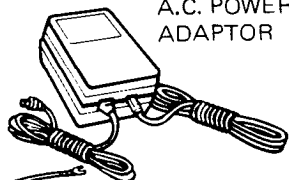
CONSOLE UNIT  
WITH HAND CONTROLS



AERIAL  
SWITCH BOX



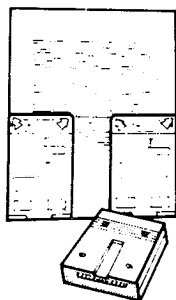
A.C. POWER  
ADAPTOR



BALUN



GAMES CARTRIDGE  
WITH HAND CONTROL  
OVERLAYS AND  
INSTRUCTION BOOKLET

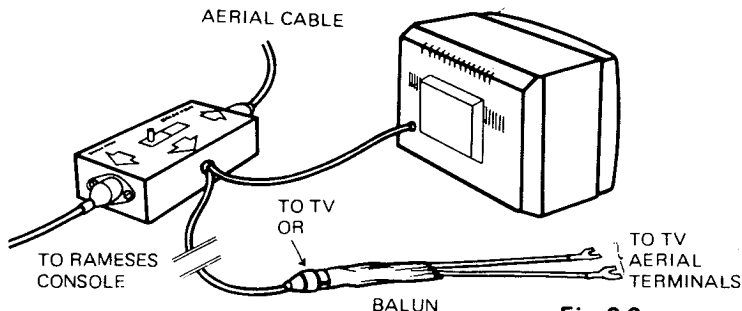


## AERIAL SWITCH BOX

The Aerial Switch Box provides you with a convenient means in using your television set for either normal TV programmes or for games.

- Remove the co-axial aerial cable from your television set and connect it to the Switch Box.
- Connect the co-axial cable from the switch box to the aerial socket of your television set.
- Connect the co-axial cable from the Main Unit to the Switch Box.

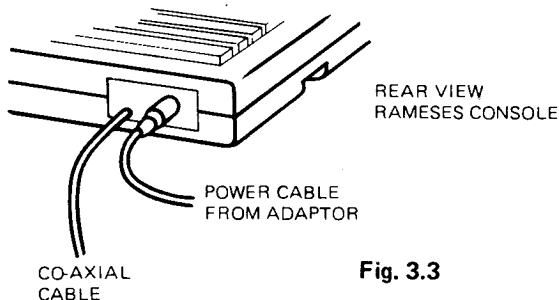
Once the installation is done, you can push the switch to make your choice.



**Fig. 3.2**

## STEPS TO SET UP FOR A COMPUTER SYSTEM

1. Make sure that the Main Unit Power Switch is off.
2. Connect the AC adaptor power plug to Rameses Unit power socket.
3. Plug the wall plug of the AC Adaptor into a normal wall AC outlet.



**Fig. 3.3**

4. Push the switch on the Switch Box to Game.
5. Insert the BASIC interpreter cartridge into the Main unit slot.

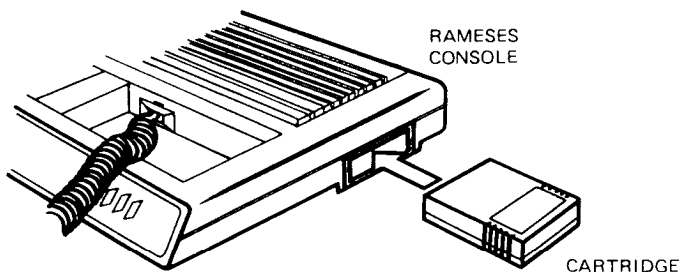


Fig. 3.4

6. Turn on the television set and select the channel ONE (1) on your TV you have chosen. The channel that you choose for the system is usually the spare one that you do not use for regular TV programmes.
7. Push Main Unit Power Switch to ON. If you are setting for the very first time, now tune the channel to receive Ramees BASIC DISPLAY.

Note: If your TV set has an AFT. (automatic Fine Tuning) button, make sure the switch is off when tuning.

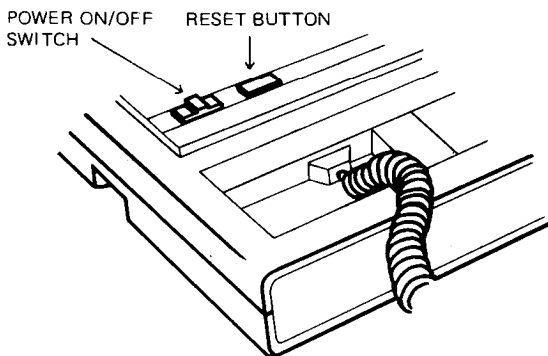


Fig. 3.5

8. Now the computer system has been set up and ready for key in statements.

## **PRECAUTIONS**

1. Keep the Main Unit keyboard and BASIC cartridges away from liquids.
2. Avoid exposing the BASIC cartridges, the Main Unit or keyboard to excessive heat. Please keep them in good ventilation conditions.
3. Switch off power when not in use.
4. Do not drop the Main Unit, keyboard or BASIC cartridge. Handle them with care.
5. Insert BASIC cartridge into the Main Unit slot slowly and make sure power is turned off when inserting or removing BASIC cartridge from the Main Unit.
6. Do not stick fingers into the open end of the cartridges. The static electricity from your fingers may in some cases damage sensitive electronic components in the cartridges.
7. Remove BASIC cartridge from the Main Unit when not in use.

## **SWITCH YOUR TELEVISION SET BACK TO NORMAL USE**

1. Turn off power.
2. Push switch on the Aerial Switch Box to TV antenna.
3. Your television set is now ready for normal use.

## **SUMMARY OF SET UP PROCEDURES**

1. Push the BASIC cartridge into the Main Unit slot properly.
2. The AC adaptor is properly plugged into wall socket and the other end to the Main Unit.
3. Aerial Switch Box is set at Game.
4. All co-axial cable are properly plugged in.
5. The power switch of the Main Unit and your TV set are on.
6. Tune to the right channel.



# **CHAPTER 4**

## **TO START PROGRAMMING**

- Immediate Execution Mode**
- Programming Mode**
- BASIC Program Formats**
- Line editing**
- LIST**
- Delete a line from the program**
- LLIST**
- NEW**



## TO START PROGRAMMING

The computer executes statements (instruction given by the programmer) in two modes, namely the immediate-execution mode and the deferred-execution (program) mode.

### IMMEDIATE-EXECUTION MODE

Any statement which does not have a "line number" at the beginning of the statement will be executed immediately after the **RET'N** (\* **RET'N** should be read as RETURN) key is depressed.

Examples: *PRINT 2+4 RET'N*

```
6
PRINT 2+4, 3/6 RET'N
6          0.5
PRINT "RESULT", (2+4)/(3/6) RET'N
RESULT 12
```

The immediate-execution mode makes the computer work like a calculator, except that this mode of operation of the computer is more powerful than a calculator. It can evaluate more than one expression at a time and also it can handle character strings.

The **PRINT** statement is necessary when using immediate-execution mode, because without it the result would not be displayed on the screen.

Now, try the above examples and observe the result.

### DEFERRED-EXECUTION (PROGRAM) MODE

Any statement which has a "line number" at the beginning of the statement will not be executed after the statement has been ended by the **RET'N** key, only after typing **RUN** and **RET'N**.

Examples: *10 PRINT 2+4 RET'N*

```
RUN RET'N
6
10 PRINT 2+4, 3/6 RET'N
RUN RET'N
6          0.5
```

```

10 PRINT 7+9 RET'N
20 PRINT 2+4, 3/6 RET'N
30 PRINT "RESULT", (2*6) RET'N

```

```

RUN RET'N

```

```

16

```

```

6          0.5

```

```

RESULT    12

```

In the last example, one can see that line number "10" is executed first, and then line number "20" is executed next, and line number "30" is executed last. This is the sequence the computer executes statements in a program. Statements (lines) with smaller line number will be executed earlier than statements (lines) with larger line number.

But one can actually tell the computer which statement to be executed next by using statements such as **GOTO** etc which will be discussed in chapter 9.

Notice that using the **RUN** command, the computer will start to execute the program from the very beginning. If you want to start your program other than the first line.

Use: **RUN** (the line number)

Example: 

```

10 PRINT 7+9 RET'N
20 PRINT 2+4, 3/6 RET'N
30 PRINT "RESULT", (2/4/3/6) RET'N

```

```

RUN 20 RET'N

```

```

6          0.5

```

```

RESULT    12

```

## BASIC PROGRAM FORMATS

1. Every program statement begins with a "line number" eg. **10**, **20** and **30** etc.
2. A line number is an integer ranging from **0** to **9999**.
3. Following the line number is the correct BASIC statement, otherwise the computer will respond with a **"SYNTAX ERROR"** error message.

Example: *10 PRINT 2+3 RET'N*  
*RUN (RET'N)*  
*SYNTAX ERROR*

4. Every program line should be ended by a **RET'N**.

line editing

If you make a mistake while you are entering a program statement, there is a ← key to help you to move back to the wrong entry and make a correction.

