Creative Computer

• •••



BASIC F Manual

2nd ED.



Easy BASIC for Science

**Copyright © 1984 by SORD COMPUTER CORPORATION**

All rights reserved. Printed in Japan.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of SORD COMPUTER CORPORATION Japan.

Contents

Chapter 1 Introduction 2

1. [1 Getting Started 2](#bookmark2)

[1-2 BASIC-F and other BASICS 3](#bookmark3)

1. -3 BASIC-F Variable Types and Numerical

Representative 3

1. [-4 BASIC-F Operating Environment 3](#bookmark5)

[1-5 BASIC-F Files and Disk System 4](#bookmark6)

[1-6 Using Disk Files 5](#bookmark7)

1. [7 Using Peripherals 6](#bookmark8)

[Chapter 2 General Description 8](#bookmark9)

1. [1 Screen Control 8](#bookmark10)

[2-2 Screen Buffers and Graphics Modes 9](#bookmark11)

[2-3 The M5 Keyboard 10](#bookmark14)

[2-4 Editing 10](#bookmark15)

[2-5 Graphics Modes 11](#bookmark16)

[2-6 Programming Style 11](#bookmark17)

1. [- 7 Statement Syntax 13](#bookmark18)

[2-8 Sound 13](#bookmark19)

1. [9 Use of Keyboard and Printer 14](#bookmark20)

[Chapter 3 Basic Commands 16](#bookmark21)

1. [1 Functions 102](#bookmark93)

[Chapter 4 Appl ications Section 147](#bookmark118)

1. [1 Loan Payments 148](#bookmark119)

[4-2 Initial Investment 150](#bookmark120)

[4-3 Regular Deposits 152](#bookmark121)

[4-4 Future Value of Regular Deposits (Annuity) 154](#bookmark122)

[4-5 Remaining Balance on a Loan 156](#bookmark123)

[4-6 Prime Factors 158](#bookmark124)

[4-7 Long Number Arithmetic 160](#bookmark125)

Appendix A Character Codes 164

[B M5 Color Codes 167](#bookmark127)

[C M5 Control Codes 168](#bookmark128)

D M5 CRT Layout Sheets 171

[E Keyboard Codes 173](#bookmark131)

F Port Assignment Table 176

[G Memory Maps 177](#bookmark134)

[H Error Codes 180](#bookmark138)

Commands and Function Index 184

Preface

This manual will be read by individuals with a wide range of skills. It is intended primarily for those who are familiar with computers and have used a BASIC language before. We have nevertheless tried to anticipate the needs of as many people as possible. At times this manual will appear too simple and detailed and at others too complex. Please bear with us when the manual doesn’t match your personal expertise exactly. We hope that this manual will be helpful and easy to use. Please feel free to send us your comments and suggestions.

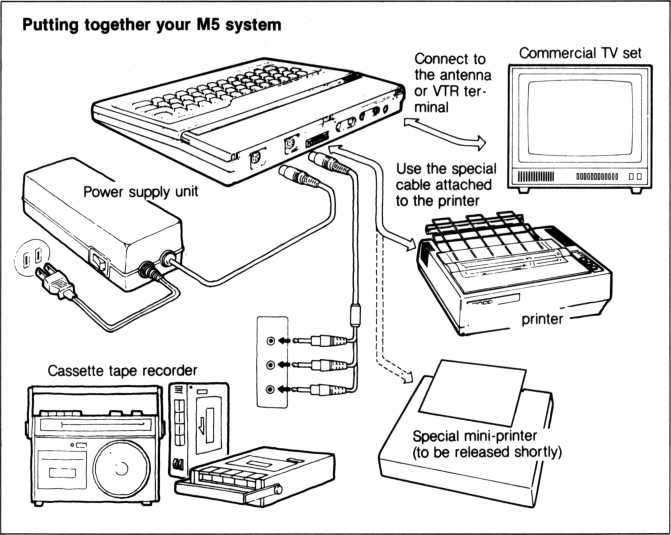
Parts one and two of this manual cover the basic capabilities of the M5 and BASIC-F, such as screen functions, files, and arithmetic. Part three contains individual explanations of each BASIC-F statement, function, and system command, arranged alphabetically each with a short example. The fourth section contains a set of useful applications programs which demonstrate the various features of the M5.

1. Introduction

Welcome to SORD’s BASIC-F. In this section we will describe the way to set up your M5 to run with BASIC-F. We will also give a brief description of the overall difference between BASIC-G and BASIC-F.

* 1. Getting Started

If your M5 is not assembled and running look at the figure below. Connect the various components as shown and power up your computer! Make sure that the BASIC-F cartridge is properly inserted before turning the power on.



Your M5 is now ready for use. The top left corner of your screen should display:

BASIC-F

READY

L

If it does not display this message, turn the power off and check all the connections between components in your system.

* 1. BASIC-F and other BASICS

BASIC-F is an enhancement of BASIC-I, which also contains some of the important graphics features of BASIC-G. The most important difference is the inclusion of floating-point arithmetic. With BASIC-F you can perform all kinds of mathematical calculations.

* 1. BASIC-F Variable Types and Numerical Representation

BASIC-F has integer, real and string variable types. Constants may also be expressed in hexadecimal. Integers are represented internally by sixteen bits in two’s complement form, giving a range of -32768 to 32767. Floating-point numbers range from ±8.6E-78 to ±7.2E—75.

Arithmetic follows this operator priority:

(,) comma functions exponent sign \*,/

MOD + > —

relational operators logical operators

We recommend that programs enforce all operations with parentheses. This removes confusion, improves readability, and prevents errors when programs are transported to and from other systems where operator precedence may be different or unknown.

Thus, example A is recommended over B even though they are arithmetically equivalent:

1. X = (2 \* C) + 4
2. X = 2 \* C + 4
   1. BASIC-F Operating Environment

BASIC-F has two operating environments: a program creation /edit mode and a program execution mode. Programs are executed, in the operating mode, simply by typing RUN.

Programs written in BASIC-G can be transferred to BASIC-F by using the LIST command to create a tape of the program listing. To access the stored program, use the I NT command, followed by RETURN, then transfer the program with the OLD command. When there are no reserved words in BASIC-F an error will occur and tape read-out will end.

Once a program is transferred it can be run in both BASIC-G and BASIC-F, and can be saved in any form. Note, however, that language-dependent commands such as floating-point operations and graphics will have to be modified after transfer.

* 1. BASIC-F Files and Disk System

BASIC-F can use data files to store and retrieve information. The file system is similar to that found on larger SORD computers with advanced operating systems, but, since the disks used are 3-1/2 inch media, the files are not interchangeable with these machines.

When you use SORD’s FD-5 Floppy Disk Drive Unit with the M5 and BASIC-F, you need a number of commands which are not strictly speaking BASIC-F commands. These are not included in the list in • Section 2 of this manual, but are described in detail in the CF-5/S Operating System Manual. A brief description is given below.

The disks are used to store both programs and data in units called files. A list of which files are on a disk is itself stored on the disk. When you wish to see what files are stored on a disk you must request that this list be displayed on the screen of your M5. To do this the command used is RUN”LIST. When you enter RUN”LIST the M5 will request that you enter the unit number of the disk that you wish listed. After you give the number, the list of names will be displayed on the screen. You will also be shown how much space on each disk is in use and how much remains. Along with each file name some file attributes will be displayed, such as the file type, and the file size.

Before a new disk can be used, the computer must prepare it.

This process is called disk initialization. The command to prepare a new disk for use is RUN”INIT. RUN”INIT clears the disk of any existing material and creates an empty list to receive the names of the files that you will place on the disk.

Since disks may be damaged by accident or by errors in your programs, it is wise to copy disks so that you have extra copies. It is also necessary to copy files for other reasons, and so BASIC-F allows you to copy one disk to another. The command is RUN”

COPY.

Each file that is stored on a disk has a name. It may be desirable to change the name of a given file from time to time. A command called RUN”RENAME is supplied to do this. When you enter this command, you must supply the old name of the file and then the new name. The computer will prompt you for each of these names separately. Each name may be up to 9 letters long.

Each file also has several attributes. These attributes change the way that the computer can use the files. These attributes are represented by the letters A, P, R, and W. The command to change attributes is RUN”CHATR. The attributes are fully explained in the CF-5/S Manual.

To use any of these SEVEN commands, just type it as it has been typed above. Then enter RETURN. The M5 will prompt you for each input required to complete the command.

* 1. Using Disk Files

With the disk system it is possible to use the disk for storing data as well as programs. The procedure is essentially the same as inputting from the keyboard and outputting to the display, but a few descriptors are needed to specify the file. We will now describe how to use data files from within a program.

Data files can be thought of as large arrays which are stored on the disk instead of in memory as regular arrays. Instead of just numbers or strings, each cell of a file can contain a block of information, and each block can be different in form from others. Before we can use a disk file, we must tell BASIC-F that we intend to do so. This is similar to using the DIM statement for arrays. The command to do this is OPEN, and it tells BASIC-F to open access to a file of a given name. It also associates the file with a channel number. Channel numbers are always used to refer to the file after it is OPENed. This allows you to write programs without knowing what the files are called, and it allows one program to process many different files. The OPEN statement may be used with a variable name for the file name so that it can be changed at run time. The channel numbers are 0 to 15 so up to 16 files may be accessed at once. Channel O is usually used by the console. When processing is complete, you must CLOSE the file with the CLOSE command.

To use OPEN with the screen display, keyboard, or printer, the format is:

OPEN “descriptor” as # channel no.

For outputting characters to a graphic display or printer see elsewhere in this manual.

To use OPEN with a cassette unit or floppy disk drive, the format is:

r input

as # channel no.

OPEN “descriptor” for J output

\* append

‘Input’ is used when you want to read data from a file and ‘output’ is used when you want to write data to a file. ‘Append’ is used when you want to add data to an already existing file, but is only applicable with the disk system.

To use OPEN with random files on disk, the format is:

OPEN “descriptor” as # channel no. record record length

The format for the CLOSE statement is:

CLOSE # channel no.

Many channels can be closed at the same time by specifying their numbers, separated by commas.

If CLOSE only is executed, all OPEN files will be CLOSED. Be sure to CLOSE all files when you are finished using them. After CLOSEing disk files perform a disk update.

When accessing a cell of an array, we use a special notation: square brackets with an index number following the name of the array. There is a special notation for files as well. To get the information out of a file cell, we use the GET command. To put information into a cell of a file we use the PUT command. Cells in a file are normally referred to as records. Each file in BASIC-F can have up to 65535 records.

* 1. Using Peripherals

Besides the floppy disk drive, you can access a variety of peripherals using BASIC-F. The following table gives the standard abbreviations for your peripherals.

|  |  |  |
| --- | --- | --- |
| Name | Output Device | Input Device |
| • CNS (Console) | Character display | Keyboard |
| GRP (Graphic) | Graphic display |  |
| PRT (Printer) | Character printer |  |
| PRI (Image) | Image printer |  |
| CMT (Cassette) | Audio cassette tape Floppy disk drive | |
| FX (File) |

To operate your cassette tape, printer, or display the procedure is much the same as the procedure for using the disk drive as outlined in the previous section. You use the OPEN and CLOSE commands, and if appropriate to the peripheral, you can utilize PRINT, INPUT, SAVE, OLD, CHAIN, and LIST.

In BASIC-F all data inputs and outputs are carried out at the level of files. To operate the peripheral you need to use a special file name called a descriptor. The form of the descriptor is as follows:

Device Name Device Number Drive Number: File Name

A drive number need only be used with a disk drive; device number is significant only with the disk drive. Channels available range from 0 to 15, but as it is usual for the console to use channel 0, in practice 1 to 15 are available. File names must be no more than 9 characters in length. Thus to use OPEN with your printer, for example, the format is as follows:

Open “Descriptor” as #Channel Number

With a disk drive or cassette tape the option to add to your data exists, so the format is a little different:

OPEN <file name> [for {input|output|append)] as # <CH> [Record < record length >] {OR |:)

The format for the PRINT command is as follows:

Print [# <channel number>] <expression>,... {CR|:}

With disks or cassette tapes only files which have been OPENed in the manner described above can be PRINTed. The format for the INPUT command is as follows:

INPUT [<prompt> 1 # <channel number>]{,|) <variable> [,..] (CR|)

You can LIST programs on the printer with the format:

List “PRT” Return

This format uses the characters integral to the printer; to print out just like on the screen use:

List “PRI:” Return

1. General Description

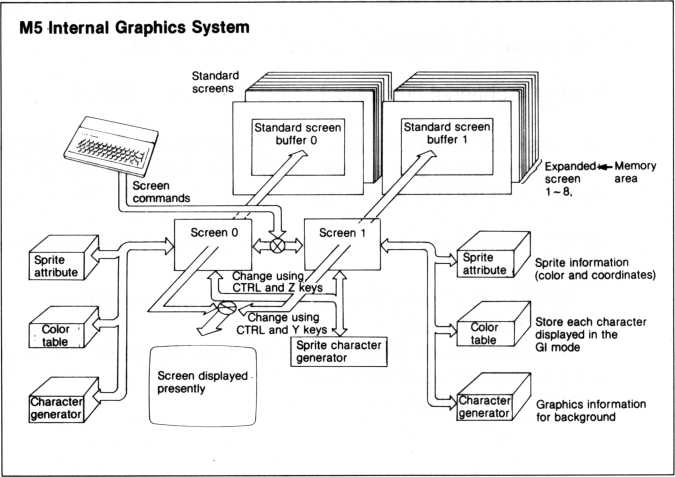
In this section we will describe the special features of the M5.