Example: *PRIMT*  
 now depress ← and the screen will display  
*PRIM*  
 now depress ← again and the screen will display  
*PRI*  
 now you can enter the correct alphabet.

**LIST**

The **LIST** command is used to display the entire program stored in the memory currently entered; and the statements are listed according to their "line number" in an ascending order.

Example: *47 PRINT "GOOD BYE" RET'N*  
*25 PRINT 2+3 RET'N*  
*33 PRINT 4+3 RET'N*  
*12 PRINT "HELLO" RET'N*  
  
*LIST RET'N*  
*12 PRINT "HELLO"*  
*25 PRINT 2+3*  
*33 PRINT 4+3*  
*47 PRINT "GOOD BYE"*

If a "line number" is entered following the **LIST** command, only the line specified is listed.

Example: *LIST 47 RET'N*  
*47 PRINT "GOOD BYE"*  
will be displayed.

If two line-numbers separated by a comma are entered following the **LIST** command, the statement lines with line-numbers between the two numbers are listed.

Example: *LIST 25, 33 RET'N*  
*25 PRINT 2+3*  
*33 PRINT 4+3*

## delete a programs line

To delete a line from the program all you have to do is to type in the "line number" of the program line which you want to delete (erase) and type **RET'N**. Then that line will be deleted from the program (using the previous example program.):

Example: **12 RET'N**  
**LIST RET'N**  
**25 PRINT 2+3**  
**33 PRINT 4+3**  
**47 PRINT "GOOD BYE"**

Program line number 12 has been deleted. Try to delete line number 33 yourself.

## LLIST

## LLIST

**LLIST** is similar to **LIST** except it lists the whole program on the printer not on the TV screen.

Syntax: **LLIST**.

When **LLIST** is used, the printer and the I/O interface must be ready. For details please refer to the operating manual of the I/O interface.

## NEW

The **NEW** command is used to erase (clear) the entire program currently stored in the memory.

Now, type

**NEW RET'N**

the above example is erased.

Now, enter the **LIST** command see if you can find any listing.

**\*NOTE**

BY NOW, WE ASSUME THAT THE READER HAS FAMILIARIZED HIMSELF WITH THE USE OF THE **RET'N** KEY. STARTING FROM THE NEXT CHAPTER THE **RET'N** ENTRY WILL NOT BE PRINTED OUT IN THE EXAMPLES.



# CHAPTER 5

## **NUMBERS & VARIABLES**

- **Numeric Constants**
- **Numeric Variables**
- **LET Statement**
- **REM Statement**
- **ABS Function**
- **SGN Function**
- **RND Function**



<b>Numeric Constants</b>
--------------------------

**Numeric Constants**

Numeric Constants retain a constant value throughout a program, and can be positive or negative. Numeric Constants can be written using decimal notation as follows:

+2 ,34, 0.432, 3E18, 4E(-35) etc.

The range of numbers which can be handled is

$$10^{-38} \leq n \leq 10^{38}$$

<b>Numeric Variables</b>
--------------------------

**Numeric Variables**

A variable is a data item whose value can be changed during program execution. A numeric variable is denoted by a fixed variable name. Any one of the 26 letters of the alphabet is a legal name for a variable. Value of variables are assigned by **LET**, **INPUT** statements. The value assigned to a variable does not change until the next time a **LET** or **INPUT** statement is encountered, which assigns a new value for the variable or until the variable is incremented by a **FOR** statement. All variables are set equal to 0 before program execution. It is necessary to assign a value to a variable only when an initial value other than 0 is required. However, it is good programming practice to set variables equal to 0 whenever the program requires. This would provide a record of values assigned to variables.

## LET Statement

The **LET** statement assigns the value of the expression on the right of “equal sign” to a variable on the left.

Each **LET** statement is of the form:

**LET** Variable = Expression

This statement does not indicate algebraic equality, but performs the evaluation within the expression (if any) and assigns the resulted value to the indicated variable.

Example: *10 LET A = 5*  
*20 LET B = 20*  
*30 LET C = 60*  
*40 LET A = A+5*  
*50 PRINT A+B+C*  
*60 END*  
  
*RUN*  
*90*

In line **40**, the old value of **A** is increased by **5** and becomes the new value of **A**.

The **BASIC** interpreter allows the user to omit the word **LET** from the **LET** statement. The user may find it easier to type: *10 A = 5* than *10 LET A = 5*

## REMARK

It is often desirable to insert notes and messages within a user program. Information such as the name and purpose of the program, how to use it, how certain parts of the program work and expected results at various points is useful in the program for ready reference by the reader. It is ignored by the BASIC interpreter and is not executed.

Syntax: **REM** (Anything you want to type in)

Example: *10 REM THIS IS REMARK*  
*20 REM BASIC PROGRAMMING*  
*30 A = 3*  
*40 PRINT A*  
*50 END*  
  
*RUN*  
*3*

**ABS****Absolute Function, ABS (X)**

The **ABS** Function returns the absolute value of the expression. For example: **ABS (-34.67) = 34.67**

Syntax: **ABS** (expression)

Example: **10 PRINT ABS (3+4-6\*5)**

**RUN**  
**23**

**SGN****Sign Function, SGN (X)**

The sign function returns a value of +1 if X is a positive value, 0 if X is 0, and -1 if X is negative. For example:

**SGN (3.42) = 1; SGN (-42) = -1 and SGN (23-23) = 0**

Syntax: **SGN** (expression)

Example: **10 A = -12**

**30 PRINT SGN (A); SGN (A-A)**

**RUN**  
**-1 0**

**RND (0)**

The Random Number function, RND (0), produces a random number between 0 and 1.

Syntax: RND (0)

Example: *10 FOR I = 1 TO 5*  
*20 PRINT RND (0)*  
*30 NEXT I*  
*40 END*

*RUN*  
 0.53675  
 0.1463  
 0.80221  
 0.34245  
 0.36985

**RND (N)**

The Random Number function, RND (N), where N is a positive number, produces a random integer between 0 to N.

Syntax: RND (N)

Example: *10 FOR I = 1 TO 5*  
*20 PRINT RND (10)*  
*30 NEXT I*  
*40 END*

*RUN*  
 6  
 4  
 7  
 2  
 8





# CHAPTER 6

## **DATA MANIPULATION: OPERATORS**

- **Arithmetic**
- **Relational**
- **Logic**
- **Functional**



## Operators

There are 4 types of operators namely, arithmetical, relational, logical and functional.

## Mathematical Operators

The BASIC interpreter automatically performs the mathematical operations of addition, subtraction, multiplication, division and exponentiation. Formula to be evaluated are represented in a format similar to standard mathematical notation. There are five arithmetic operators used to write such formula as follows:

<u>Operator</u>	<u>Example</u>	<u>Meaning</u>
+	<b>A+B</b>	Adds <b>B</b> to <b>A</b>
-	<b>A-B</b>	Subtracts <b>B</b> from <b>A</b>
*	<b>A*B</b>	Multiplies <b>A</b> by <b>B</b>
/	<b>A/B</b>	Divides <b>A</b> by <b>B</b>
**	<b>A**B</b>	Calculates <b>A</b> to the power of <b>B</b> i.e. $A^B$

The use of parentheses in expressions will change the order of operators. Operators inside parentheses are performed first. Inside the parentheses, the normal operator precedence is observed. The arithmetic operations are performed in the following sequence, with the operation described in item 1 below having precedence.

- Any formula within parentheses is evaluated before the parenthesized quantity is used in further computations. Where parentheses are nested, as follows: **A+(B\*(D\*\*2))** the innermost parenthetical quantity is calculated first.
- In the absence of parentheses in a formula, BASIC interpreter performs operations in the following order:
  - 1st Exponentiation
  - 2nd Unary minus
  - 3rd Multiplication and division
  - 4th Addition and subtraction

Thus, for example,  $-A**B$  with a unary minus is a legal expression and is the same as  $-(A**B)$ . This implies that  $-2**2$  will be evaluated as  $-4$ . The one exception to this rule is that the term  $A**-B$  is allowed and is evaluated as  $A**(-B)$ .

3. In the absence of parenthesis in a formula involving more than one operation on the same level (see item 2 above), the operations are performed left to right, in the order that the formula is written. For example:

$A/B/C$  is evaluated as  $(A/B)/C$

$A*B/C$  is evaluated as  $(A*B)/C$

The expression  $A+B*C**D$  is evaluated in the following order:

1.  $C$  is raised to the  $D$  power.
2. The result of the first operation is multiplied by  $B$ .
3. The result of the previous operation is added to  $A$ .

The user is encouraged to use parentheses even where they are not strictly required in order to make expressions easier to read and to reduce the possibility of writing an unintended expression.

### Examples

#### Algebraic Expression

$3X-2Y$

$(X^2)Y$

$B^2-4AC$

#### BASIC Expression

$3*X-2*Y$

$X**2*Y$

$B**2-4*A*C$

## Relational Operators

Relational operators compare 2 values and make decisions regarding the program flow of a BASIC program. The result of comparison is either "1" or "0". "1" means true while "0" means false.

<u>Operator</u>	<u>Relation Tested</u>	<u>Expression</u>
=	Equality	$X = Y$
<>	Inequality	$X < > Y$
<	Less than	$X < Y$
>	Greater than	$X > Y$
<= or = <	Less than or equal to	$X < = Y, X = < Y$
>= or = >	Greater than or equal to	$X > = Y, X = > Y$

Example: *10 INPUT A, B*  
*20 IF A < B THEN PRINT "A < B"*  
*30 IF A = B THEN PRINT "A = B"*  
*40 IF A > B THEN PRINT "A > B"*  
*50 END*

*RUN*  
*? 10*  
*? 5*  
*A > B*

## Logical Operators

Logical operators are used in **IF-THEN** and such statements where condition is used to determine subsequent operations within the user program. For this discussion, A and B are relational expressions having only **TRUE** (1) and **FALSE** (0) values. Logical operators are performed after arithmetic and relational operations. The logical operators are as follows:

<u>Operator</u>	<u>Example</u>	<u>Meaning</u>
<b>NOT</b>	<b>NOT A</b>	The logical negative of A. If A is true, <b>NOT A</b> is false.
<b>AND</b>	<b>A AND B</b>	The logical product of A and B. A AND B has the value true only if A and B are both true and has the value false if either A or B is false.
<b>OR</b>	<b>A OR B</b>	The logical sum of A and B. A OR B has the value true if either A or B or both is true and has the value false only if both are false.

The following tables are called truth tables. They illustrate the results of the above logical operations with both **A** and **B** given for every possible combination of values.

A	NOT A
T	F
F	T

TRUTH TABLE FOR "NOT" FUNCTION.

A	B	A AND B
T	T	T
T	F	F
F	T	F
F	F	F

TRUTH TABLE FOR "AND" FUNCTION.

A	B	A OR B
T	T	T
T	F	T
F	T	T
F	F	F

TRUTH TABLE FOR "OR" FUNCTION.

Example: 10 INPUT A, B, C  
20 IF A > B AND B > C THEN PRINT "A > B > C"  
30 IF NOT (A > B) OR NOT (B > C)  
    THEN PRINT "A > B > C IS FALSE"  
  
40 END  
RUN  
? 10  
? 5  
? 7  
A > B > C IS FALSE



Within the course of a user's programming experience, he encounters many cases where relatively common mathematical operations are performed. The results of these common operations can often be found in volumes of mathematical tables. e.g. sine, cosine, square root, log, etc. Since it is this sort of operation that computers perform with speed and accuracy, such operations are built into the BASIC interpreter. The user does not require to look up tables to obtain the value of the sine of 25 degrees or the natural log of 150. When such values are to be used in an expression, intrinsic functions, such as:

**SIN (25\*3.14/180)**  
**LOG (150)**

are substituted.

The various mathematical functions available in our BASIC interpreter are detailed in the following table.

Function Code	Meaning
<b>ABS (X)</b>	Gives the absolute value of X.
<b>SGN (X)</b>	Gives the sign function of X, a value of 1 preceded by the sign of X, $\text{SGN}(\emptyset) = \emptyset$
<b>INT (X)</b>	Gives the greatest integer which is less than or equal to X, $\text{INT}(-0.5) = -1$ .
<b>COS (X)</b>	Gives the cosine of X (in radians).
<b>SIN (X)</b>	Gives the sine of X (in radians).
<b>TAN (X)</b>	Gives the tangent of X (in radians).
<b>SQR (X)</b>	Gives the square root of X.
<b>EXP (X)</b>	Gives the value $e^x$ , where $e = 2.71828$ .
<b>LOG (X)</b>	Gives the natural logarithm of X, $\log_e X$ .
<b>RND (0)</b>	Gives a random number between 0 and 1.
<b>RND (N)</b>	Gives a random integer from 0 to N-1
<i>In sin (X), cos (X) and tan (X), the range of X should be: <math>-1000 &lt; X &lt; 1000</math>.</i>	

Example: 10 REM TO PRINT A SINE AND COSINE TABLE  
 20 PRINT "SIN (X)", "COS (X)"  
 30 FOR X = 0 TO 2 STEP 0.5  
 40 PRINT SIN (X), COS (X)  
 50 NEXT

RUN

SIN (X)	COS (X)
0	1
0.47942	0.87758
0.84147	0.5403
0.9975	0.07073
0.9093	-0.41614

# CHAPTER 7

## STRING FUNCTIONS

- Character Strings
  - String Constants
  - String Variables
- String Manipulation
- String Functions:
  - LEFT\$
  - RIGHT\$
  - MID\$
  - CHR\$
  - STR\$
  - LEN
  - VAL
  - ASC
- String Comparison



## CHARACTER STRINGS

The previous chapters describe the manipulation of numerical information. However, the BASIC interpreter also processes information in the form of character strings. A string, in this context, is a sequence of characters treated as a unit. A string can be composed of any combination of ASCII characters.

Example: *"ABC"*  
*"BASIC"*

### String Constants

#### String Constants

Just as numbers can be used as constants or referenced by variable names, BASIC interpreter also processes character string constants. Character string constants are delimited by double quotes. For example:

*10 LET A\$ = "XYZ123"*

This means that the value of **A\$** is the character string **XYZ123**.

### String Variables

#### String Variables

Variable names can be assigned to strings. Any letter of the alphabet followed by a dollar sign (\$) character is a legal name for a string variable such as **A\$**. The maximum value of a string variable is 32 characters. During execution, the length of any string expression cannot exceed 32 characters.

## String Manipulation

Rameses BASIC allows programmers to manipulate characters. The only operator for string expressions and variables is '+'. '+' means concatenation (linking).

Example 1: *10 LET A\$ = "XYZ123"*  
*20 P\$ = A\$ + "PQR"*  
*30 PRINT P\$*  
*40 END*

*RUN*  
*XYZ123 PQR*

Example 2: *10 A\$ = " "*  
*20 FOR I = 1 TO 5*  
*30 A\$ = A\$ + "#"*  
*40 PRINT A\$*  
*50 NEXT I*

*RUN*  
*#*  
*# #*  
*# # #*  
*# # # #*  
*# # # # #*

## STRING FUNCTIONS

Besides intrinsic mathematical functions (e.g. **SIN**, **LOG**), our **BASIC** interpreter provides various functions for character string manipulation. These functions allow the program to perform arithmetic operations with numeric strings, concatenate two strings, access part of a string, determine the number of characters in a string, generate a character string corresponding to a given number or vice versa, search for a substring within a larger string and perform other useful operations.

### LEFT\$

Syntax: **LEFT\$** (String expression, numeric expression)  
e.g. **LEFT\$ (A\$, N)**

It returns a substring from the first character (the leftmost characters of the string **A\$**) through the **Nth** character.

Example: *10 A\$ = "ABC"*  
*20 C\$ = LEFT\$ (A\$+"XYZ", 3+1)*  
*30 PRINT C\$*  
  
*RUN*  
*ABCX*

### RIGHT\$

Syntax: **RIGHT\$** (String expression, numeric expression)  
e.g. **RIGHT\$ (A\$, N)**

It returns a substring of the string **A\$** consisting of the last **N** characters.

Example: *10 A\$ = RIGHT\$ ("BASIC", 3)*  
*20 PRINT A\$*  
  
*RUN*  
*SIC*

**MID\$**

Syntax: **MID\$** (String expression, numeric expression, numeric expression)  
 e.g. **MID\$(A\$, M, N)**

It returns a substring of the string **A\$** starting from the Mth character with a length of **N**.

Example: *10 A\$= "ABCDEFGH"*  
*20 C\$= MID\$ (A\$, 3, 4)*  
*30 PRINT C\$*  
  
*RUN*  
*CDEF*

**CHR\$**

Syntax: **CHR\$** (Numeric expression)  
 e.g. **CHR\$ (N)**

It generates a 1-character string having the ASCII value of **N**

(see Appendix D). For example **CHR\$(65)** is equivalent to "A".  
 Only one character can be generated.

Example: *10 FOR I = 65 TO 70*  
*20 PRINT CHR\$ (I)*  
*30 NEXT I*  
  
*RUN*  
*A*  
*B*  
*C*  
*D*  
*E*  
*F*



**STR\$**

Syntax: **STR\$** (Numeric expression)

e.g. **STR\$** gives a string expression of a numeric argument.

Example 1: **10 A\$ = STR\$ (3\*2+1)**

**20 C\$ = "00"+A\$**

**30 PRINT A\$, C\$**

**RUN**

**7 007**

Example 2: **10 A\$ = STR\$ (0.125 + 0.5)**

**20 C\$ = A\$+"K"**

**30 PRINT C\$**

**RUN**

**0.625K**

**LEN**

Syntax: **LEN** (String expression)

e.g. **LEN (A\$)**

It returns the number of characters, including blank spaces in the string A\$.

Example 1: **10 A\$ = "ABCDEF"**

**20 X = LEN (A\$)**

**30 PRINT X**

**RUN**

**6**

Example 2: **10 A\$ = "XY"**

**20 C\$ = "123"**

**30 X = LEN (A\$+C\$)+6**

**40 PRINT X**

**RUN**

**11**

**VAL**

Syntax: **VAL** (String expression)

e.g. **VAL(A\$)**

It returns the numeric value of the string expression.