The M5 has many special features which are different from other computers. These include the M5’s special graphics capabilities and music and sound generation. Used together, these features allow you to create advanced programs on the M5.

* 1. Screen Control

The M5 computer uses memory mapping for the display screen. The computer display screen is represented by a complete block of memory. When you type a character, the M5 places the character in the video memory and the circuitry of the M5 places the image on the screen, where you can see it. A powerful feature of the M5 is that it has two such video memory buffers. This allows you to ‘type’ on two screens or to write to two different places. A simple command will tell the M5 to change which of the two buffers will actually be shown on the screen. When one is displayed the other video buffer is preserved exactly. The computer can switch between the two displays quickly without any screen flicker or noise.

This feature is extremely powerful and allows the development of • very useful interaction environments. For instance, a program could, when requested, display help information on a second screen allowing a user to request help, receive it, and continue using the program, all without disturbing the display of the original program!



* 1. Screen Buffers and Graphics Modes

DISPLAY CONTROL

The M5’s two screen buffers are referred to as ‘screen 0’ and ‘screen 1’. When the computer is turned on, screen 0 is displayed. At any time screen 0 may be displayed either by PRINTing a ‘control U’ or by typing it from the keyboard.

To change to the ‘other’ screen type PRINT CRTL-V. CTRL-V always causes the screen to change. If the screen now displayed is screen 0 then it will change to screen 1. If it is screen 1 then it will change to screen 0.

WRITE CONTROL

On most computers with only one display screen, any text typed from the keyboard will go directly to the screen. On the M5, text entered from the keyboard can go to either of the screens. This can result in some confusing situations, so be careful!

When the M5 is turned on, screen 0 is displayed, and all text is displayed on that screen, like a normal computer. The M5 can be instructed to put text on the hidden screen by PRINTing or typing a CTRL-Z. CTRL-Z is the WRITE TO HIDDEN SCREEN command.

If screen 0 is visible, the text will go to screen 1 and if screen 1 is visible the text will go to screen 0. The M5 maintains a cursor position for each screen. If text is written alternately to one screen and then the other and back again, it will be as if the text had been entered separately on each screen.

The M5 also has the ability to flip screens and write control at the same time. This allows the user to look at the hidden screen and continue to type on the previous screen. For example, if the user is typing on screen 0 but needs some information .from screen 1, the user can display screen 1 but continue typing on screen 0. This is done with CTRL-Y. Notice that this command is actually redundant, the same effect being achieved with CTRL-V, CTRL-Z.

TABLE OF WRITE CONTROL COMMANDS

|  |  |  |  |
| --- | --- | --- | --- |
| CODE | DEC | HEX | effect |
| CTRL-Q | 20 | 14 | multicolor |
| CTRL-R | 18 | 12 | Gil |
| CTRL-S | 19 | 13 | Gl |
| CTRL-T | 20 | 14 | enter TEXT mode |
| CTRL-U | 21 | 15 | display screen 0 |
| CTRL-V | 22 | 16 | display alternate screen |
| CRTL-Y | 25 | 19 | display alternate screen and write to hidden screen |
| CTRL-Z | 26 | 1A | write to hidden screen |

Hr

* 1. The M5 Keyboard

The M5 has several features which make programming easier. The keyboard has several control functions which save typing time. The following table summarises these functions:

-shift screen down -shift screen left -shift screen up -shift screen right -delete previous character -tab

CTRL-C

CTRL-D

CTRL-E

CTRL-F

CTRL-H

CTRL-1

CTRL-J

CTRL-K

CTRL-0

CTRL-P

CTRL-X

CTRL-up

CTRL-down

CTRL-left

CTRL-right

CTRL-reset

SHI FT-reset

-move cursor down

-cursor home

-exit insert mode

-enter insert mode

-clear to end of line

-move cursor up

-move cursor down

-move cursor left

-move cursor right

-interrupt program execution

-stop program execution permanently

These functions can be utilized by typing them into the M5 from the keyboard directly, or by a program using the PRINT statement. When used in a PRINT statement, they must be enclosed in double quotes as part of a string. Furthermore, you must indicate to the M5 that you wish to use the control code but not execute it immediately. This is done by typing ‘SHIFT-CTRL-letter,’ all at once. This tells the M5 that you wish the PRINT statement to use the correct code but that it should not be used now. When this is done, the controlled character will be displayed in reverse video.

* 1. Editing

These commands are often useful when editing. Naturally, when typing in programs, errors will occur. The M5 has several screen editing functions which facilitate error correction. When an error is discovered it is not necessary to retype the entire line. When the line containing the error is displayed on the screen, the arrow keys can be used to move the cursor directly to the position of the error. The error can then be corrected right on the screen. When the corrections have been made, entering a ‘CR’ will tell BASIC-F to enter the new, corrected line into the program.

The other control keys are also useful when editing. For instance, CTRL-X will delete the characters all the way to the end of the line. CTRL-N will move the cursor to the beginning of the next line.

CTRL-H will delete the character under the cursor, anywhere in the line. CTRL-P will allow you to insert new characters into the line and CTRL-0 will exit insert mode. At first these functions may seem confusing and time consuming, but with practice they speed up programming considerably. The screen also has cursor wrap around. This means that if you move past the edge of the screen the cursor will reappear on the opposite edge. This can save time if you have to move the cursor a long distance.

* 1. Graphics Modes

Under BASIC-F some of the graphics capabilities of BASIC-G have been retained. Graphics are facilitated through the inclusion of several commands such as DRAW, PLOT, PAINT, and COLOR. These are described in the command section of this manual. Here we will only describe the characteristics of graphics in the M5.

The M5 implements graphics through the use of different screen modes. Recall that the M5 has two screens. Each of these screens can be set to different modes. Under BASIC-F there are four such modes. Each screen mode has different properties which facilitate different tasks.

In TEXT MODE (G1) the screen can display 24 lines of 40 characters at once. Each character is composed of 48 dots in a 6x8 font. TEXT MODE is set for either screen by PRINTing or typing a CRTL-T.

An alternate screen mode is G2. Graphics cannot be displayed, but more characters will fit on the screen. The M5 is in G1 mode when first powered up.

mode font

Gl 8x8 32x24

Gil 8x8 32x24

Text 6x8 40|x24

Multi 8x8 32x24

* 1. Programming Style

BASIC-F includes features which can make your programs easier to understand and debug. Programs in BASIC-F can be more clearly written, easier to debug and more efficient. They can also be stylistically superior. Features include:

-labels for branching -indenting of code -a REPEAT command

BASIC-F has the ability to indent lines of program text. In the following example the inside of the loop is indented.

Labels can be attached to line numbers. This allows routines also to be given mnemonic names. A mnemonic name relates a name to the function of the routine itself. This allows a reader to see quickly what a section of a program is supposed to do. Labels can be used with GOTO, GOSUB, and RESTORE in BASIC-F.

BASIC-F has the ability to indent lines of program text. In the following example the inside of the loop is indented.

Labels can be attached to line numbers. This allows routines also to be given mnemonic names. A mnemonic name relates a name to the function of the routine itself. This allows a reader to see quickly what a section of a program is supposed to do. Labels can be used with GOTO, GOSUB, and RESTORE in BASIC-F.

Within REPEAT and FOR loops, code should be indented two or three spaces to make programs more legible. This shows which instructions are within the scope of a given loop. For example:

tor 1=1 to 100 9 o s u b $GETURL 9 os u b fDOPRQC n ext I

100 110 120 130

This clearly shows which commands belong inside the loop and where the loop ends. This is a great aid to debugging and understanding programs. For this reason, BASIC-F does not remove the extra blanks as do some other BASICS.

The REPEAT command should be used whenever a counter is not required for a loop. This reduces the number of variables used, and thereby improves readability and saves both memory and memory accesses. REPEAT loops are always executed at least once, since the test for looping occurs at the end of the loop. Code should also be indented in the REPEAT loop. For example:

100 repeat

1. 9osub fGET INPUT
2. 9 o s u b ^PROCESS

130 **until** DQNEf="TRUE"

Older BASICS allowed variables to have one letter and one number as names. Modern BASICS such as BASIC-F no longer have this restriction. It is good programming practice to make your variable names meaningful. Using mnemonics helps debugging and understanding programs. In BASIC-F variables may have names consisting of up to 32 alphanumeric characters. Of course, names that are too long also increase the likelihood of typing and reading errors, remember to keep a balance.

* 1. Statement Syntax

Each line of a BASIC-F program begins with a line number. These line numbers range from 1 to 32,767 Each line may contain several BASIC statements separated by full colons, Each line may be up to 252 characters in length.

After a line number any BASIC statement may occur. Also, after a line number, but not a colon, a label should occur. The label must begin with a dollar sign '$’, and no blanks should occur between the line number and the label name. The label can then be used by other BASIC commands to refer to the line number. When referring to the label the dollar sign must be included.

A special statement, called a comment, may also appear on a line. When a comment appears, no other BASIC statements should occur after it on the line. A comment and anything after it is always ignored by BASIC-F. A comment is indicated by the keyword REM, an exclamation mark T, or an apostrophe

The syntax of all other statements and commands is explicitly defined in this section of the manual.

* 1. Sound

The M5 has a SN sound generation chip. This gives the M5 the ability to produce sound effects. These functions can be performed from BASIC-F for use in any program.

To produce sounds, the M5 contains three tone generators and one noise generator. These can be used singly or in combination to produce a wide variety of sounds. BASIC-F has one command to control all four of these sound generators. The command is SG and it has three parameters: the channel, the frequency and the volume of the sound to be produced. The second parameter describes the type of noise to be produced when accessing the noise generator. The comand is as follows:

SG channel, frequency, volume

The sound channels are 0,1, and 2. The noise channel is number 3. The frequency value is specified as a value from 1 to 1023. For any given value the frequency is calculated as follows:

frequency = 111.86 (khz)/desired frequency

For example, a middle C on a piano is'261.6 (Hz). Thus:

code value = 111.86 (khz)/256  
= 428

and:

SG 0,428, 15

will produce a pure middle C from your TV.

When the channel specified is 3, the noise generator is accessed. The noise generator produces eight types of noise from 0 to 7. When used in combination with channel 2, interesting sounds can be produced.

See the command section for more information about SG.

* 1. Use of Keyboard and Printer

The printer port on the M5 can be used to connect several printers to your computer. With dot matrix printers, the M5 can print out graphics images from the screen as well as text and program listings.

Be sure to read the instructions with your printer before connecting it to your M5. If you have problems contact your dealer.

The M5 connects to printers via a parallel port. This requires a special cable for the connector on both the printer and the M5.

The printer is used for two main purposes: to print out program listings and to produce ‘hard copy’ output of programs written on the M5. Program listings are produced by the LIST command and its variations. For further information on these commands see the command section of this manual.

Two kinds of hard copy output can be produced by the M5. One is standard text, such as found in letters and reports. The other form of output is graphic images. Text is output directly to the printer with variations of the PRINT command. See the command section for further information.

Graphics images are sent to the printer with the GCOPY command. Again, see the command section for further details. Graphics images can only be printed on dot matrix printers.

1. Basic Commands

In this section we provide a list of all BASIC-F statements and system commands arranged alphabetically. Each statement has a small example shown in isolation; for more examples see the section of applications programs.

The FORMAT part of the statement descriptions is read as follows:

(CRI:) means that the statement can be ended with either a carriage return or a full colon and another statement.

[ ] indicates that the component in braces is optional.

[,..] indicates that the previous component is optionally repeatable.

| either of two options can be used.

< > the word(s) inside the brackets refer(s) to a single

object.

$< label name> {CR |:}

Remark statement or label-name.

$

**FORMAT: FUNCTION : Comments:**

**Example:**

Up to 32 characters in length; the $ must be situated next to the line number.

1. 0 0 ‘3 o t o $ S U 8 i 0 0 0 **■$** S U 8

AUTO

FORMAT : AUTO [<first line number>] [,< increment > ]

|  |  |
| --- | --- |
| FUNCTION : | AUTOmatic line numbering when writing a program. |
| Comments: | When entering a program, AUTO will put a line number on the screen after you complete each line. AUTO sometimes conflicts with the natural progression of programming by enforcing an artificial regime on the order in which code is entered. The use of this command is a matter of personal style. |
| Example: | fl U T 0 1 0'Q ? 2 0 10 0 a = 1 12 0 b = 2 14 0 c = a + b 1 6 0 P r i n t- c |

BCOL

FORMAT : BCOL [< color code> ] {CR

|  |  |
| --- | --- |
| FUNCTION : | Sets the background color of the screen. |
| Comments: | The default color code is 0, no color. If the color code is the same as that for characters (FCOL) then characters will be invisible. See Appendix B for the color codes. |
| Example: | 10’ BCOL TEST 28’  3y for 1=0 to 10 4 0 be o1 I  4 5 c1s:P r i n t cursor( 15? 13) J I 50 sleep 1   1. 0 n e x t I 2. 0 b c o 1 0 |

CALL < address > [,<AF registers >] [,<BC registers >] [,<DE registers>] [,<HL registers>]

CALL transfers program control to a specific machine address, after leaving the return address on the stack.

CALL

**FORMAT: FUNCTION : Comments:**

This command is used to call machine language routines. These can be part of the M5 monitor, or written by the user. When the machine routine executes a RET instruction, control is returned to the BASIC-F program. After the return, the following registers are stored in the following memory locations:

PSW : &7262

A : &7263

B : &7265

C : &7264

D : &7267

E : &7266

H : &7269

L : &7268

Caution should be exercised with the use of CALL. Always save your program to disk or tape before testing a program with a CALL. Often the contents of the machine memory will be altered and your program will no longer even LIST properly. On the other hand, machine routines will run very fast, often 300 to 1000 times faster than a BASIC routine which performs the same function. Thus if speed is required, the CALL function may be necessary. (Also it can set the values to the registers; see the REG function.) You should however refer to a machine language manual for the proper use of this statement.

**Example:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | 9 | o s u | b $SE | TRGU |
| 20 | C | a 1 1 | RDDR | ESS ? |
| 30 | e | n d |  |  |
| 40 | $ | SETR0UTI | | NE |
| 5 0 | fi | DDR | ESS = & | 0C9 7 |
| 60 | B | CRE | G = & 0 5 | 00 |
| 70 | r | etu | r n |  |

CHAIN

FORMAT : CHAIN [<file-name >] [, all] |CR

|  |  |
| --- | --- |
| FUNCTION : | Retrieves and executes a program stored on tape or disk. |
| Comments: | A statement on the same line as a CHAIN command, separated by a will never be executed. The variables and values of the current program are lost. And by setting [all], the variables and values will remain as is. |
| Example: | 1 0 6 P r i n t " D o n e "  1 1 0 r h a i n " P R 0 G 2 " |

CLEAR

FORMAT: CLEAR [<work field >] [, < last user field >] {CR

|  |  |
| --- | --- |
| FUNCTION : | Clears a portion of memory for use as a PAINT and character buffer. Memory is freed in 256 byte blocks. The second parameter sets the highest memory location used by the program. |
| Comments: | The top of the memory field is below the buffer, and prevents the program from running into it as it uses memory. |
| Example: | 1 00 clear 5 1 2 j &8 F F F |

CLIST

FORMAT : C LIST [< file name>] [, < line number 1 >]

|  |  |
| --- | --- |
| FUNCTION : | [, <line number 2>] {CR |:}  Lists a program in upper case only. |
| Comments: | This function is identical to LIST except that all lower case letters are converted to capitals before printing. |
| Example: | 1. V i\* '='!-'•=\* -:d t- 1   28 fi$=inkey$  25 Print asci i CH$> 1 39 until Pif = ch r $ < 13) 1  cl i s t 1 10 RE PERT   1. 0 A $ = I N K E V $   25 PRINT ASCII <A$>   1. 0 1JN TIL A$ = CHR$< 13) |

CLOSE

FORMAT : CLOSE [# < channel no.> ] [CR

|  |  |
| --- | --- |
| FUNCTION : | Ends usage of user file. |
| Comments: | If the channel number is omitted, all channels will stop. |
| Example: | 1 9 0 oP en " PR T : 11 as#2 110 p r i n t # 2 " '3 o o d "  12 0 c1os e # 2 1 ■ j y e n d |

CLS [<initialize code>] {CR |:j Clears the screen.

The initialize code may specify the character code which will fill the screen. This is normally a null.

CLS

**FORMAT: FUNCTION : Comments:**

**Example:**

C 1 S

1i 0 p ri nt "This is the toP o t the

S C\*r" 8 0 i“i "

COLOR

FORMAT : COLOR < character-code >, < color-code>

|  |  |
| --- | --- |
| FUNCTION : | Sets up the character color in the Gl and Gil modes. |
| Comments: | The higher-order four bits indicate the character color and the lower-four bits designate the background color, or  color-code = character - color x 16+background-color or  color-code = & HL (H = character-color, L= background-color: H, L are hexadecimal)   * The Gl mode—each time a character is colored, it actually affects seven other characters with contiguous ASCII codes (If the 16x16 ASCII table is split into upper and lower halves, each half contains columns of eight characters. These eight are colored identically). * The Gil mode—each character is colored individually. |
| Example: | c o1or 6 3? & y0 |

CONSOLE

FORMAT: CONSOLE [<A>] [, <B>][, <C>][, <D>][, <E>]

[, < F>] [, <G>] {OR |:j

FUNCTION : Enables/Disables keyboard function keys.

Comments:

|  |  |  |
| --- | --- | --- |
| Function | 0 | 1 |
| A Keyboard sounds | OFF | ON |
| B Generate keyboard keywords | OFF | ON |
| C Display page | page 0 | page 1 |
| D Process page | page 0 | page 1 |
| E Display mode 0 : | M, 1 : G2, 2 : G1, 3 : T |  |
| F Screen lock | | |
| G ACMT Baud rate 33 | = 2,000 Baud |  |

Example : c o **nsole 8,, 9 ,9,3**

CONT

FORMAT: CONT {OR

|  |  |
| --- | --- |
| FUNCTION : | Restarts a program after it has been interrupted by a STOP command or a keyboard interrupt. |
| Comments: | CONTinue will not work after an END command has been executed. By entering STOP commands while testing a new program, the programmer may check individual sections of a program for errors. |
| Example: | 1 0 0 p r i n t- " Here w e are?..." 1 1 0 s t o P \_  1 2 0 P ri nt "Here we 9 o ? . . . " 130 end r u n  Here we ere?■■■  StoP at 110 R e a d y c o n t  Here we 9 o ? • • •  R e a d y |

CURSOR

FORMAT : CURSOR (X, Y)

|  |  |
| --- | --- |
| FUNCTION : | Moves the cursor to the coordinates specified by (X, Y). |
| Comments: | A semicolonis usually used after this keyword. |
| Example: | 1 0 8 P r i n i c u r s o r (15? 1 0 } ; 11 G Q M £ Q f |

DATA <constant > (CR |:j

Stores constant information to be used by the program and accessed via READ.

DATA

**FORMAT: FUNCTION :**

**Comments: Example:**

Data may be numerical of any type, or character data. Collect DATA statements near the end of the program. When using disk systems keep DATA use to a minimum as they occupy valuable memory. Instead keep the information on a file.

**10 els**

**20 -For 1=0 to 7 30 read 40 print R$,**

1. **next I**

**100 data 12,24,1955,Smith 110 data 3,7,1942,Jones**

|  |  |
| --- | --- |
| DEL |  |
| FORMAT: | DEL [< line number 1 > ] [, < line number 2 > ] (CR |:} |
| FUNCTION : | DELetes a line or lines from the current program in memory. |
| Comments: | Be careful when entering two line numbers not to delete entire blocks of code from a program. |
| Example: | 1 0 0 r 0 nr\* ft 8 C D E 1 1 8 r e m t 9 h i ■j 1 2 0 r e m k 1 rn n o 1 3 8 r e m P r s t 14 0 v e m IJ U W X V z  R 0 a d y  del 110,130  R 0 a d y list  i 0 0 r 0 rn ft 8 C D E 1 4 0 r e rn UUI..JXV z  R 0 a d y |

DIM <array name> (<array size>[,...]) [,...] {CR |:}

DIM allocates memory for an array.

DIM

**FORMAT: FUNCTION : Comments:**

**Example:**

A DIM must be executed before an array is used. Each DIM statement must only be executed once. It is not legal to change the DIMensions of an array after it has been declared. An array can use up a large amount of memory, so care should be taken to allocate only as much as is needed. (Max. 255 dimension is possible, it does not in-

|  |  |  |
| --- | --- | --- |
| elude 0 dimension) | |  |
| i 0 0 | d i m O < 5 > | in  a: |
| 1 1 0 | i~4  II  hH  0 | t. O 5 |
| 120 | ft < I > = | 1+64 |
| 130 | 0 $ < I) | = c h r $ < fi ( I) |
| 140 | n e x t. I |  |
| 150 | t o r J = 1 | t o 5 |
| 160 | P r i n t | 0 < J > 7 fl $ ( J |
| 170 | n e x t |  |
| 1 8 8 | e n d |  |
| r u ri |  |  |
| 6 5 | 0 |  |
| 6 6 | g |  |
| 6 7 | c |  |
| 6 8 | D |  |
| 6 9 | E |  |

**R** e a **d y**

DRAW

FORMAT : DRAW <GR-coordinates>[,<GR-coordinates>] {CR |:}

FUNCTION : DRAWS a line on the screen.

Comments : If only one pair of coordinates are given then the line will be drawn from the current graphics-cursor position, and the graphics-cursor becomes the end of the line. If two pairs of coordinates are given they specify a line and the graphics-cursor does not change position.

Example:

|  |  |  |
| --- | --- | --- |
| >"1 Pi | P r i n t | "W3W“ s 9 in it |
| 1 0 | d r a w | 1 Pi Pi , 5 0 , i 0 0 , 1 0 |
| 2 9 | 9m o m e | 50?100 |
| 3 0 | d r a w | i 9 0 ?15 © |
| 4 0 | d r a w | 2 0 0 ? 1 3 0 ■ 1 0 0 ? 0 |
| 5 0 | P r i n t | "W |

END [CR |:}

Indicates the end of a program and halts execution.

END

**FORMAT: FUNCTION : Comments:**

**Example:**

Once the END instruction is executed, execution cannot be resumed with CONT. This instruction is not strictly necessary. (All of the channel will be closed)

1. 8 0 0 P r i n t " This is a u e r y s h o r t. P r o '3 r a rri " i 010 a nd

r u n

This is a v e r y s h o r t P r o 9 r a m

R e a d y

EVENT

FORMAT : EVENT < interrupt-interval > [, < delay-time >] {CR |:}

FUNCTION : Sets the interrupt interval accessed by the ON EVENT GOSUB statement.

Comments : The interrupt-interval is the interval between consecutive event timer interrupts that can take on values between 0 and 255 (number of 1/60 second units). Be careful since 0 is assumed to be 256 time units. The delay-time is the delay time until the first event timer interrupt and can take on values from 0 to 32767 (1/60 second units). 0 is assumed to be 32768. Unless otherwise specified, the first event timer interrupt occurs immediately after setting the event timer interrupt-interval. If negative values are specified, the event timer will stop.

Example: 10 els

1. 0 event 6 0 > 6 0
2. 0 on **event 9 osub $EU** 40 event on

**50 1=0:C=0 60 1=1+1**

**7 0 Print cursor ( 15? 3) ; 11 I = " ? I ;**

80 goto 60 **90 $EU 100 C=C+1**

**110 Print cursor(15? 13) " C = " J C 5**

**120 return**

EVENT (ON/OFFj

FORMAT : EVENT (ON/OFF) (OR |:}

FUNCTION : Enables/Disables event timer interrupt.

Comments: <ON/OFF>

ON = Allows calling of subroutine by an event timer interrupt set up by an ON EVENT GOSUB.. statement

OFF = Disables event timer interrupts

Example:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 0 | !~ | 0 n | | s 0 | 1 | 0 1 j y | , 0 ? 2 |  |
| l | i | 0 | c |  | ■r |  |  |  |  |  |
| ■i  i | z | 0 | 0 | y | 0 | n t |  | 6 0 ? 6 | 0 |  |
| l | -y | 0 | 0 n | |  | 0 M | 0 | n t 9 | 0 s u h | $ S U B |
| i | 4 0 | | 0 | M | 0 | n t |  | 0 n |  |  |
| 1 | 5 | 0 | H | = | 0 | » T  3 J. | = | 0 |  |  |
| ■» | 6 | 0 | t | fi | h | D |  |  |  |  |
| l | —7 | 0 | r | 0 | P | 0 a | t |  |  |  |
| i |  | 0 |  |  |  | H = | H | + 1 |  |  |
| 1 | 9 | 0 |  |  |  | P r | i | n t c | u r s 0 r C 10? 1 0 > “ 11 H = | |
| Z | 0 | 0 |  |  |  | f\$ | = | i n k 0 | y $ |  |
| z | 1 | 0 | u n t | | | i 1 |  | H $ < > | :! ii |  |
| z | z | 0 | 0 | n d | |  |  |  |  |  |
| z | T | 0 | $ | i~ | u | 3 |  |  |  |  |
| z | 4 0 | | 1 | = | 1 | 4. 1 |  |  |  |  |
| jlL | 5 | 0 | P | r | i n t- | |  | c u r s | 0 r ( 1 | !~j 4 “T \ a il \_ If a T  kl 7 x .\_|t J 7 l\_ — 7 j. |
| z | 6 | 0 | r | 0 | +  i.- | u r | n |  |  |  |

FCOL

FORMAT: FCOL [< color code >] {CR

|  |  |
| --- | --- |
| FUNCTION : | In Text Mode, sets the color of the characters.  In G2 mode or Multi-color mode, sets the color of the graphics display. |
| Comments: | The default value for the color code is 14, grey. If the background color (BCOL) matches FCOL then characters will be invisible. See Appendix B for color codes. |
| Example: | 10J FCOL TEST 2 0 tor 1=8 to 14 30 fcol I 40 sleep 1 5 0 n e x t. I |

.TOJSTEP]

FOR < control variable > = < initial value> TO < final value> [STEP <step value >] {CR |:j

FOR.

**FORMAT: FUNCTION :**

**Comments:**

FOR NEXT loops are used to perform many iterations of a section of the program. The control variable is set to the initial value and control transfers to the next statement. When a NEXT statement is reached with the same control variable then the control variable is incremented by the step value. When the control variable reaches or exceeds the final value then control is transferred to the statement immediately after the NEXT statement. Otherwise the statements between the FOR and the NEXT are executed once more.

The number of times that the program will iterate the commands in the loop is calculated as:

iterations = (final value-initial value)/step value

If the step value is not specified then it is given the default value of 1. The control variable should not be altered by the instructions in the loop. This is very dangerous and is poor programming style. When several loops occur one within the other they can take a long time to execute. Care should be taken to realize this when writing programs. Programs may contain complete loops within other loops but they may not overlap. Thus for each FOR statement encountered a NEXT must occur in the reverse sequence.