Example: *10 A\$ = "123 + 1"*  
*20 X = VAL (A\$+"-100")*  
*30 PRINT X*  
  
*RUN*  
*24*

**ASC**

Syntax: **ASC** (String expression)

e.g. **ASC (A\$)**

It returns the ASCII decimal value of the first character in A\$. For example the ASCII decimal value of "X" is 88. If B\$ = "XAB", then *ASC(B\$) = 88*

Example: *10 X = ASC ("AXY")*  
*20 PRINT X*  
  
*RUN*  
*65*

## String Comparison

The relational operators can also be applied to string expressions as they can to numeric expressions. e.g. **A\$ = "123A", X**

The comparison is done by taking the corresponding numerical value of the string characters and then comparing. According to Appendix D we know the value for 'A' is 65, 'B' is 66, 'C' is 67 and 'D' is 68, etc. We can then have the following examples working.

```
Example: 10 A$ = "AA"
          20 B$ = "BA"
          30 IF A$ < B$ THEN PRINT 20

          RUN
          20
```

- \* Since the numerical value of character A is less than the numerical value of character B (65 is less than 66, therefore A\$ is less than B\$.)

```
Example: 10 A$ = "ABC"
          20 B$ = "ABD"
          30 IF B$ > A$ THEN PRINT 20

          RUN
          20
```

- \* The first two characters of each string are equal. Then the computer will search for the third character and do the comparison, i.e. Compare "D" and "C". We know the numerical value of "D" is 68 and "C" is 67. We can therefore conclude that B\$ is greater than A\$.

```
Examples: 10 A$ = CHR$(30*2+6)
           20 IF A$ >= "B" THEN PRINT 1
           30 IF A$ >= "9" THEN PRINT 2

           RUN
           1
           2
```

Examples: 10 W\$ = "BASIC"  
 20 X\$ = "BA"  
 30 IF LEFT\$ (W\$, 3) > X\$ THEN 60  
 40 STOP  
 60 PRINT X\$  
  
 RUN  
 BA

Examples: 10 W\$ = "APPLE"  
 20 X\$ = "ORANGE"  
 30 IF W\$ <> X\$ THEN 60  
 40 STOP  
 60 PRINT X\$  
  
 RUN  
 ORANGE

Examples: 10 REM. ...PRINT ASCII CHARACTERS  
 20 FOR I = 1 TO 128  
 30 PRINT CHR\$ (I)  
 40 NEXT I

RUN

.

.

.

.

.

A

B

C

.

.

.

.

.

\* 128 ASCII characters will be printed. Only printable characters are shown on the screen.

# **CHAPTER 8**

## **SYSTEM CONTROL**

- STOP**
- END**
- CONT**
- CNTL/C**
- RESET**



<b>System Control</b>
-----------------------

In order to facilitate you to control your program execution and debugging, several control commands/statements are introduced. They are the **STOP & END** statements and **CONT & CNTL/C** commands.

You can also stop your looping program by pressing the "RESET" button.

<b>STOP</b>
-------------

The **STOP** statement terminates program execution and returns control to command mode. It can occur several times throughout a single program with conditional jumps determining the actual end of the program.

Syntax: line number **STOP**

After execution, it causes

**STOP AT** line-number

to be printed.

A **CONTINUE** command entered at this point resumes execution at the statement following **STOP** (see latter section). The **STOP** statement in fact may facilitate you to debug programs since you can add break points at various locations of your program. Hence, you can trace the program execution and examine the intermediate results of the variables in the Immediate — execution mode.

```
Example: 10 PRINT "TEST STOP"
          20 PRINT 123
          30 STOP
          40 END

          RUN
          TEST STOP
          123
          STOP AT 30
```

**END**

The **END** statement is used to terminate program execution. It must be the last statement in a BASIC program.

Syntax: line number **END**.

The line number of the **END** statement should normally be the largest line number in the program. A program will execute without an **END** statement. However, it is a good practice to include an **END** statement since it is generally accepted by all BASIC interpreters.

Unlike **STOP**, a program which has stopped after **END** cannot be continued by the **CONT** command. (latter section)

**CONT**

The user can place **STOP** statements liberally throughout the program. Each **STOP** statement causes the program to halt and print the line number at which the **STOP** occurred. The user can then examine various data values, change them in immediate-execution mode, and give the **CONT** command to continue program execution.

**CNTL/C**

It is already known that **STOP** can halt a program at pre-determined points. On the other hand, the user can also halt program execution at any time by typing **CNTL** and **C** together. Immediate mode can then be used to examine and/or change data values. Typing the **CONT** command resumes program execution.

When a program is interrupted by typing the **CNTL/C** combination, it causes the following to be printed:

**STOP AT**

line-number, statement.

**CNTL/C** Combination:

**CNTL** **C**

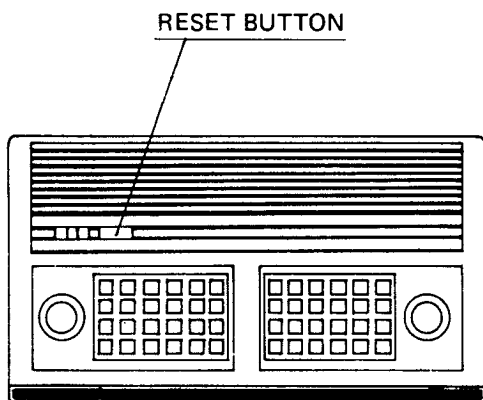
\*Press the above two keys simultaneously (or hold down **CNTL** and press **C**).



**RESET**

On the main console, there is a **RESET** button next to the Power ON-OFF switch. This button is used to reset the computer system. When this button is pressed, the computer will stop running your program, all characters are reset to normal form.

So if your program is in a dead loop because of some mistakes in programming and you cannot exit the loop by **CTL-C**, you can press the button. The BASIC program that you have written will not be affected.



**Fig. 8.1**



# CHAPTER 9

## LOOPING & CONDITION BRANCHING

- IF ... THEN
- FOR ... NEXT
- GOSUB/RETURN
- GO TO



In this chapter, we will discuss statements which allow the programmer to direct the computer to execute a program statement based on the results or conditions of arithmetic and logical operations; the programmer can also direct the computer to execute any program line unconditionally.

**IF . . . THEN**

Program flow is directed under conditions established in a logical expression.

In writing programs, we need a statement that can provide a conditional branch to another statement. The **IF . . . THEN** statement does just that.

Syntax: **IF** logical expression **THEN** BASIC statement or  
Line number.

Example: *10 S = 0*  
*20 N = 0*  
*30 N = N + 1*  
*40 S = S + N*  
*50 IF N <> 10 THEN 30*  
*60 PRINTS*  
*70 END*  
  
*RUN*  
*55*

**FOR . . . NEXT**

One advantage of computers is their ability to perform repetitive tasks. The loop performs a series of BASIC statements for a given number of times. Suppose we want a table of square roots, for the integers from 1 to 10. We need not write the same statement 10 times for the same function. Instead, we use looping to do it ten times.

```
Example: 10 FOR N= 1 TO 5
          20 PRINT N, SQR (N)
          30 NEXT N

          RUN

          1      1
          2      1.41412
          3      1.73205
          4      2
          5      2.23606
```

The above example repeats the line 20 five times with increment of 1 each time. However, we can also indicate the step size for values other than 1. If we want to find only the square root of the odd number from 1 to 10, we must modify the above example.

```
Example: 10 FOR N = 1 TO 10 STEP 2
          20 PRINT "N=", SQR (N)
          30 NEXT N

          RUN
```

**Syntax:** **FOR** VARIABLE = Exp 1 **TO** Exp 2 **STEP** Exp 3. **STEP** is the increment value, if it is omitted the increment value is 1. Exp 1, Exp 2 and Exp 3 are simple arithmetic expressions Exp 1 is the starting value and Exp 2 is the upper limit.

All **FOR** loops must be closed with a **NEXT** statement.

**Syntax:** **NEXT** (Variable)

The variable name of the **NEXT** statement must be the same as the variable name used in the **FOR** statement. See examples above.

**GOSUB/RETURN**

Frequently, identical or nearly identical operations must be performed repeatedly within a program. It is impractical to repeat the same set of instructions again and again in the program because this will waste a lot of computer memory and programmer time. When the same operation must be performed at several different locations within a program, it is wise to make it into a subroutine. Whenever we want to execute this subroutine, we use the **GOSUB** statement. All subroutines must be ended with a **RETURN** statement, which directs the computer back to the statement immediately following the **GOSUB** in order to continue executing the rest of the program.

Syntax: **GOSUB** (1st Line number of the subroutine)

```
.  
.   
.   
(1st Line number of the subroutine)   
.   
.   
.   
RETURN
```

} SUBROUTINE

**GO TO**

The **GO TO** statement transfers program execution immediately and unconditionally to a specified program line. Usually the specified line is not the next sequential line in the program.

Syntax: **GO TO** < line number >

The line number to which the program jumps can be either greater than or less than the current line number. It is thus possible to jump forward or backward within a program.