**Example:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 0 0 | c 1 S |  |  |  |
| 1 1 0 | t o r | T —  I ~ | t. m 2 7 ■- | t 0 P |
| 120 |  | o r | J = 1 t. o 9 |  |
| 130 |  |  | I o c a t- e I ? | J |
| 1 4 0 |  |  | P r i n t r i g | h t $ |
| 150 | n | 0 X | t J |  |
| 1 6 0 | n e x t | i |  |  |
| 1 7 0 | P r i n | t |  |  |
| 1 8 0 | 0 n d |  |  |  |

Note the use of indentation for readability.

GCOPY

FORMAT : GCOPY [< format type>] {CR |:}

FUNCTION : Prints the current screen image on the printer.

Comments : The default format type is 0. Formats are defined as follows?

1. = 40 character image format
2. = 80 character image format

Before this command is executed, it is necessary to issue the GMODE command.

**Example:**

|  |  |  |
| --- | --- | --- |
| 1 M H ? | 9 c o p y T | ESI |
| 1 I 0 | P r i n t " | ii |
| 1 2 0 | 9 in i t |  |
| 1 3 0 | 9 m o d s 4 |  |
| 1^0 | E R A S E |  |
| 1 5 0 | t C O 1 & 0 0 0 h | |
| 1 6 0 | GV 1=30; | G V2 = 5:H = 0 |
| 1 7 0 | P I ot 2 2 | 8 ? 0 |
| 1 8 0 | t o r T H = | 0 t o 6 + 1 8 0 s t 0 y |
| 1 9 0 | H = H+ 1 |  |
| 2 8 0 | X = c o s < 8 | 0 - H / 2 , T H > |
| 2 1 0 | G V = G V 1 + 1 | |
| 2 2 0 | G X=12 8 + | X |
| 2 3 8 | G V 2 = G V 2 | +1 |
| 2 4 0 | d r a w G X | ? G V 1 |
| 25 0 | 9 ffi 0 M 0 1 | 2 8 , G V 1 |
| 2 6 0 | d r a w 6 X | 7 G V 1 |
| 27 0 | n e x t- T H |  |
| 2 8 0 | P r i n t # 2 |  |
| 2 9 0 |  |  |
| 3 0 8 | 9 m o d 0 0 |  |
| 3 2 0 | t o r G = 0 | t o 1 |
| 3 3 0 | P r i n t i 1 | ? c u r s o r < 1 ? 2 1 > " i! G C P V H \* G " |
| 34 0 | 9 c o p y G |  |
| 3 5 8 | P rint#2 | Si ii |
| 3 6 0 | n 0 x t G |  |
| 37 0 | P r i n t " | ii 3  n |
| 3 8 0 | 0 n d |  |

GET [# <CH>][[,]<variable> [,..]] |CR |:}

**Comments: Example:**

Reads data from a designated channel to a designated variable.

It is desirable that data obtained with GET be data created with PUT. With character variables it is necessary to use LEN before GET to control the size of the variable.

g et #3 C0DE”■, NfiMEf » TEL$

GINIT [CR |:}

GINIT

**FORMAT: FUNCTION :**

**Comments:**

Enter the graphics mode (applicable to the multicolor and . Gil modes). The specified screen-clear-specifier is used for graphics (default is 255).

<screen-clear-request>

1. = display character, font clear colors, graphics

cursor and initialize graphics mode

1. = display characters
2. = font clear

In the multi-color mode, values greater than 1 clear the font and display characters.

If the screen is not cleared, go ahead and arrange characters on the displayed screen.

**Example:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0  1 | 1 | P  c | r  0 | i  n | n | t  Q 1 | H | jus  . ? w ? | ft | •3 i n , 1 | i | t |
| ~l | 8 | d | r | a | w | 1 | 0 | ; 1 0 ; | 2 | 00 , | 1 | 8 0 |
| j | 8 | a | i | n | i | t |  |  |  |  |  |
| 4 | 8 | t | c | o | 1 | 8 |  |  |  |  |  |  |
| 5 | 0 | d | r | a | w | 8 | T | 19 1, |  | 55, | 0 |  |
| b | 8 | 0 | n | d |  |  |  |  |  |  |  |  |

FUNCTION : Sets up the graphics display mode.

Comments : Affects the PLOT, DRAW, and PAINT statements. It takes the color or pixel already on the screen and the newly specified color or pixel, and then applies one of the functions below to decide the resulting color or pixel, e.g. a boolean operation on the specified color code of a pixel ORed with the existing color code of that pixel yields the new pixel condition in GMODE 1.

< mode 1 > • Color

0

1

2

3

Pixel

4

5

1. 7

= replace = OR = AND = old display

= replace = AND = XOR = old display

The following relationship holds for the “GRP” device.

< mode 1 > Color

0

replace

OR

AND

old display

OR

AND

XOR

Old display

1

2

3

Image

4

5

1. 7

When you specify “1” in mode 2, only the image will be processed.

**Example**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 0 |  | o n | s | O 1 0 | ? | 7 | 0 | ;t | 0 |  | i |  |  |
| 1 | 10 | g | i n | i | t |  |  |  |  |  |  |  |  |  |
| 1  i | 2 0 | t | c 0 | 1 | 3 |  |  |  |  |  |  |  |  |  |
| 1 | 3 0 | d | r a | w | 1 0 0 | | 1 | 0 | 3 | i 0 0, | | | 1 0 0 ; 0 , | 1 0 0 |
| <4  i | 4 0 | p | a i | n | t 5 | 0 | ? | 5 | 0 |  |  |  |  |  |
| i | 5 0 | t | c 0 | 1 | T |  |  |  |  |  |  |  |  |  |
| 1 | 6 0 | d | r a | i.,.i | 55 | 7 | 0 | 3  1 | 5 | 5 | j | 5 0 | ; 0,5 0 \* | 0 j 0 |
| 1  J. | 7 0 | g | m o | d | e 1 |  |  |  |  |  |  |  |  |  |
| 1 | 8 0 | i | c 0 | 1 | !j |  |  |  |  |  |  |  |  |  |
| 4 | 9 0 | r | a i | n | t 2 | 5 | 7 |  | o | n | "T |  |  |  |
| v | 0 0 | p | r i | n | t c | h | i\*- | ■$ | r' |  | 6 | > |  |  |
| •“« | 1 0 | 0 | n d |  |  |  |  |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| FUNCTION : | Moves the graphics cursor to the desired coordinates. |
| Comments: | No line is drawn with this command; it is only used to position the graphics cursor. |
| Example: | X 0 ■ i\_i (\_! | i 0 i yl i X  1 1 0 9 i n i t 12 0 t co1 B  1 3 0 d r a w 1 0 0,0 ; 1 0 0 , l 0 0 ; 0 , 1 0 0 ; 0 : 140 fco1 15 1 5-0 9 rn o ■...' s 2 0 0 , 1 0 0  1 6 9 d ra w 1 0 0 |0'1 0 0 1 0 0 j 0 j 1 0 0 " 0 : 1 7 0 P r i n t c h r $< 2 6)  1 8 0 e n d |

GOSUB

FORMAT: GOSUB <destination [CR |:j

FUNCTION : Transfers control to a subroutine so that control will be returned BACK to the point IMMEDIATELY AFTER the GOSUB.

Comments : Use the RETURN statement to transfer control back to

the point where the subroutine was called. Notice that this allows the subroutine to be used from many places in the program. GOSUB should be used in place of GOTO whenever possible. The BASIC GOSUB has recursive cabability: that is, a subroutine may GOSUB to itself, but as BASIC has no parameter passing abilities, all variables must be managed by the programmer. Destination = (line number | label name|numeric variable (line number)|string variable (label name)}

**Example:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 |  | P | r | i | n | t |  | it | IM=H | | | 1" |  |  |  |
| jL | 0 |  | 9 | o | s | u | b |  | $ |  | T | c | HR |  |  |  |
| -j | 0 |  | 9 | o | 3 | u | b |  | $ | D | P | ■ | SE | T |  |  |
| 4 | 0 |  | 0 | n | d |  |  |  |  |  |  |  |  |  |  |  |
| 1 | 0 | 0 |  | $ | b | T | c | H | R |  |  |  |  |  |  |  |
| 1 | 1 | 0 |  | -5 | t | c | h | r |  | fi | 0 | 0 | 13 | 3 | c 6 | 6 |
| 1 | “I | 0 |  | r | 0 | t | u | r | n |  |  |  |  |  |  |  |
| •? | 0 | 0 |  | $ | s | p | . | b | E | T |  |  |  |  |  |  |
| •~i  jL | 1 | 0 |  | s | c | o | d |  | 0 | ? | & | 7 | F: |  | CO | 1 |
| M- |  | 0 |  | 1 | o | c |  | 0 |  | t | o |  | 12 | 3 | ? 9 | 6 |
| 9 |  | 0 |  | r | 0 | t | u | r | n |  |  |  |  |  |  |  |

b?e2400" to &7F ? 0 0,5

GOTO

FORMAT : GOTO < destination > {CR |:J

FUNCTION : GOTO transfers program control from the current line to the line number or label in the statement.

Comments : GOTO statements should be used as little as possible. Although this is difficult in BASIC their use can be limited. To make programs clear, write programs in blocks, with little use of GOTO within blocks. A statement appearing after a GOTO on the same line (with a : separator) will never be executed. Destination = {line number | label name | numeric variable (line number) | string variable (label name)}

10 0 fi =1 00 1 1 0 9 O t O 1 0 0 0 120 8 = PI+ 5 1 3 0 9 o t o $ D I S P

**Example:**

1. 0 0 0 B = Pi 5 1 0 1 0 ■$ D I S P

1 0 2 0 P r i n t "a = " ; PI; " b = 1030 end

IF..THEN..ELSE

FORMAT : IF <conditional expression> THEN <statement>

[ELSE < statement > [:..]] [CR |:}

FUNCTION : Evaluates the conditional expression. If it is true (the condition is satisfied) then the statements(s) after the THEN are executed. If it is false (the condition is not satisfied) then the statements after the ELSE are executed.

Comments:

If there is no ELSE part and the condition is not satisfied then the statement has no effect. Conditional expressions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| are : | = , < | >, | if  A  a' | < |  |
| 1 9 0 | i n P | u t | " 1 0 | + | i 5 = ?"; fl |
| 1 1 0 | i i | ft = | 2 5 t- h | e n | P r i n t " 9 o o d " |
| 120 | e 1 s e n d | 0 | Pr in t | ii | a9 a in":go t o 109 |

Example:

INPUT

FORMAT : INPUT [<prompt> | # <channel number>] (, |;)

< variable > [,..]{CR|:j

FUNCTION : INPUT accepts alphanumeric data from the keyboard and assigns their values to variables.

Comments : The INPUT statement will place a question mark on the screen if the program does not include the < prompt > field. If multiple inputs are to be entered at one time then the variable names should be separated with commas. When multiple entries are to be made by the user running the program, then the inputs must be separated by commas. This format results in a problem inherent in BASIC: when the user is requested to input a string of alphabetic characters into a string variable, the string should not contain any commas. The only way to overcome this shortcoming is to write input routines using the INKEY function. For programs to be used by inexperienced users this may' be preferable to INPUT, as ‘bomb-proof input routines can be created. A ‘bomb-proof input routine cannot cause a BASIC error, no matter what the user types in. With the INPUT statement, if the user enters the wrong number of parameters, or enters a comma in a string, BASIC will print an error message. These may not be understood by inexperienced users.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ■ | 1 0 0 | i n P u t | " lil h a t | i s y 0 u r a '3 0 ? " | ; A G |
|  | 1 1 0 | i n P u t- | " I...I h a t- | i s y 0 u r n a rn 0 ? | "; N |
|  | 120 | Prin t. | flQE; " | is a '3 0 0 d a ‘3 0 | "; N |
|  | 1 3 0 | 0 n d |  |  |  |

Example:

r u n

Wh at is your a90 ? 4

W h at is yo ur n am e ? Rona 1 d 4 is 9 ood a 9 0 R on a 1d

**ll! a:**

R e a d y

|  |  |
| --- | --- |
| KILL  FORMAT: | KILL <file name> {CR |:} |
| FUNCTION : | Deletes specified files. |
| Comments: | Effective only with an external disk drive. |
| Example: | kill " FXfi 0:0 L D DAT R" |

|  |  |
| --- | --- |
| LEN  FORMAT: | LEN < character string length > {CR |:j |
| FUNCTION : | Resets the maximum length of string variables. |
| Comments: | The default value for string lengths is 18 characters. LEN can be used to set the value from 1 to 255. |
| Example: | 1 0 0 fi $ = " a b c d e t 9 h i J k 1 m n o P 9 r s t u "  1 1 0 Print- fl T  r u n  Err 15 in 10 0 R e a d y  10 0 1 an 24 : fl $ = " ab cdet 9 h i J k 1 mo P 9 r s t- u " 110 Print fl $  run  a b cdet 9 h i J k 1 rn n o P 9 r s t- u R e a d y |

|  |  |
| --- | --- |
| LET |  |
| FORMAT: | [LET] < variable >[,< variable >] [, ]= < expression >  (CR |:} |
| FUNCTION : | LET assigns the result of an expression to a variable. This result may then be referenced in other expressions. |
| Comments: | The word LET is optional. A variable name may be followed by the assignment operator ‘ = ’ and an expression. LET itself is a vestigial command. |
| Example: | 1 00 let fi = 5 110 B = 6 1 2 0 Pr int A>B |

|  |  |
| --- | --- |
| LIST |  |
| FORMAT: | LIST [<descriptor> | <file name>] [<line number-1 >] [, < line number2>] {CR |:} |
| FUNCTION : | Lists a file or portion of a file to another file, the printer, or on the screen. |
| Comments: | The LIST command can be used instead of the SAVE command to store files. When a file that was LISTed to disk or tape is subsequently read, the program in memory will not be erased, unlike a file that was SAVEd. However if the program being read has any line numbers identical to the memory program these lines will be replaced with the new lines. To distinguish these files from normal files, they are called “listing files.” |
| Example: | 1 i s t  10 0 fl = 1 : B = 2 1 1 0 P r i n t fl 1 2 S P r i n t 8 |

|  |  |
| --- | --- |
|  | R e a d y |

LOC < sprite-number> TO <GR-coordinates> {OR |:}

Moves sprite-number to the specified GR-coordinates.

LOC

**FORMAT: FUNCTION : Comments:**

**Example:**

Sprite-numbers are 0 to 31, 0 has highest priority. The highest priority number at that location will be displayed.

|  |  |  |
| --- | --- | --- |
| 10' | LOC TEST |  |
| 20 | Print " LIMB" |  |
| 3 0 | stchr "0013 | 3 c 6 6 d b 7 e 2 4 8 0 " |
| 4 0 | scod 0 , &7F: | scol 0 ? 4 |
| 5 0 | ■for 1=0 to | 255 |
| 6 0 | 1 o C 0 t. o | 1,90 |
| 7 0 | next I |  |

LOCATE

FORMAT: LOCATE <column>, <line> {CR |:}

FUNCTION : Moves the cursor to the specified line and column on the screen.

Comments : This statement is identical to CURSOR (X,Y) except that

because CURSOR is a function, it can be used in a PRINT statement and LOCATE cannot. The screen uses a 0 origin coordinate system. The legal range of each coordinate depends on the screen mode in use.

Example : 1 0 8 1 o c a te **9.11**

1. 1 0 Print, " B o 11 o m c o r n e r

MAG

FORMAT : MAG [< sprite-modifier >] {CR |:}

FUNCTION : Change the sprite size and format.

Comments : < sprite-modifier >

1. = 8x8 dot matrix
2. = 8x 8 dot matrix (by 2)
3. = 16x 16 dot matrix
4. = 16x 16 dot matrix (by 2)

Example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i 1 0 | P rint |  |  |  |
| i 2 0 | s t- c h r | " 0 0 1 8 3 | c 6 6 d b 7 e 2 4 0 0 " | t o & 7 F j 0 |
| 139 | S C 0 ij 0 | J &7F |  |  |
| 1 4 0 | SCO] 0 | , 4 |  |  |
| 150 | m a 9 1 |  |  |  |
| 1 6 | 1 O C 0 | t 0 1 0 0 | , 1 0 0 |  |
| 1 7 0 | e n d |  |  |  |

MAGO

MAG 1

MAG 2

MAG 3

**Example of display  
for ‘A’, or & 41**

**Character dot matrix**

**Enlarged dot matrix**

**Notes**

N

8x8

dots

8x8 dots

* Default mode when power is first supplied
* Sharp picture



16x16

dots

16x16

dots

16x 16 dots 16x 16 dots

One large picture can be created by combining four smaller pictures • Hazy picture

a

H

B

C

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 1 | 1 | 3 |
| PI | | ( | n\* |

8x8

dots

16x16

dots

1 Large sprite can be easily created by combining four characters 1 Sharp picture

16x16

dots

32 x 32 dots

One large picture can be created by combining four smaller pictures Hazy picture

NEW

FORMAT: NEW (CR

|  |  |
| --- | --- |
| FUNCTION : | Clears the current program and memory contents. |
| Comments: | This command prepares the M5 for beginning a new program. MAKE SURE that the current program has been saved on disk or tape before issuing a NEW command. |
| Example: | 1. 0 r e m P r o 9 r a m 1 2. 0 d i m R < 1 0 0} 3. 0 9 o s u b $ I N I T |
|  | R e a d y  n e w  R a a d y 1 i s t  R e a d y |

NEXT

FORMAT : NEXT [<control variable >] (OR

|  |  |
| --- | --- |
| FUNCTION : | NEXT ends a section of a program started by a FOR statement which is to be executed repeatedly. The control variable indicates which FOR statement this NEXT matches. |
| Comments: | NEXT should always be used with explicit control variables as leaving them out makes the program very confusing. All FOR NEXT loops should be indented two or three spaces to show the extent of each loop graphically. Further, each loop should be a maximum of one screen or one page in length. Longer loops should call subroutines. |
| Example: | 10 0 t or 1 = 0 t o 21 step 3 i i y v i n t\* i t 2 0 n e x t I |

|  |  |
| --- | --- |
| OLD  FORMAT: | OLD [<file-name>] (CR |:j |
| FUNCTION : | Reads a file from external storage. |
| Comments: | When the file-name is omitted, the first file found is read into memory. |
| Example: | old " CMTiPROG 1 " |

FUNCTION : Transfers control to the line number when any BASIC-F error is detected during program execution.

Comments : This command allows the program to continue to run

when an error occurs. The routine should display an error message which will explain the error that the user made. The program should then allow the user to correct the mistake. This will only work with run time errors, not program syntax errors. Destination = {line number | label name | numeric variable (line number) | string variable (label name)}

Example:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | o | n | e | r | r | or |  | •3 | o |  | u | b | $ER | R |
| -X | 0 | i | n P | u | t |  | " I | N | P | U | T |  | N U M B E | | R" \* fi |
| ■T  .J | 0 |  | P r | i | n | t | fi |  |  |  |  |  |  |  |  |
| 4 | 0 | $ | ER | R |  |  |  |  |  |  |  |  |  |  |  |
| 5 | 0 | i | t | 0 | r | r | = 2 | 5 |  | t | h | e | n | r es | ume 20 |

ON EVENT GOSUB



FORMAT : ‘ON EVENT GOSUB <destination (CR |:}

FUNCTION : Calls subroutine beginning at line number when event timer interrupts—interrupt priority 2. (The event timer is initialized with the EVENT statement.)

Comments : Destination = {line number | label name | numeric variable (line number)| string variable (label name)}

Example " i y 0 $ u 0 n t\* S 0 ? y

1. 1 0 o n . 0 y e n t 9 0 s u b E U
2. 2 0 0 m 0 n t o n 1 3 0 9 o t o 1 3 0 140’

15 0 $E U 1 6 0 Prin t- c h r $ < 7 ) ;

17 0 r 01 u rn

This is a multiway branch instruction. After evaluating ‘expression, branches to the nth line number in the line number list. On executing a RETURN, returns to the next statement after this ‘ON GOSUB.’

This type of instruction is also known as a CASE or SELECT statement. It allows one of many options to take place from one point in the program. Be sure that the range of results for the expression is limited to the number of line numbers in the line number list. It is suggested that labels be used for all line numbers. Destination = (line number | label name | numeric variable (line number) | string variable (label name))

**Comments:**

**Example:**

i kJ

i n P u t " E

< 1 - 5 > M ; c

■j n ij \*3 o s

n t e r t h e c o rn m a n d n u rn fa e r

i

u fa $C 0 HI,fC0M2 ?$C 0M 3

1. 0 0 0 $ C 0 M 1
2. 0 0 0 $ C 0 M 2
3. 0 0 0 $ C 0 M 3

GOTO.

ON <expression> GOTO destination> [CR |:}

This is a multiway branch instruction. After evaluating the expression, branch to the nth line number in the line number list.

For using subroutines the ON GOSUB statement will be more convenient. Destination = [line number | label name | numeric variable (line number) | string variable (label name)}

**Comments: Example:**

1 90 i n P ut " No , < 1 3> '? " ; CPISE\*

110 it 0ftSE<1 or CR S E>3 then oto 18 12 0 on CASE 9o to $WHEN 1? $WHENS?$W H £ N 1 0 0 8 $I..J HEN

1. 8 8 8 $ !.ij HENS
2. 0 8 8 $ W H h N 3

RESTORE.

ON <expression> RESTORE <destination> {OR |:}

Sets the data pointer to one of several data groups depending on expression. After evaluating the expression sets the data pointer to the data statements after the nth line number in the line number list.

Use labels instead of line numbers for each of the data groups. See the DATA and RESTORE statements. The expression must result in a number less than or equal to the number of labels in the line number list. Destination = (line number | label name | numeric variable (line number) | string variable (label name)}

**Comments: Example:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 0 | i n P | IJ | t | si | W h a t | c h a r a c t e r | 9 r o u P |
| 1 | 1 0 | i i | T |  | i  JL | o r T > | 2 t h e n '3 o 1 | o 1 8 0 |
| 4 | 2 0 | g n | l |  | r 0 | s t o r 0 | $ S ? $ L |  |
| i | 3 0 | t o r |  | T | \_ 4  ~ i | t o 5 |  |  |
| i  i | 4 0 |  | r | 0 | a d |  |  |  |
| 4  i | 5 0 |  | P | r | i n | t hi $ 5 |  |  |
| 1 | 6 0 | n e x | t |  | t |  |  |  |
| 4 | 7 0 | e n d |  |  |  |  |  |  |
| 1 | 8 0 $ | x" |  |  |  |  |  |  |
| 1 | 9 0 | d a t | a |  | a ? | q ? C ? d | ? 0 |  |
|  | 0 0 $ L | |  |  |  |  |  |  |
| V | i 0 | d a t | a |  | hi ? | 8 , C ? D | , E |  |

OPEN

FORMAT : OPEN <file name> [for [input | output | append)] as#

<CH> [Record < record length >]{CR |:)

FUNCTION : Opens user files, for further usage.

Comments : File names must have no more than 9 characters.

Input is used in conjunction with the INPUT # command, and is reading data from a file. Output is used in conjuction with the PRIN # command, and is for writing data to a file.

Append is used like output, but only when you are adding data tc

**Example:**

|  |  |  |
| --- | --- | --- |
| already existing | | file. |
| 1 0 0 | o P 0 n | " P R T |
| i l 0 | P r i n t | #2 " |
| 120 | C 1 O S 0 | #2 |
| 130 | e n d |  |

PAINT

FORMAT : PAINT <GR-coordinates> [, <boundary-color> [,..]]

ICR |:}

FUNCTION : Paints an area delimited by the GR-coordinates using one of up to 16 colors indicated by boundary-color.

Even if the boundary-color is omitted, the appropriate area will not be colored transparent (invisible).

**Comments:**

Example:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ■» | 0 0 |  | o n | s o : e ? | , 0 | ? 0 7 1 |
| 1 | i 0 | r | 1 5 |  |  |  |
| i | 2 0 | g | i n | i t |  |  |
| 1 |  | f | o r | 1 = 1 | t O | 1 0 |
| i | 4 0 |  |  | X = r n | d < | 2 3 0': > |
| l | 5 0 |  |  | V X - r n | d < | 18 0 \*: > |
| 1 | 6 0 |  |  | i.ij X X = r | n d | ( 5 0 1 0 ’/. |
| 1 | 7 0 |  |  | i.ij v — r | Fs d | < 5 0 ■; > + 1 8 |
| i | 3 0 |  |  | C 0 L "\*« = | r n | d < i 3 \*/. > + 2 y. |
| i | 9 0 |  |  | t C 0 1 | C 0 | L |
| Z | 0 0 |  |  | g m o y e | V | ■ 1 t  •\*B ? 1 |
| Z | 1 0 |  |  | o r a w | i,,1 a,.  f\*\*"i ‘i | + i.j x y. ^ v %' x + u x y. 7 v % + y v \*•:; |
|  |  |  |  | i i ■/ •l ? i | + i.ij | M a V V U V T ? r’« ? i •'\* |
| Z | 2 0 |  |  | t C 0 1 | r n | d < 13 y. +2 y. |
| z | 3 0 |  |  | P a i n t | v | y. + y x y. / 2 y., v ■; + y v y. / 2 , c o L y. |
| z | 4 0 | n | ex | t i |  |  |
| z | 5 0 | c | o n | sol e ? | 7 7 | i |

PLOT

FORMAT : PLOT <GR-coordinates > [;„] [CR |:}

FUNCTION : Displays the dot associated with the coordinates. Use the color set up by a FCOL statement.

Comments : After execution, the graphics cursor will reside at these coordinates.

Example • i y y c n n **sole.’ ? y ? y ?** 1 1 1 0 C 1 S 1 2 0 9 init

13 8 io r ' I = 1 t o 3 0 0

15 0 f c o 1 r n d < 1 3 X) + 2X

1 60 P1ot r nd<255X> » rnd< 1 9 1 X>

**1 3 0 n e x t I**

190 con so 1e j ? ? 1

|  |  |
| --- | --- |
| FUNCTION : | POKE writes data directly into specified locations in the computer memory. |
| Comments: | The data may also be an expression to be evaluated before the rest of the statement. Care should be taken with memory addresses and data contents. Certain locations will destroy the current program in memory. Programs written for other computers will not work on the M5 if they have POKE (or PEEK) statements without special modifications to these statements. |
| Example: | 1 0 0 P o k e & F 8 0 0 ?S F E |

|  |  |
| --- | --- |
| FUNCTION : | Writes the data to the specified memory address in CPU memory. |
| Comments: | The data must be numeric. The lower-order byte is written in the specified address while the upper-order byte is written in address +1. |
| Example: | 1 0 0 C ’ S  110 B = & F F F F  12 0 P X = yar Ptr< 8 X)  1 3 0 p o k e w h , & 1 0 0 0 1 4 y pi\*- in t- h e x T \*\*. y j 150 end |

PRINT

FORMAT : PRINT [# <channel number>] <expression>,... {CR

|  |  |
| --- | --- |
| FUNCTION : | Puts text in the screen display buffers. |
| Comments: | PRINT is also used to send control characters which may not appear as characters but which have important effects on the screen buffers. Most characters sent by PRINT to the screen buffers are merely deposited and appear on the screen. Control .characters, on the other hand, may clear the contents of a buffer, change which buffer will be displayed on the screen, erase a character from the buffer, or cause future PRINT commands to send characters to one or the other buffer. It is important to realize that the PRINT command can put characters into the buffer that is not currently visible. |
| Example: | 1 8 8 P r i n t " G o o d d a y , e h ? "  r u n  •j o o d d a y a h Y R e a d y |

|  |  |
| --- | --- |
| PUT  FORMAT: | PUT [# <CH>] [ [,] <expression> [,..]] {CR |:} |
| FUNCTION : | Assigns a binary form to the value of an expression and outputs it. |
| Comments: | Binary form is integral in BASIC; the chief purpose of this command is to create records of fixed length. It ensures that when output, characters will be of constant length. |
| Example: | p u t #3 C0DE ? NfiM E $. > TEL$ |

RANDOMIZE

FORMAT : RANDOMIZE [CR |:}

FUNCTION : Resets the seed for the random number generator.