Example: *10 N = 1*  
*20 S = 0*  
*30 S = S + N*  
*40 N = N + 1*  
*50 PRINT N, S*  
*60 GO TO 30*

*RUN*

<i>1</i>	<i>1</i>
<i>2</i>	<i>2</i>
<i>3</i>	<i>6</i>
<i>4</i>	<i>10</i>
<i>.</i>	<i>.</i>
<i>.</i>	<i>.</i>
<i>.</i>	<i>.</i>

\*N.B. The program will continue executing without ending. Use control-C to stop it.



Example: 10 FOR I = 1 TO 5  
 20 GOSUB 60  
 30 PRINT I, S  
 40 NEXT I  
 50 END  
 60 S = 0  
 70 FOR J = 1 to I  
 80 S = S+J  
 90 NEXT J  
 100 RETURN

RUN

1	1
2	3
3	6
4	10
5	15

Example: 10 A = 30  
 15 PRINT A  
 20 GO TO A  
 25 PRINT A\*A  
 30 END

RUN

30

You can see that line 25 is skipped.



# CHAPTER 10

## ARRAYS — DIM



## ARRAYS

An array is a set of variables, or elements, all with the same name, and are only distinguished by a number written in brackets after the name (i.e. the subscript). This is the method to set up a table of variables in the computer. **A (I)** is the I'th element in the Array **A**, where I is integer

Syntax:

Example: *10 REM THE GAS BILLS OF HALF YEAR*  
*20 DIM A(6)*  
*30 REM ASK FOR 6 BILLS*  
*40 FOR I = 1 to 6*  
*50 PRINT "THE BILL AMOUNT =";*  
*60 INPUT A (I)*  
*70 S = S+A (I)*  
*80 NEXT I*  
*90 PRINT "THE TOTAL AMOUNT ="; S*

**DIM**ension reserves storage locations for arrays and matrices. The dimension is either 1 or 2.

**A (6)** is a one dimensional array with 6 elements, whereas **A (6, 6)** is a two dimensional array or matrix with  $6 \times 6 = 36$  elements.

The two-dimensional array requires two subscripts to specify each of the elements — just like the column and row numbers in a normal matrix.

For example, to set up a two-dimensional array **A** with dimensions 3 and 5, you use a **DIM** statement

**DIM A(3, 5)**

This then gives you  $3 \times 5 = 15$  subscripted variables

**A(1, 1), A(1, 2) ... A(1, 5)**

**A(2, 1), A(2, 2) ... A(2, 5)**

**A(3, 1), A(3, 2) ... A(3, 5)**

Example: 10 DIM A (3, 3)  
20 FOR I = 1 to 3  
30 FOR J = 1 to 3  
40 A (I, J) = I+J  
50 PRINT A (I, J);  
60 NEXT J  
70 PRINT  
80 NEXT I

RUN

2	3	4
3	4	5
4	5	6

# CHAPTER 11

## INPUT/OUTPUT COMMANDS

- INPUT
- PRINT
- TAB
- READ/DATA
- RESTORE





**INPUT****INPUT**

INPUT accepts data from the keyboard during program execution. The "?" is displayed as a prompt.

Syntax: **INPUT** variable 1, variable 2, . . . . .

Examples: *10 INPUT A, B*  
*20 PRINT A, B, A+B*  
*30 END*

*RUN*

? 3 }  
 ? 4 } any number you type in.

3    4    7        (computer output)

PRINT

**PRINT**

**PRINT** displays the value of a variable, or the value of an expression on screen.

Syntax: **PRINT** item, item, item . . . .

```
Example: 10 PRINT "BASIC PROGRAM"
          20 LET A = 3
          30 PRINT A, A+A, A * A
          40 END

          RUN
          BASIC PROGRAM      (computer output)
          3    6    9
```

In the **PRINT** statement, items can be separated by ";" instead of ",". Then no space between items will be printed.

Syntax: **PRINT** item ; item ; item ; item

```
Example: 10 PRINT 1; 2; 1+2
          20 PRINT "A="; 3

          RUN
          1 2 3      (computer output)
          A = 3
```

The **PRINT** statement terminated by a semi-colon does not generate a line feed.

Syntax: **PRINT** item ;

```
Example: 10 FOR A = 1 TO 10
          20 PRINT A;
          30 NEXT
          40 END

          RUN
          1 2 3 4 5 6 7 8 9 10      (computer output)
```

**LPRINT****LPRINT**

**LPRINT** is just like **PRINT** except that it prints on the printer instead of on the television screen.

Syntax: **LPRINT** item, item

Example: *10 LPRINT "THIS PROGRAM"*  
*20 LPRINT "PRINT OUT THE CHARACTER SET"*  
*30 FOR N = 32 TO 96*  
*40 LPRINT CHR\$(N);*  
*50 NEXT*

*RUN*

(A set of characters is printed on the printer).

\*Note that when **LPRINT** is used, the printer and I/O interface must be ready. For details please refer to the operating manual of the parallel and Serial I/O Interface.

**TAB**

**TAB** is used in a **PRINT** statement to move the cursor to the right before printing. The number of positions moved is indicated in the ( ).

Syntax: **TAB** (variable)

**TAB** (n) puts the cursor to the nth column on the screen. n can be any integer from 0 to 64.

Example: *PRINT TAB(5); "A"; TAB(5); "A"*

A        A

**READ**

**READ** and **DATA** statements are coordinated to furnish a fixed list of data values to the user program. A **READ** statement initializes variables by getting value of data items from **DATA** statements.

Syntax: **READ** (Variable 1, Variable 2 . . . .)

Example: *10 DATA 1, 3, 5, 7*  
*20 READ X, Y*  
*30 READ A*  
*40 READ B*  
*50 PRINT X+Y, A+B*  
*60 END*

*RUN*  
 4     12

## DATA

## DATA

**DATA** supplies data items to a **READ** statement. The **READ** statement reads items according to the order of the line number of **DATA** statement. All **DATA** statements must be at the beginning of a BASIC program or before the **READ** statements.

Syntax: **DATA** constant 1, constant 2, constant 3 . . . . .

Example: *10 DATA 8, 1, 6*  
*20 DATA 3, 5, 7, 4, 9, 2*  
*30 FOR I = 1 TO 3*  
*40 READ A, B, C*  
*50 PRINT A, B, C*  
*60 NEXT I*  
*70 END*

*RUN*

8	1	6
3	5	7
4	9	2

Example: *10 REM FIND MAXIMUM*  
*20 REM AND AVERAGE*  
*30 DATA 0.125, 3, 0.6, 7*  
*40 DATA 23, 9.3, 25.2, 8*  
*50 M = -99*  
*60 S = 0*  
*70 FOR I = 1 TO 8*  
*80 READ N*  
*90 S = S+N*  
*100 IF N > M THEN M = N*  
*110 NEXT I*  
*120 A = S/8*  
*130 PRINT M, A*  
*RUN 25.2 9.5281*

**RESTORE**

If it should become necessary to use the same data more than once in a program, the **RESTORE** statement makes it possible to move the **DATA** pointer back to the first **DATA** value.

Syntax: **RESTORE**

Example: *10 DATA 1, 3, 5, 7, 9*  
*20 READ A, B, C*  
*30 RESTORE*  
*40 READ X*  
*50 PRINT A, C*  
*60 PRINT X*  
*70 END*  
  
*RUN*  
*1     5*  
*1*

# CHAPTER 12

## TAPE STORAGE

- CSAVE
- CLOAD
- CRUN





## Cassette Storage Module

You can further expand your Rameses Computer System by using the **Cassette Storage Module**. The Creativision BASIC allows you to load and save programs which you enter in the computer. By recording a program on a tape, you can save it as a permanent record. Later you can load the program from the cassette tape into the computer's memory if you want to use that program again.

The Cassette Storage Module is connected to the computer system by a cassette interface cable, as shown in fig 12.1.

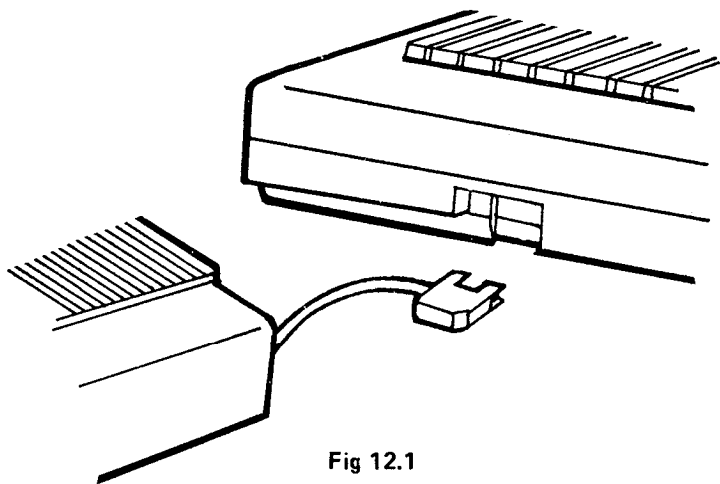


Fig 12.1

**Syntax: CSAVE****Procedure:**

1. Connect the cassette storage module with the computer system by the cable.
2. Put a good quality tape cassette in place.
3. Press the **"PLAY"** and **"RECORD"** buttons on the module to select the **"RECORD"** mode. Notice that the cassette will not start to move. The computer controls the cassette motor power supply. Therefore the tape does not start to move until the computer tells it to do so.
4. Record the reading of the digital tape counter in the module. This is for you to easily locate the correct tape position when you want to load your program back into the computer.
5. Key in the **"CSAVE"** command (followed by RET'N). As soon as you do, your program will begin recording on the tape, and the screen will list this program.
6. When the program has been recorded, the prompt sign will appear on the screen, Press the **"stop"** button on the module.