Comments : This affects RND

Example:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | r | a | n | d | 0 | m | i | ze |
| 20 | i | o | r |  | I | = | 0 | t |
| 30 |  | R | = | r | n | d | < | 1> |
| 40 |  | P | r | i | n | t |  | R |
| 50 | n | 0 | X | t |  | I |  |  |
| 60 | 0 | n | d |  |  |  |  |  |

READ

FORMAT : READ < variable list> {CR |:j

FUNCTION : READ loads data from DATA statements into variables.

Comments : When a READ statement is issued, the variable gets the value of the data item at the current value of the data pointer. The data pointer is maintained by BASIC-F. When you type RUN the data pointer is set to the first data item in the first DATA statement wherever it appears in your program. After the first READ you must exercise care in using READs, to ensure that you are aware of the position of the data pointer. When modifying the program always ensure that the order of the DATA statements is preserved. Use the RESTORE command with labels, to ensure correct positioning of the data pointer.

Example:

|  |  |  |
| --- | --- | --- |
| 1 0 0 | r 0 a d | X ? V $ |
| i i 0 | P r i n t |  |
| 120 | 0 n d |  |
| 1 3 0 | d a t a | 2 0 0 |
| r u n | | |
| 2 0 0 | j Q | h n |

R e a d y

RECORD

FORMAT : RECORD # <CH>] [,] < Record number > {OR

|  |  |
| --- | --- |
| FUNCTION : | Specifies the record to be accessed next, and the execution order of reading and writing. |
| Comments: | A record is a collection of files of uniform length. The record number may range from 0 to 65535, and the length is specified by OPEN. Random access is available. Normal execution order is write then read. |
| Example: | r e c o r d # 3 j C D |

|  |  |
| --- | --- |
| REM  FORMAT: | {rem |! | ’} < comments > |
| FUNCTION : | Stores programmer comments within the program |
| Comments: | REM is short for ‘remark’. Remarks are not executed by the computer. It is recommended that remarks be placed liberally throughout all programs to help when debugging programs. Unfortunately REM statements can use up a lot of memory. A trade-off must be made between readability and ease of modification and debugging. No statements may occur after a REM in BASIC-F. This is unlike most other BASICS. |
| Example: | 1 0 0 re m This P r o.9 r a m w a s w r it t e n M a r k U o u m a r d 110 ’  12 0 ’the -foil o w i n 9 sect i on i n i t i a t h e m a i n v a r i a b 1 e s 130 ’and opens files |

RENUM

FORMAT : RENUM [<new line number>][, <old line number>][,

<step>]

FUNCTION : Changes the line numbering of the program. Maintains all GOTO and GOSUB commands.

Comments : As a program grows in size, more lines of code may

need to be added than was anticipated. If room runs out, RENUM can be used to create more space and even out the line numbering.

fi , B , C = 1 for 1=1 to 10 ft = R + I B = B - I

**txampie:**

1 0 0 1 1 3 126 139 152 16 5 178 19 1

C = C \* I n e x t I Print fi ? B, C e n d

**Read y**

|  |  |
| --- | --- |
| r e n u m | 10 0 0, 1 0 0,20 |
| list |  |
| 1 0 0 0 | fi , B , C = I |
| 1 0 2 0 | f or 1 = 1 t o 10 0 |
| 1 0 4 0 | fi = fi + I |
| 1 0 6 0 | B = B - I |
| 1 0 8 0 | C = C \* I |
| 1110 | n e x t. I |
| 1120 | Print fi,B,C |
| 1140 | e n d |

Read y

**r**

REPEAT

FORMAT : REPEAT [CR |:}

FUNCTION : Sets up a loop with a logical test at the end. The end of the loop is specified with an UNTIL.

Comments : The loop is used much like a FOR NEXT loop where no counter is required. A REPEAT loop is always executed at least once! The test to end the loop is in the UNTIL statement at the END of the loop. When a REPEAT instruction is encountered, BASIC-F marks the line and continues execution of the next and succeeding statements. When an UNTIL statement is encountered, the logical test is evaluated (see UNTIL). If it is true the last REPEAT mark is removed and execution continues on the line following the UNTIL. If it is false, execution continues on the line following the last marked REPEAT statement.

Example:

|  |  |  |
| --- | --- | --- |
| 1 0 0 | o  II  ct |  |
| 1 1 0 | r eP | e .31 |
| 120 |  | H = ft + 1 |
| 130 |  | P r i n t |
| 1 4 0 | u n t- | i 1 fl = |
| 150 | en d |  |

M

|  |  |
| --- | --- |
| FUNCTION : | Resets the data pointer for a group of data items in DATA statements. |
| Comments: | This function is convenient when you need to extract several categories of information from the same group of data. Destination = {line number | label name | numeric variable (line number) | String variable (label name)} |
| Example: | 1 8 8 r 8 store $D R T fi 2 118 tor 1 = 1 t o 5 1 ^y r e a d N 5 ? H 13 8 Print $ R,fl 1 4 8 n e x t. I 15 8 0 n d 1 6 0 \* D A T A 1  17 0 data A = > 100,B = ? 13 8 ? C = ?300 1 8 0 $ D fl T fi 2  1 9 8 data Ll=? 2 0 5 ? E = ? 4o4? ~ = ? 554? ld=? 1 H = , 4 4 4 |

|  |  |
| --- | --- |
| FUNCTION : | Bypasses an error and begins execution from line number. |
| Comments: | When line-number is omitted, the next statement is executed. Destination = {line number | label name | numeric- variable (line number) | string variable (label name)) |
| Example: | i y y o n e r r o r 9 o s u b $ E H R 110 fl £ = " ":B = 6 5 120 H$ = fi•$ + ch r $ <: B>  130 Print fl$  1 4 0 9 o t- o 1 2 0  150 end  1 6 0 $E R R  17 0 A $ ="": Q = B+1  1 8 0 it B >9 0 then res u me 15 0  1 9 y res ume 17 y |

\_

RETURN

FORMAT: RETURN [ <destination > ] {CR

|  |  |
| --- | --- |
| FUNCTION : | Returns the program control to the point immediately after the point from where it was called by a GOSUB. |
| Comments: | A RETURN must not be executed if it has not been preceded by a GOSUB. Thus a subroutine which ends in a RETURN must always be called by a GOSUB. GOSUB and RETURN should be used frequently to create a modular program. Using the optional line number version of RETURN generally defeats the purpose of GOSUB. Destination = (line number | label name | numeric variable (line number) | string variable (label name)} |
| Example: | 1 6 0 9 o s u b f E X 4 M 1 1 0 9 o -■ u b $ D I b P L h V  N  N  1 0 0 0 $ E X H M  n  N  14 9 0 re tur n 1 5 0 8$D I S P L fiV |
|  | 1 3 2 0 r e t u r n 1 0 0 |

|  |  |
| --- | --- |
| RUN  FORMAT: | RUN [< destination > 1 <file name>] [CR |:} |
| FUNCTION : | Executes the current program. |
| Comments: | The optional line number can specify the starting point of the current execution. All variables are cleared when a RUN is executed. |
| Example: | r u n |

SAVE

FORMAT : SAVE < file-name > [, < first-address >, < last-address >

|  |  |
| --- | --- |
|  | [, <fitart-address>] ] [CR |:} |
| FUNCTION : | Writes to external memory. |
| Comments: | When a file name only is specified, or when an address is specified in a Basic-G program, this command will write from the CPU’s optional memory area to tape or disk. |
| Example: | s a v e " C M T : p R 0 6 1 " |

|  |  |
| --- | --- |
| FUNCTION : | Assigns character-code to sprite-number. |
| Comments: | Sprite numbers can be from 0 to 31 only. |
| Example: | 1 y y console? ? ? ? 2  110 CIS  1 2 8 s t c h r "9 0 1 8 3 c 6 6 d b 7 e 2 4 0 0 " t 13 0 s cod 0 ? &7 F 1 4 Gi sco 1 0 ,? 4 1 5 0 m a ‘3 1   1. 0 1 0C 0 t O 1 0 0 ? 100 2. 0 en d |

FUNCTION : Colors sprite-number using color code.

Comments : Only the character color can be changed, not the pixel background color.

**Example: 10’ SCOL TEST-**

**20 Print "tiHI"**

**■30 stchr " 00 1 83c66d b 7e2400 " to &7F,0 48 seed 0 » &7F**

1. **0 -for 1 = 1 to 14**
2. 0 Scot 0 j I

**70 1 o c 0 t o 1 2 3 ? 9 6**

**30 sleep 1**

**9 0 n e x t- I**

1. = tone generator channel 0
2. = tone generator channel 1

|  |  |
| --- | --- |
| SG |  |
| FORMAT: | SG <channel-number>,[{<frequency> | <noise>}] [, < volume >] {CR |:} |
| FUNCTION : | Turns the three tone generators and noise generator on and off as well as making them produce sound effects. |
| Frequency : | <channel-number> |

1. = tone generator channel 2
2. = noise generator

< frequency >

Frequency value varies from 1 to 1023; 1 is the highest frequency and 1023 is the lowest. (1024 is the default.)

< noise >

When the noise generator is used, channel 3,

0 — 7 specifies the type of noise. 0-3 are tone while 4-7 are variations of white noise.

Notice noises 3 and 7 are dependent on the frequency of channel 2 (even if channel 2 is not on).

|  |  |
| --- | --- |
| Noise | Frequency |
| 0 | N/512 |
| 1 | N/1024 |
| 2 | N/2048 |
| 3 | dependent on |
|  | channel 2 |
| 4 | N/512 |
| 5 | N/1024 |
| 6 | N/2048 |
| 7 | dependent on |
|  | channel 2 |

< volume >

Varies from 0-15 with 15 being the loudest.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 0 0 | t 0 | r | i = i t | !J | 1 0 | |  |
| i 1 0 | t | o r | J = 1 8 |  | t 0 | 45 | S t 0 p |
| 1 2 0 |  | S 9 | 1,30 |  | J X | 10 |  |
| 1 3 0 |  | S 9 | 2 ? J ? | 0 |  |  |  |
| 140 |  | S 9 | 7> —j  •j ? r 7 | J | / 3 |  |  |
| 1 5 0 |  | s 1 | 0 0 p 8 | 7 | 1 |  |  |
| 1 6 0 | n | 0 X | t J |  |  |  |  |
| 170 | t | o r | K = 4 5 |  | t- 0 | 18 | S t 0 P |
| 1 8 8 |  | s 9 | 1,30 | 7 | K / | 1 0 |  |
| 1 9 0 |  | S 9 | 2 , J, | 0 |  |  |  |
| 2 0 0 |  | S 9 | \*7 -n  •J ? I- ? | K | 71 |  |  |
| 2 1 0 |  | s 1 | e e p 5 | 7 | i" |  |  |
| 2 2 0 | n | 0 X | t K |  |  |  |  |
| 2 3 0 | s | g | 1 , , 0 : | s | •g | 3 7 7 0 |  |
| 2 4 0 | •r | 1 0 | eP 5, | 1 |  |  |  |

Example:

2 5 0 n e x t I 2 6 8 e n d

SLEEP

FORMAT : SLEEP < sleep-count > [, < base-time >] {CR 1:j

FUNCTION : Stops execution for the specified sleep time. BASIC event interrupts will be ignored. However, machine language interrupts will be processed.

Comments : Sleep time is (sec):

sleep-count x increment-time/60

If the increment-time is omitted, 60 is assumed.

Example : i @ 0 c 1 s

110 for '1 = 1 t. o 10 0

1. 0 Print. I >
2. 0 if I> 5 0 then sleep 30? 1
3. 0 n e x t. I
4. 0 e n d

STCHR

FORMAT : STCHR < pattern-code > TO < character-code >

[, <character-set-number>] {CR |:}

FUNCTION : Assign a pattern-code to character-code (usually a hexadecimal number associated with an ASCII-code) to determine a character’s shape.

Comments : A character can also be colored using the STCHR statement in the Gil mode.

< Character-set-number >

1. = sprites
2. = for character patterns in other than the Gil mode or

character patterns in the top third of a Gil screen

1. = for character patterns in the middle third of a Gil

screen

1. = for character patterns in the bottom third of a Gil

screen

1. = for color codes for characters in the top third of a Gil

screen

1. = for color codes for characters in the middle third of a

Gil screen

1. = for color codes for characters in the bottom third of a

Gil screen

1. = for character patterns in the entire screen
2. = for color codes for characters in the entire screen

Note: 2 to 8 correspond to the Gil mode. The character ‘A’ is constructed below: its pattern code is to the right (use hexadecimal notation for its pattern code).