**CLOAD****Syntax: CLOAD****Procedure:**

1. Connect the cassette storage module with the computer system by the cable.
2. Put the tape cassette in place and rewind it to the correct position.
3. Press the play button only to select the **"LOAD"** option of the module.
4. Key in the **"CLOAD"** command. As soon as you do so, the recorded program will begin to load into the computer system, and the screen will list this program.
5. After the program has been loaded, the prompt sign will appear on the screen. Press the **"stop"** button on the module.

N.B. If the program in tape is from the Rameses Tape Library, there will be speech and sound effects coming out from the TV set during loading of the program.

**CRUN****Syntax: CRUN**

**CRUN** is the same as **CLOAD+RUN**. The computer will load the program and then **RUN** it automatically.

The procedure is the same as **CLOAD** except you key in **"CRUN"** instead of **"CLOAD"**.



# CHAPTER 13

## GRAPHICS & SOUND FUNCTIONS

- CLS
- COLOR
- CHAR
- PLOT
- SOUND
- JOY



## Graphic Commands

The graphic commands feature a 32-column by 24-row screen display. The 28 print positions normally used in Rameses BASIC correspond to columns 3 through 30. Because some display screens may not show the two leftmost and two rightmost characters, your graphics may be more satisfactory if you use columns 3 through 30 and ignore columns 1 and 2 on the left and 31 and 32 on the right.

There are 4 Graphic commands, they are:

1. **CLS** — clear screen.
2. **COLOR** — Define color.
3. **CHAR** — Define characters.
4. **PLOT** — Locate and display characters.

**CLS**

### CLS

The Clear command is used to clear the entire screen. When CLS statement is executed, the space character is placed in all positions on the screen.

*e.g. 10 CLS*

*20 PRINT "CLEAR SCREEN"*

When this program is run, the screen is cleared and then "CLEAR SCREEN" appears on the screen.

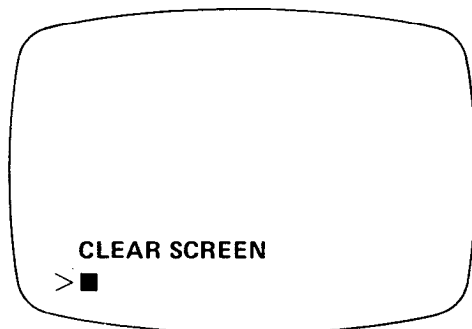


Fig. 13.1

## COLOR

## COLOR

The "COLOR" command provides a capability for you to specify screen character colors.

Syntax: **COLOR** Character-set-number, foreground-color-code, background-color-code.

Example: *COLOR 2, 10, 14.*

Each character-display on your computer screen has two colors. The color that makes up the character itself is called the foreground color. The color that occupies the rest of the character position on the screen is called the background color. Sixteen colors are available on the Rameses BASIC. The color codes are given below:

Color Code	Color
1	Transparent
2	Black
3	Medium Green
4	Light Green
5	Dark Blue
6	Light Blue
7	Dark Red
8	Cyan
9	Medium Red
10	Light Red
11	Dark Yellow
12	Light Yellow
13	Dark Green
14	Magenta
15	Grey
16	White

To use the COLOR commands, you must also specify to which character set the character to be printed belongs. The list of ASCII



character codes for standard characters is given in Appendix D. The character-set-numbers are given below.

CHARACTER SET NUMBER	ASCII CODE	CHARACTER SET NUMBER	ASCII CODE
1	0-7	17	128-135
2	8-15	18	136-143
3	16-23	19	144-151
4	24-31	20	152-159
5	32-39	21	160-167
6	40-47	22	168-175
7	48-55	23	176-183
8	56-63	24	184-191
9	64-71	25	192-199
10	72-79	26	200-207
11	80-87	27	208-215
12	88-95	28	216-223
13	96-103	29	224-231
14	104-111	30	232-239
15	112-119	31	240-247
16	120-127	32	248-255

Note that the screen is filled with space characters until you place other characters in some of these positions. If you use character set 5 in the **COLOR** statement, all space characters on the screen are changed to background-color specified since the space character is contained in set 5.

This is demonstrated by the program:

```
Example: > 100 REM COLOR DEMOSTRATION
          > 200 CLS
          > 300 COLOR 5, 6, 6
          > 400 GO TO 400

          > RUN
          (the SCREEN color is changed from light green to light blue.)
```

\*Press **CTL-C** to stop the program.

CHAR

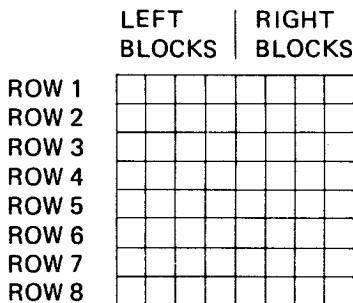
**CHAR**

The "CHAR" command allows you to define your own special graphics characters. You can redefine a new set of character patterns.

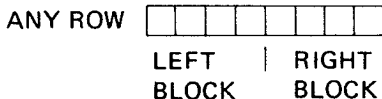
Syntax: **CHAR** char-code, pattern-identifier

The char-code specifies the code of the character you wish to define and must be a numeric expression with a value between 0 and 255.

The pattern-identifier is a 16-character expression which specifies the pattern of the character you want to use in your program. This expression is a coded representation of the 64 dots which makes a character pattern on the screen. These 64 dots comprise an 8 x 8 grid.

**Fig. 13.2**

Each row is partitioned into two blocks of four dots each.

**Fig. 13.3**

Each character in the string expression describes the pattern of dots in one block of a row. The rows are described from left to right and from top to bottom. That is, the first two characters in the string describe the pattern for row one of the dot-grid, the next two describe row two, and so on.

Characters are created by turning some dots "On" and leaving others "Off". To create a new character, you must tell the computer what dots to turn on or leave off in each of the 16 blocks that construct the character.

If the string expression is less than 16 characters, the computer will assume that the remaining characters are zero. If string is longer than 16 characters, the computer will ignore the excess.

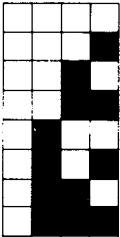
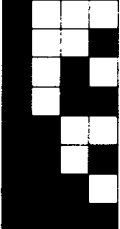



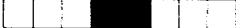

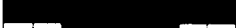

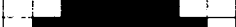

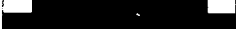







<i>Blocks</i>	<i>(0 = Off. 1 = On)</i>	<i>Code</i>
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
	0111	7
	1000	8
	1001	9
	1010	A
	1011	B
	1100	C
	1101	D
	1110	E
	1111	F

Fig. 13.4

Example: To describe the dot pattern pictured below:

**CHAR 32, 18 18 FF 3C 7E FF 14 36**

	LEFT BLOCKS	RIGHT BLOCKS	BLOCK CODES
ROW 1			18
ROW 2			18
ROW 3			FF
ROW 4			3C
ROW 5			7E
ROW 6			FF
ROW 7			14
ROW 8			36

**Fig. 13.5**

### CHARACTER PATTERN OF CHARACTER CODE – 32.

Remember that CHAR command only defines a character. To display the character on the screen you will need to use the **PLOT** command. When CHAR is performed, any character already on the screen with the same char-code is changed to the new character.

Example: `> 10 CLS`  
`> 15 COLOR 5, 4, 6`  
`> 20 CHAR 32, 18 18 FF 3C 7E FF 14 36`  
`> 30 GO TO 30`  
`> RUN`

(The screen will be filled with the above pattern because the screen is filled with space characters after CLS and the space character has a code number of 32 which has been redefined.)

Note that the characters 32-95 is already defined in the computer. But you can redefine the characters yourself.

As the example above, we redefined the space by a special character.

If the reset button is pressed, all redefined characters and screen color will be reset.

## PLOT

The "PLOT" statement allows you to place a character anywhere on the screen.

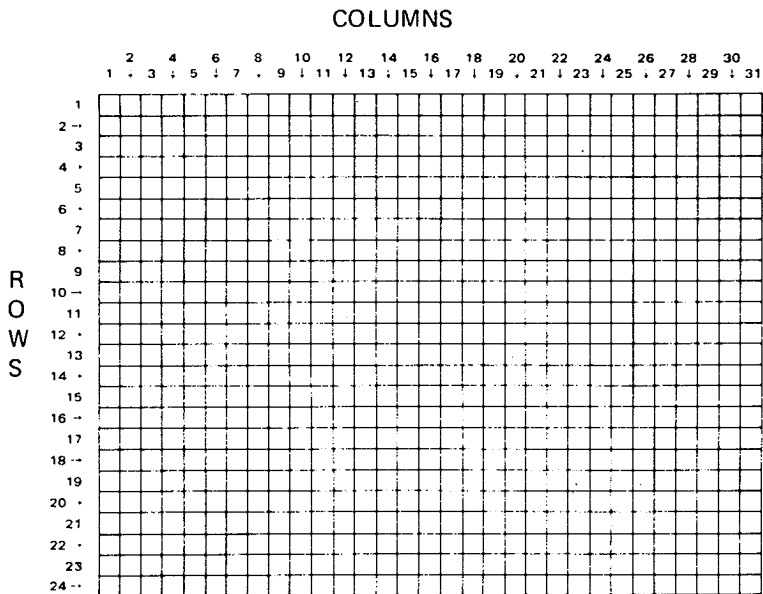


Fig. 13.6

A value of 1 for column-number indicates the left most column of the screen.

A value of 1 for row-number indicates the top row of the screen. The screen can be thought of as a "grid" as shown.

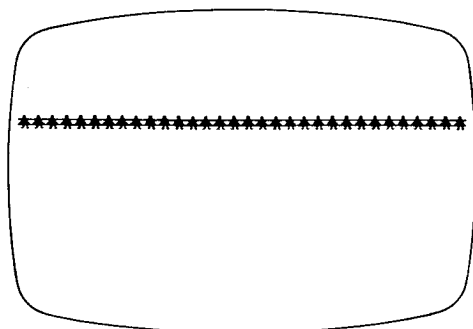
Because columns 1, 2, 31 and 32 may not show on some TVscreens, you may want to use only column-numbers 3 through 30.

Syntax: **PLOT** column number, row number, character code

Example: *PLOT 15, 10, 65*

This makes the screen show a character "A" on the position specified by column-number = 15, row-number = 10.

Example: *10 CLS*  
*20 COLOR 5, 4, 6*  
*30 CHAR 33, 18 18 FF 3C 7E FF 14 36*  
*40 FOR X = 1 TO 32*  
*50 PLOT X, 10, 33*  
*60 NEXT*  
*70 GOTO 70*  
  
*RUN*



**Fig. 13.7**

The pattern forms a line across the screen.

Example: 05 CLS  
10 FOR A = 1 TO 16  
20 COLOR A+16, A, A  
30 NEXT  
40 FOR X = 1 TO 32  
50 FOR Y = 1 TO 20  
60 PLOT X, Y, X\*4+128  
70 NEXT  
80 NEXT  
90 GOTO 90  
RUN

The program shows a set of colour bars on the TV screen.

## SOUND

**Sound:**

The **SOUND** statement tells the computer to produce tones of different frequencies. There are 3 music channels and you can program the 3 channels independently.

Syntax: **SOUND** frequency code; duration code, frequency code; duration code, . . . . .

Example: *10 SOUND 4;5, 6;7, 7;7*  
(plays a bell sound)









The frequency code and duration code are numeric expressions. If the evaluation of any of the numeric expressions results in a non-integer value, the result is rounded to obtain an integer.

**Frequency code:**

1	Rest	17	D <sup>#</sup> , E <sup>b</sup>
2	C	18	E
3	C <sup>#</sup> , D <sup>b</sup>	19	F
4	D	20	F <sup>#</sup> , G <sup>b</sup>
5	D <sup>#</sup> , E <sup>b</sup>	21	G
6	E	22	G <sup>#</sup> , A <sup>b</sup>
7	F	23	A (above middle C)
8	F <sup>#</sup> , G <sup>b</sup>	24	A <sup>#</sup> , B <sup>b</sup>
9	G	25	B
10	G <sup>#</sup> , A <sup>b</sup>	26	C (high C)
11	A (below middle C)	27	C <sup>#</sup> , D <sup>b</sup>
12	A <sup>#</sup> , B <sup>b</sup>	28	D
13	B	29	D <sup>#</sup> , E <sup>b</sup>
14	C (middle C)	30	E
15	C <sup>#</sup> , D <sup>b</sup>	31	F
16	D	32	Rest



Duration code:

0	$\frac{1}{4}$	
1	$\frac{1}{2}$	
2	$\frac{3}{4}$	
3	1	
4	$1\frac{1}{2}$	
5	2	
6	3	
7	4	

In a SOUND statement, the first pair of data will specify the first channel, the second pair will specify the second channel and the third pair will specify the third channel.

Example: 10 REM SONG  
 20 SOUND 26;3, 21;3, 14;3  
 30 SOUND 30;3, 25;3, 18;3  
 40 SOUND 28;3, 23;3, 16;3  
 50 SOUND 21;5, 16;5, 9;5  
 60 SOUND 26;3, 21;3, 14;3  
 70 SOUND 28;3, 23;3, 16;3  
 80 SOUND 30;3, 25;3, 18;3  
 90 SOUND 26;7, 21;7, 14;7  
 100 GOTO 20

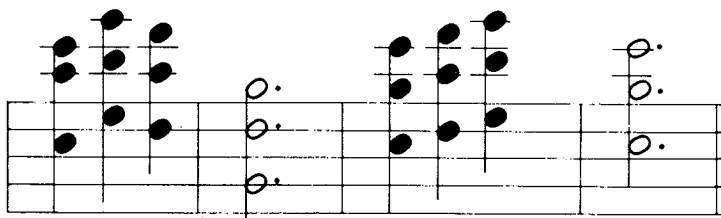


Fig. 13.8

> RUN

The computer will play the melody as shown in Fig. 13.1

## JOY

The joy-stick function, **JOY**, allows the user to input information to the computer based on the position of the joy-stick. This function is useful for the user in writing game program.

The key board of Creativision computer is divided into two parts, ie, left and right. Each part has a joy-stick and two fire buttons. The joy-stick function is use to input these conditions.

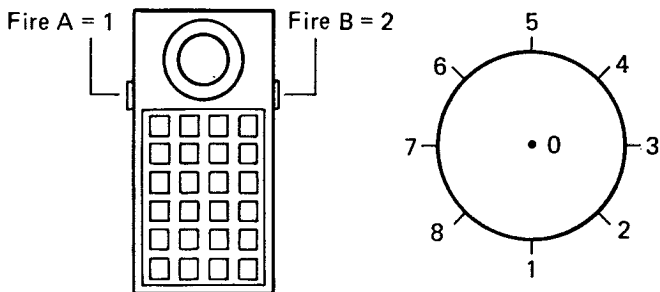


Fig 13.8 The corresponding value of the joy-stick position and fire button.

Syntax: **JOY (N)** N = 1 for left hand side joy-stick position.  
 N = 2 for right hand side joy-stick position.  
 N = 3 for left hand side joy-stick fire button.  
 N = 4 for right hand side joy-stick fire button.

The function give the value of the position of the joy-sticks.

Example: *10 PRINT JOY (1)*

*RUN*  
 3

This example prints out the position of the left hand side joy-stick. In this example, the left hand side joy-stick is at right direction.

Example: 05    CLS  
          10    LET A=JOY (1)  
          20    IF A=3 THEN GOTO 100  
          30    IF A=7 THEN GOTO 200  
          40    GOTO 10  
          100   PLOT 4, 10, 32  
          110   PLOT 28, 10, 65  
          120   GOTO 10  
          200   PLOT 28, 10, 32  
          210   PLOT 4, 10, 65  
          220   GOTO 10  
  
          RUN

This program will display the character "A" on the left hand side or right hand side of the TV screen according to the position of the joystick.

# CHAPTER 14

## SYSTEM MEMORY ACCESS

- PEEK
- POKE



**PEEK**

**PEEK** is a function that returns the value (decimal integer in the range 0 to 255) read from the specified location in the memory.

Syntax: **PEEK** (address).

For example you can see what value is at a specified location.

Example: *10 PRINT "ADDRESS", "CONTENT"*  
*20 FOR K = 0 TO 20*  
*30 PRINT K, PEEK (K)*  
*40 NEXT K*

**POKE**

As discussed above, with the **PEEK** function, we can read the memory content from any location. The function of **POKE** is just the complement of **PEEK**. Hence, you can write a data into a memory location.

SYNTAX: **POKE** address, data

Example: *POKE 1000, 48*

This ~~makes~~ the content at the address location ~~1000~~ with the value 48. If you type

*PRINT PEEK 300*

you get your number 48 back. (Try poking in other values).

Note: **POKE**ing values into some memory locations may cause the computer to "crash". To remedy this, press **RESET**.





# CHAPTER 15

## RAMESES SYSTEM EXPANSION



There are some options for expanding your Rameses system. The following are the introductions. For details please refer to the corresponding manual.

1. The Cassette Storage Module

Together with the Rameses BASIC you can save BASIC programs on audio cassette tapes and read it back again.

The operation of the cassette can be program controlled. The Baud rate of data transfer is 600.

2. The Parallel and Serial I/O interface

The Parallel and Serial I/O Interface module provides a wide range of input and output facilities. The module has one parallel port which is for your printer. This port is of standard "Centronics Bus" so that it can connect to every Centronics Bus printer, eg EPSON MX80, SEIKOSHA GP-80, CENTRONICS 779, etc.