Example:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 0 0 | con so 10 ? ?y? y? |  |  |
| 1 i 0 | c 1 s |  |  |
| 120 | s t c h r " 0 0 1 8 3 c | 6 6 d fa | 7 e 2 4 0 0 " t o |
|  | < " fi “ > , 1 |  |  |
| 130 | r 9 P e a t |  |  |
| 1 4 0 | ] o c a t 0 r n d | < 3 1 J£ | > ?r n d ( 2 3 V.) |
| 150 | P r i n t ” fi " |  |  |
| 1 6 0 | fi $ = i n k 0 y $ |  |  |
| 1 7 0 | u nt.il fi $ < > " " |  |  |
| 1 8 0 | 0 n d |  |  |

STEP

FORMAT : STEP (on | off) (CR

|  |  |
| --- | --- |
| FUNCTION : | STEP ON stops execution whenever the statements in the program change. STEP OFF resumes normal execution. |
| Comments: | The number of the statement to be executed after STEP ON is indicated as follows:  STOP AT 0000 where 0000 is the statement number. |
| Example: | s t 0 P o n  R e a d y  r u n  st oP AT 10 9 read V c o n t.  Stop AT 110 |

**STOP**

**FORMAT: FUNCTION : Comments:**

**Example:**

STOP {CR |:}

Halts execution of a program from within the program.

A program stopped by STOP can be restarted by CONT. This is useful for debugging programs.

|  |  |  |
| --- | --- | --- |
| 1 0 0 | P r i n t 11 a b c d | 0 f g |
| i i 0 | s t 0 P |  |
| 1 ~ {7-i | P •\*'" i n t “ h i J k | 1 m n |
| r u n |  |  |
| a h c | d e t 9 |  |
| S t o | P at 11 0 |  |
| R e d | d 9 |  |

SWAP

FORMAT: SWAP <variable>, <variable> {CR |:}

FUNCTION : Transfers the values or contents of designated variables.

Comments : Both variables must be of the same form.

Example: 10 **’Bubble sortin'?**

**20 on error 9o s u b E R 30 dim R < 1 0)**

**40 9 o s u b $ INPUT 50 for 1=0 to 10 60 for J=I to 10**

**70 if R<J><=R<I)then swap fl<J>,R(I>**

**30 next J**

90 next I

**100 9o s u b ^OUTPUT**

110 end

**120 $ INPUT**

**130 for K = 0 to 10**

**140 inPut fi < K >**

**150 next K**

**160 return**

**170 $ 0 U T P U T**

**130 for 0 P = 0 to 10**

**190 Print fl COP)**

**200 next OP** 210 end **220 $ER**

**230 if err=25 then resume 240 Print err?errl**

TAB

FORMAT:

TAB (X)

FUNCTION : Moves the cursor X character spaces right.

Comments : TAB should be followed by a in order to print material in particular columns on the screen.

Example : 1 0 0 p rint t a 0 < i 0 > ; " T h i:

1 !”1 11

•• u n

This is c o 1 u rn n 1 0

R e a d y

**TAPE**

**FORMAT: FUNCTION : Comments: Example:**

TAPE [CR |:}

Accesses the assembler supplied on tape.

For use with external data tape recorder only.

t y P e

This will read the machine language from tape into the M5 memory.

THETA

FORMAT: THETA (0| 1} (CR

|  |  |
| --- | --- |
| FUNCTION : | Sets the mode for trigonometric functions to degrees or radians. |
| Comments: | 0 is radian mode and the default. 1 is degree mode. |
| Example: | 1 0 0 t h e t a 0 110 pri nt sin<30> 120 theta 1 13 0 Pr in t si n <3 0) 1 4 0 e n d  r u n  - y „ 9 y 8 0 3 11 z 4 u 9 2 9 0 3 5 |
|  | R e a d y |

**TRACE**

FORMAT : TRACE {ON | OFF) {CR |:}

|  |  |
| --- | --- |
| FUNCTION : | Displays a trace of executed line numbers while the program is executing. |
| Comments: | TRACE allows the programmer to see which lines of the program are to be executed. This is useful only for debugging programs under development. |
| Example: | 10 0 tor 1 = 1 t o 2 110 ■ P r i n t- " h i "  12 0 P r int "d one" 1 3 0 n ex t I 1 4 0 P r i n t- " e n d "  15 0 e n d trace on r u n R e a d y |

TYPE (int | dbl | str) [CR |:}

**TYPE**

**FORMAT: FUNCTION : Comments:**

**Example:**

Defines the type of a variable.

In cases where the type designation of a constant or a variable has been omitted, the TYPE command designates the type given to the variable or constant. Character-type designation is effective only with variables.

t y P e s t r

Read y list.

10 A= " AB C "

1. 0 B ’: = 3 9 0 0 #
2. 0 C# = 12 . 3 4 5 #

R e a d y t y P e d b 1

r e a d y **list**

10 A $ = " A B C "

1. 0 R v = 3 0 0 0
2. 8 C = 1 2 . 3 4 5

**UNTIL**

FORMAT : UNTIL < conditional expression> (CR |:j

FUNCTION : Marks the end of a REPEAT loop.

Comments : The REPEAT loop is demarcated by a REPEAT at the beginning and an UNTIL at the end. All the instructions between them are executed REPEATedly until the expression in the UNTIL statement is TRUE. See REPEAT.

Example : 10 0 rep e a t

110 • X J£ = r n d < 1 0 \*•; >

1 2 0 p r i n t- X 1 3 0 u n t i 1 X X = 3 X

This loop will print random integers between 0 and 10 until it hits an 8.

**VERIFY**

FORMAT: VERIFY [< file-name >] [CR

|  |  |
| --- | --- |
| FUNCTION : | Compares a program in memory to another stored on cassette tape. |
| Comments: | When the file-name is omitted, the first file found on external memory is compared. |
| Example: | verify "CMT:PROG1" |

VIEW [<x0>, <y0>, <x1 >, <y1 >] [CR |:J

Creates a view-port on the display screen.

**VIEW**

**FORMAT: FUNCTION : Comments:**

A view-port is a logical screen display. It can be used to temporarily reduce the amount of the screen used for display. When a window is in effect the cursor cannot be moved outside it. The top left corner of the view-port becomes cursor position 0,0. Thus CURSOR and LOCATE commands work relative to the new view-port In general the M5 can be said be have a view-port of 0,0,39,23 when the power is turned on. X0,Y0 is the top left corner of the new view-port and X1,Y1 is the bottom left. X is the column and Y the row. VIEW does not clear the new view port. A view-port can be used to do relative character positioning for easily displaying graphs.

**Example:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 0 0 | i e w 5 .•> | 5 , |  | 5 ? 2 0 |
| This | e x a m P 1 | 0 | c | r e at e s a w i n d o w o t t o u r |
| 1 ines | a t t h | 0 | b | o11- om o t t h 0 sc r 08 " - |

**VPOKE**

FORMAT : VPOKE < memory address >, < output data > {CR |:}

FUNCTION : Outputs data to the video memory.

Comments : The data may also take the form of an expression. The data may not be a list as in POKE.

Example : u p o k e & 2 9 9 8 ,|f\*

R e a d y

Pr i a t vPee k ( &2 98 8 > 255 R e a d y

**VSAVE**

FORMAT: VSAVE <file-name>, <start-address>, <end-address>

|  |  |
| --- | --- |
|  | {CR |:} |
| FUNCTION : | Writes Video RAM data to external memory. |
| Comments: | Writes VRAM data between these two addresses to cassette tape. |
| Example: | u s a u e " c rn t. : P I C 1 " |

WAIT <time out count >[, cbase time>] {CR |:}

WAIT limits the amount of time that the computer will wait for input from the keyboard.

**WAIT**

**FORMAT: FUNCTION :**

**Comments: Example:**

The actual wait time in seconds is computed as:

wait = time out count \* (base time / 60)

If the time out count is 0 then the timer will not function. If the base time is 0 then it is set to 256. If the base time is left out, then it is set to 60. This function is most useful in writing games in which response time is important.

**18 r e m wait -for 0.5 second 20 wait 3 0 j1 30 inPut "data";fl$**

40 Print 50 goto 20

* 1. **Functions**

In this section all the BASIC-F functions are defined. The layout is similar to the previous section. The FORMAT entry is slightly different; all functions may appear anywhere in an expression hence we do not specify the context syntax. Functions can not be executed directly without the use of a statement such as PRINT. Functions which take parameters are shown with sample variable names; constants may also be used. If more than one type is possible, variables of both types are shown.

|  |  |
| --- | --- |
| ABS  FORMAT: | ABS (X) |
| FUNCTION : | Returns the absolute value of X. |
| Comments: | X must be a number. |
| Example: | P r i n t a fa s C - 3 4 > 34 |

R e a d y

**ASCII**

FORMAT : ASCII (X$) {CR

|  |  |
| --- | --- |
| FUNCTION : | Returns the ASCII code for the first character of string X$. |
| Comments: | As a matter of legibility X$ should be one character long or replaced with ASCII(LEFT$(A$,1)). ‘ASCII’ stands for ‘American Standard Code for Information Interchange. The converse function is CHR$(X). |
| Example: | 10 0 A $ = " A BC"  1 1 s3 p r i Ti t ascii < A $) 128 end |

|  |  |
| --- | --- |
| ATN  FORMAT: | ATN (X) |
| FUNCTION : | Returns the arc tangent of X. |
| Comments: | Legal range for X is ±6.8 E 74  Care should be taken to use either radians or degrees  consistently. |
| Example: | 1 00 fl = atn <10) 1 1 0 P r i n t fi 1 2 0 e n d |

ATN (X)

**CALC**

FORMAT: CALC (< string >)

|  |  |
| --- | --- |
| FUNCTION : | Performs a BASIC-F operation on an expression represented as a string. |
| Comments: | This function allows BASIC-F to compute expressions unknown at the time the program is written. This is a very powerful function, but it may introduce problems in debugging. This function is useful in mathematical programs involving equations unknown to the programmer. See also EXE. |
| Example: | 10 ' Cal cl at ion 20 input "calc";fi$ 30 fl=calc(fi$)  40 Print R $; " = " ; fi 45 Print 50 goto 20 |

**CDBL**

FORMAT: CDBL (X)

|  |  |
| --- | --- |
| FUNCTION : | Converts an integer into a real number. |
| Comments: | None. |
| Example: | 10 0 x x = 10 x  110 P r i n t X 3 X 1 2 0 P r i n t c d b 1 < X X > / 3 130 end |

r u n

|  |  |
| --- | --- |
|  | R e a d y |

**CHR$**

FORMAT: CHR$ (X)

|  |  |
| --- | --- |
| FUNCTION : | Returns the character whose internal code is X. |
| Comments: | This function is useful for accessing invisible or non- displayed characters. A complete list of character codes can be found in the appendices. X must be a valid ASC11 code. |
| Example: | 10 ’ CHR$ TEST 20 for 1=65 to 90 30 Print c h r %< I>’ 40 next I |

|  |  |
| --- | --- |
| CINT |  |
| FORMAT: | CINT (R) |
| FUNCTION : | Converts a real number to an integer. |
| Comments: | Rounding occurs in the conversion. |
| Example: | 10 0 Print c i n t <1.4) 110 Print c i n t <1.5) 120 end |

|  |  |
| --- | --- |
| cos  FORMAT: | COS(X) |
| FUNCTION : | Returns the cosine of X. |
| Comments: | As with all trigonometric functions, care should be taken  to use either radians or degrees consistently. The range of values  for X is ±2.8 E 16. |
| Example: | 1 0 0 fi — c o s (. P i •••■' 3 1 i 0 P r i n t fi 12 0 en d |

|  |  |
| --- | --- |
| ERR  FORMAT: | ERR |
| FUNCTION : | Returns the error code of the most recent error. |
| Comments: | This function, along with ERRL and ERRLS, is useful for programs which trap errors and attempt to deal with them internally. |
| Example: | 18 0 $ Pi B C 118 ft = B $ 1 2 8 P r i n t- ft 1 3 0 r e t u r n |

ERRL

See also ERR and ERRL$.

**ERRL**

**FORMAT: FUNCTION : Comments: Example:**

Returns the line number of the most recent error.

1 0 0 $ 0 B C 110 Pi = B $

12 0 Pri n t Pi

run

Err 13 in 110 Ready-

print err»errl?err 13 110 ABC

**Ready**

**ERRLS**

FORMAT: ERRLS

|  |  |
| --- | --- |
| FUNCTION : | Returns the label of the line in which the most recent error occurred. |
| Comments: | See also ERR and ERRL. |
| Example: | 1 0 0 $ fi 8 C 110 fi = B $  1 2 0 P r i n t Q 1 3 0 r e t u r n  run  Err 13 in 110 Ready  print err?errl»err 1$ 13 110 ABC  Ready |

FORMAT : EXE < character string > {CR |:}

FUNCTION : Executes a BASIC-F statement which has a string representation.

Comments : This statement allows the program to execute BASIC-F

statements which are unknown when the program is written. It allows the program to write program segments and execute them. The string must not contain another EXE command and an error will occur if there is a loop stack. This statement may also result in programs which are very difficult to debug. Since the program may execute another program, which does not exist in the original program, bugs may be difficult to find. This function is very useful for a BASIC-F program which can control other BASIC-F programs, and still maintain control of the computer. See also CALC.