As for the other two ports, one is for Floppy Disk Drive and one is for Telephone Interface.

3. The Memory Expansion Module

The memory of the Rameses Computer can be expanded by adding the "Memory Expansion Modules" to the system. The Memory Expansion Module has the option of 16K and 32K.

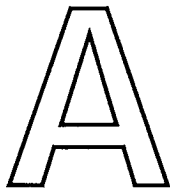
You can add 16K, 32K or more than one module to your system. The Rameses Computer can be expanded up to 64K byte.

4. The standard ASCII keyboard

A standard rubber keyboard having superior feeling and more ease to key in.



# **APPENDIX**



## **TABLE OF BASIC STATEMENTS**



## The BASIC-Quick Reference Manual

Numbers are stored to an accuracy of 6 digits. The largest number you can get is  $10^{38}$ . All statements can be used either as commands or in program.

For calculation answers, comprising more than six (6) digits result, the screen will show a decimal representation of the answer.

E.G.  $2^{19} = 524288$  will show as 524288 (because it has 6 digits)  
 BUT  $2^{20} = 1048576$  will show as 1.04857E + 06

This means that the decimal point of the answer shown is to be moved 6 places to the Right. Similarly,  $10^{38}$  give 1.0E 38,  $2^{38}$  gives 2.7487E + 11"

### Functions:

#### 1) Arithmetic operators

+, -, \*, /, \*\*

#### 2) Relational operators

>, <, =, >=, <=, <>

#### 3) Arithmetic functions:

SQR — Square root

INT — Integer part

RND — Random number

ABS — Absolute magnitude

SGN — Sign

COS — Cosine

SIN — Sine

EXP —  $e^x$

TAN — Tangent

LOG — Natural logarithm

## 4) String functions:

LEN – Length  
 STR\$ – String of numeric argument  
 VAL – Numeric value of string  
 ASC – ASCII value  
 CHR\$ – Character  
 LEFT\$ – Left character  
 MID\$ – Middle character  
 RIGHT\$ – Right character

## 5) Logical operators

AND  
 OR Relation and logical expressions have value 1 if true, 0 if false.  
 NOT

## 6) Graphic and sound functions:

CLS – Clear screen  
 PLOT – Plot character on screen  
 COLOR – Set colour  
 SOUND – Produce tone of different frequency and duration  
 CHAR – Define character  
 JOY – Joy stick function

## 7) Program statements

DIM – Dimensions  
 STOP  
 END  
 GOTO  
 GOSUB  
 RETURN  
 FOR.....TO.....STEP  
 NEXT  
 REM  
 IF.....THEN  
 INPUT  
 PRINT  
 PRINT TAB



LET  
DATA  
READ  
RESTORE

8) Commands:

LIST  
RUN  
NEW  
CONT  
CLOAD – Load program on tape  
CSAVE – Save program on tape  
CRUN – Load program on tape and run  
CNT'L C – To halt program

9) Other Statements

PEEK – Return the value stored at the location specified  
POKE – Load a value into a specified location  
LPRINT – Print on line printer  
LLIST – List on line printer



# **APPENDIX**

# **B**

## **ERROR MESSAGES**



## Error Messages

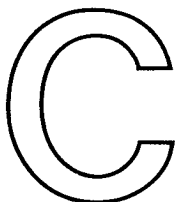
If Rameses BASIC detects an error that causes program execution to terminate, an error message is printed. The explanation of each error is given below.

### The error message

1. NEXT WITHOUT FOR ERROR
2. WITHOUT GOSUB ERROR
3. MISSING LINE NUMBER ERROR
4. MISSING OPERAND ERROR
5. SYNTAX ERROR
6. OVERFLOW ERROR
7. ILLEGAL NESTED FOR ERROR
8. ILLEGAL NESTED GOSUB ERROR
9. SYSTEM ERROR
10. STACK OVERFLOW ERROR
11. ILLEGAL OPERAND ERROR
12. IF ERROR
13. ( ( ..... ) ) ERROR
14. ( ( ..... ) ) LEVEL ERROR
15. STRING NOT FOUND ERROR
16. STRING EVALUATION ERROR
17. DIVIDED BY ZERO ERROR
18. OUT OF DATA ERROR
19. DATA AREA OVERFLOW
20. DIM ERROR
21. STRING LENGTH ERROR



## **APPENDIX**



### **TYPICAL PROGRAM EXAMPLES**

- (i) FIND H.C.F.
- (ii) SOLVE QUADRATIC EQUATION
- (iii) FIND THE SQUARE ROOT BY ITERATIVE METHOD
- (iv) FIND THE AREA OF A TRIANGLE
- (v) MARK SIX GAME





Example (i): 10 REM FIND H.C.F.  
20 INPUT X, Y  
30 IF  $X > Y$  THEN  $X = X - Y$   
40 IF  $X < Y$  THEN  $Y = Y - X$   
50 IF  $X > < Y$  THEN 30  
60 PRINT X  
70 END  
  
RUN  
? 3  
? 9  
3

Example (ii): *10 REM SOLVE QUADRATIC EQUATION.*

*20 REM A X\*X + B\*X + C = 0*

*25 REM X\*X - 1.5X + 0.5 = 0*

*30 DATA 1, -1.5, 0.5*

*40 READ A, B, C*

*50 D=B\*B - 4\*A\*C*

*60 IF D < 0 THEN 120*

*70 D = SQR (D)*

*80 X = (-B+D)/(2\*A)*

*90 Y = (-B-D)/(2\*A)*

*100 PRINT X, Y*

*110 END*

*120 PRINT "NO REAL ROOTS"*

*130 END*

*RUN*

*1 0.5*

```

Example (iii): 10 REM: TAKE SQUARE ROOT
                20 REM: BY ITERATIVE METHOD
                30 REM: I = INITIAL VALUE
                40 REM: N NUMBER
                50 N = 20
                60 PRINT "VALUE I=";
                70 INPUT I
                80 FOR J = 1 TO 10
                90 I = (I+N/I) * 0.5
                100 PRINT J; "APPROXIMATION"; I
                120 NEXT I
                150 PRINT "SQUARE ROOT I="; I
                200 END

```

RUN

VALUE I =

20

```

1 APPROXIMATION 10.5
2 APPROXIMATION 6.2023
3 APPROXIMATION 4.7134
4 APPROXIMATION 4.4721
6 APPROXIMATION 4.4721
7 APPROXIMATION 4.4721
8 APPROXIMATION 4.4721
9 APPROXIMATION 4.4721
10 APPROXIMATION 4.4721
SQUARE ROOT I = 4.4721

```

Example (iv): *10 REM FIND AREA OF TRIANGLE*

*20 REM A, B, AND C ARE 3 SIDES*

*30 INPUT A, B, C*

*40 S = A+B+C*

*50 S = 0.5\*S*

*60 P = S\*(S-A)\*(S-B)\*(S-C)*

*70 A = SQR (P)*

*80 PRINT "AREA="; A*

*90 END*

*RUN*

*A = 3*

*B = 4*

*C = 5*

*AREA = 6.0*

- \* This program does not check the values of A, B, and C such that they can form a triangle. As an exercise you may add a few statements for the checking.

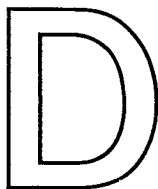
Example (v): *10 REM MARK SIX GAME*  
*20 DIM A(6)*  
*30 FOR N = 1 TO 6*  
*40 R = RND (0) \* 36*  
*50 P = INT(R)*  
*60 A(N) = P+1*  
*70 IF N = 1 THEN 150*  
*80 M = N-1*  
*90 J = 0*  
*100 J = J+1*  
*110 IF A(N) = A(J) THEN 40*  
*120 IF J < M THEN 100*  
*150 PRINT A(N)*  
*200 NEXT N*  
*300 END*

*RUN*

<i>3</i>	}	or any random number
<i>4</i>		
<i>32</i>		
<i>17</i>		
<i>24</i>		
<i>15</i>	)	



**APPENDIX**



**ASCII CODES**





ASCII CODE	CHARACTER	ASCII CODE	CHARACTER
32	(Space)	64	@ (at sign)
33	! (exclamation point)	65	A
34	" (quote)	66	B
35	# (number or pound sign)	67	C
36	\$ (dollar)	68	D
37	% (percent)	69	E
38	& (ampersand)	70	F
39	' (apostrophe)	71	G
40	( (open parenthesis)	72	H
41	) (close parenthesis)	73	I
42	* (asterisk)	74	J
43	+ (plus)	75	K
44	, (comma)	76	L
45	- (minus)	77	M
46	• (period)	78	N
47	/ (slant)	79	O
48	0	80	P
49	1	81	Q
50	2	82	R
51	3	83	S
52	4	84	T
53	5	85	U
54	6	86	V
55	7	87	W
56	8	88	X
57	9	89	Y
58	: (colon)	90	Z
59	; (semicolon)		
60	< (less than)		
61	= (equals)		
62	> (greater than)		
63	? (question mark)		





HANIMEX

Rameses®

# V1 BASIC INTERPRETER

©1982 VTL  
MADE IN HONG KONG  
91-0129-01