Example:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 0 | 0 | n |  | 0 | r | r | o | r |  | •3 | 0 | s | IJ | b |
| 1 | 1 | 0 | i | n | P | ij | t | ; | P | $ |  |  |  |  |  |  |
| T—1 | •-> | 0 | 0 |  | 0 |  | I! | g | o | s | ij | b | II | + | P | $ |
| 1 1 | T | 0 | •3 | o | t | o |  | i | 1 | 0 |  |  |  |  |  |  |
| 1 | 4 | 0 $ H | | fi | fi |  |  |  |  |  |  |  |  |  |  |  |
| 1 | CT | 0 | P | r | i | n | t |  | It | H | H | fi | fi | fi | fi | fi |
| 4  i | fi | 0 | r | 0 | t | u | r | n |  |  |  |  |  |  |  |  |
| 1 | (' | 0 $ B | | B | B |  |  |  |  |  |  |  |  |  |  |  |
| 1 | o | 0 | P | r | i | n | t |  | II | B | R | B | B | B | B | B |
| 1 | 9 | 0 | r | 0 | t | u | r | n |  |  |  |  |  |  |  |  |
| 'jt | 0 | 0 $ C | | c | c |  |  |  |  |  |  |  |  |  |  |  |
| 2 | i | 0 | p | r | i | n | +  L\* |  | ii | C | c | C | c | c | c | c |
| •“i  jL | 2 | 0 | r | 0 | t | ij | r | n |  |  |  |  |  |  |  |  |
| •” | T | 0 | \* | E | R | R |  |  |  |  |  |  |  |  |  |  |
|  | 4 | 0 | 7 | t |  | 0 | u-  1 | r | = | 0 |  | t | h | 0 | n |  |
|  | 5 | 0 | r | 0 | s | IJ | m | 0 |  | 1  X | i | 0 |  |  |  |  |



|  |  |
| --- | --- |
| EXP  FORMAT: | EXP (X) |
| FUNCTION : | Reforms e to the power of X. |
| Comments: | The range of X is from -175 to +175 exclusive. |
| Example: | i 00 fi = exp <2 > 1 2 9 p r i n t h i o 0 0 n d |

FIX (X)

Returns the truncated integer portion of X. FIX (X) = SGN (X)\* I NT (AGS(X))

**FIX**

**FORMAT: FUNCTION : Comments: Example:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | yj. | y = |  | a |
| 1 | ■i  i | 0 | ! i \_ ‘I — | i | 1 |
| ■f |  | 0 | P r | i | n |
| i | T | 0 | e n | d |  |
| run | | |  |  |  |
|  | 1 |  |  |  |  |

**Ready**

|  |  |
| --- | --- |
| FRE  FORMAT: | FRE (X) |
| FUNCTION : | Returns information about memory usage. |
| Comments: | FRE returns five values according to the following table:   1. - maximum size of work area 2. - remaining user area 3. - remaining free work area 4. - remaining free user area and work area 5. - last address used by BASIC |
| Example: | P r 1 n t- t f" h c 3 6849 |
|  | This represents the total free RAM available to the user. |

FRE (X)

**HEX$**

FORMAT: HEX$ (X)

|  |  |
| --- | --- |
| FUNCTION: | Returns the hexadecimal equivalent of X in four hexadecimal places. |
| Comments: | No Zero suppression. |
| Example: | Pr int he x $<65535> FFFF |
|  | Ready |

**INKEYS**

FORMAT: INKEYS

|  |  |
| --- | --- |
| FUNCTION : | Reads in a character from the keyboard input buffer, without waiting for a keystrike. |
| Comments: | This function takes a character without waiting, like INPUT does. If the user has typed a character, it will be read in; otherwise the character will be a null (00). This function is useful in games as well as for writing ‘bomb’ proof input function. With INKEYS it is impossible to create an error as with INPUT, so routines can be written which will accept input of any form without generating BASIC-F errors. |
| Example: | 10 0 »$=inke y $  110 it fl $ <> ch r $<13 > the n 9ot o 1 8 8 1 2 8 p r i n t " E x i t "  138 end |

|  |  |
| --- | --- |
| \ |  |

INP(X)

Accepts one byte of input from port X.

**INP**

**FORMAT: FUNCTION : Comments:**

**Example:**

X must be in the range 0-255. A port will always provide a byte without waiting for input. For example, a program to read a serial port connected to a modem will always have a byte ready even if the modem has not received a byte. To use a serial port correctly two actual ports must be read. The first port will have a bit to indicate when a byte has come to the other port from the sending device, modem or otherwise.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | r e | rn |  | M | 0 | D | EM = | 5 | I 0 P 0 r t |
| i | 1 | 0 | r | 0 | P | 0 | a | t |  |  |  |
| l | •- | 0 |  |  |  | «\*\*« | = | i | n P < S T | fi | T U S > |
| i | \*7 | 0 | u | n | + | i | 1 |  | y = |  |  |
| i | 4 | 0 | X | = | i | n | P |  | M 0 D E M | ) |  |
| l | cr  .j | 0 | c | \* | = | c | h | r | ■$ <X ) |  |  |
| i | 6 | 0 | r | 0 | t | y | r | n |  |  |  |

This loop will wait until it receives a byte from the port. For this loop to work STATUS must be assigned the number of the status port on your serial port and MODEM must be assigned the number of the serial port itself.

**INSTR**

FORMAT : INSTR ([X,] STRS, SUBS)

FUNCTION : Searches string STRS for the first occurrence of string SUBS.

Comments : The search for SUBS starts after the Xth character in STRS.

If the substring is not found, then the result returned is zero. This function is very useful for some applications but consumes a great deal of time, especially if the strings involved are long.

Example:

|  |  |  |
| --- | --- | --- |
| 1 0 0 | n $ =" fi b c | D E F G " |
| 1 1 0 | 8 i = " D " |  |
| 1 2 0 | C = i n s t r | < 1 , R $ ,B $ |
| 1 7t 0 | P r i n t C |  |
| 1 4 0 | C = inst r | < 5 , fl $ , B $ |
| 150 | P r i n t- C |  |
| 1 6 0 | B $ = " H " |  |
| 170 | C = l n s t r | ( 1 ? R $ ? B $ |
| 1 8 0 | P r i n t C |  |
| 1 9 0 | e n d |  |

r u n 4 0 0

**R** e a **d y** 0

INT (X)

Truncates the real and returns only the integer portion of X. Use the FIX function to round a real number.

**Comments: Example:**

|  |  |  |
| --- | --- | --- |
| 1 0 0 | 0=10:E | 3=1.5 |
| 1 1 0 | P r i n t | Pi \* e |
| 1 2 0 | Print | Pi \* i n |
| 130 | e n d |  |
| r u n |  |  |
| 1 5 |  |  |
| 1 0 |  |  |
| Rea | d y |  |

**LEFTS**

FORMAT: LEFTS (X$, Y)

|  |  |
| --- | --- |
| FUNCTION : | Returns the left substring of X$ whose length is Y |
| Comments: | Care must be taken to ensure that Y does not exceed the length of X$. |
| Example: | 10 0 fi $ =" 12 3 4 5 RB CDE uwx yz " 110 B $ = 1 e f t. $ ( fi $ , 5 > 1 2 0 P r i n t B $ 130 end r u n 1 2 3 4 5  R e a d y |

LEN (X$)

Returns the length of character string X$.

**LEN**

**FORMAT: FUNCTION : Comments: Example:**

The maximum length of a character string is 18.

10 0 R $ = " H El C D E F13 " 110 B = 1 e n (.R $ >

1 2 0 P r i n t B r u n

R e a d y

LN (X)

Returns the natural log of X.

**LN**

**FORMAT: FUNCTION : Comments: Example:**

The natural log is a logarithm with base e.

|  |  |  |
| --- | --- | --- |
| 100 | ■for 1 = 1 to | 10 |
| 110 | R = 1 n < I ) |  |
| 120 | Print I;" | -> |
| 130 | next I |  |
| 140 | end |  |

|  |  |
| --- | --- |
| LOG |  |
| FORMAT: | LOG (X) |
| FUNCTION : | Returns the log of X to the base 10. |
| Comments: | None. |
| Example: | 10 for 1=1 to 10 20 fi = 1o9 < I >  30 Print ; fl 40 next I 50 end |

**MID$**

**FORMAT: FUNCTION :**

**Comments: Example:**

MID$ (X$, X [,Y])

Returns the substring of X$ starting at character X and ending at character Y.

The maximum length of a character string is 18.

100 fl$="12345RBCDEvwxyz 110 B$ = mid$(R$j 5)

120 Print B$

130 end

NUMS

FORMAT : NUM$ (X)

FUNCTION : Converts the numeric value of X to its character equivalent.

|  |  |  |
| --- | --- | --- |
| Comments: | Do not | exceed maximum integer value. |
| Example: | 1 0 0 | M = 9 9 9 |
|  | 1 1 0 | B $ = " fi B C " |
|  | 120 | C $ = B $ + n u m $ < fi > + " K V " |
|  | 1 3 0 | P r i n t. C $ |
|  | 1 4 0 | e n d |
|  | r u n fi B C | 9 9 9 y y |
|  | R e a d | y |

|  |  |
| --- | --- |
| OUT  FORMAT : | OUT <port number>, [<output data>[,..]] {CR |:} |
| FUNCTION : | Sends data byte by byte to a port. |
| Comments: | The output data may be the result of an expression. Care should be taken with port numbers as both hexadecimal and decimal numbers are used to refer to both ports and data. Output to an incorrect port could damage a program saved on tape or disk. |
| Example: | 1 0 0 O U t St 2 0 ? Z: 1 0 |

PEEK (X)

Returns the contents of memory address X. Returns 8 bits stored in CPU memory address X.

PEEK

**FORMAT: FUNCTION : Comments: Example:**

18 0 A = p ee k (& F F F F)

1. **Print- A** 120 end

PEEKW

FORMAT: PEEKW (X)

|  |  |
| --- | --- |
| FUNCTION : | Returns 16-bits from CPU memory address X and address X+1. |
| Comments: | This will return both X and X+1. |
| Example: | 1 8 0 P o k e & 7 2 0 0 , & 3 0 1 i 0 Poke &: f\* 2 0 1 ? 4 0 12 0 fi = P e e k w<&7 2 00)  1 4 0 P r i n t i n t- ( fi •••" 2.5 6 > 150 Print fi and 255 1 6 8 0 n d |

|  |  |
| --- | --- |
| PI |  |
| FORMAT: | PI |
| FUNCTION : | Returns the value of Pi |
| Comments: | Pi = 3.14159265359 |
| Example: | 10@ fi = s i n< Px/ |

1 1 0 P r l n t H i z y 0 n d

RDSTS

FORMAT: RDSTS (X)

|  |  |
| --- | --- |
| FUNCTION : | Reads the statement indicated by the cursor from the beginning. |
| Comments: | Characters to the right of position x will not be read. The end of the statement is indicated in the display by '00. If X does not exhaust the statement, the string CR (’OD) is attached. |
| Example: | 10 els   1. Print cursor(5>1)5"Hel1o there!" 2. locate 5 > 1   40 for C=1 to 13 50 A $ = r d s t $ < C >  60 Print cursor(5fC+5)  70 1ocat e 5 > 1 30 next C 9 0 end |

|  |  |
| --- | --- |
| REG |  |
| FORMAT: | REG (X) |
| FUNCTION : | Returns the register value after a CALL statement has been executed. |
| Comments: | X can signify various registers:   1. AF 2. BC 3. DE 4. HL   Data is stored “high-low.” Thus if Ar contains &2E and Fr contains &30 then “NEWAF” below will have &2E30. EX- CHG (NEWAF) gives &302E and EXCHG (NEWAF) AND 255 gives &2E. |
| Example: | 10 els  20 9 o s u b $SETCRLL:’Prepare for call 30 call ROUTINE,RFREG,, ,HLREG 40 RF=re9<0>  50 HL=re9<3>   1. 0 Print cursor<0,2)?“ RFREG = Sc";hex$<RF   "HLREG=&"\*hex$(HL)   1. 0 end   30 $SETCRLL 90 R0UTINE = Stl4BD 100 RFRE6=&4100 110 HLREG=S3300 120 return |

RIGHTS

FORMAT : RIGHTS (X$, < length >)

|  |  |
| --- | --- |
| FUNCTION : | Returns a substring from the right side of X$ of specified length. |
| Comments: | In general all strings can be done with just the MIDS instruction. |
| Example: | 10 0 fi $ =" 1 2 3 4 5 fiBC DE u wx yz" 1 1 0 B $ = r i 2 h t- $ ( fl $ ; 5 >  1 2 8 P r i fi t 8 $  1 3 8 e n d  r u n •j i.,j x y z |

R e a d y

Returns a random number between 0 and X.

X is the seed. For most purposes, RND produces an acceptable distribution of numbers. For extremely sensitive applications, however, other methods of random generation should be used, even to the point of special hardware.

RND

**FORMAT: FUNCTION : Comments:**

**Example:**

When X is an integer, the value returned will be an integer between O and X. When X is a real number, the value returned will be a real number between O and X.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i 00 | r a n d o m i z e | |  |  |
| 1 1 0 | P = r n i j < | 1 0 > |  |  |
| 120 | 8 = r n d < | : 10 | > |  |
| 1 3 0 | P r i n t | " r n | d < | •• 1  •• X |
| 1 4 0 | P r i n t | " r n | d 1 | .. l |
| 150 | e n d |  |  |  |

RPT$ (<repetitions>, X$)

Returns a string which consists of < repetitions> of X$.

RPT$

**FORMAT: FUNCTION : Comments: Example:**

This is useful for building long strings from a small pattern.

1 0 0 1 e n 3 0 1 1 0 X$ = rPt$<3 0? "\* " >

1. 0 P rin t X$
2. 9 e n d

r IJ n

\* \* \* + \* He \* He \* \* '¥ ■¥ '•¥ '■¥ He \* He \* \* \* He ■¥ \* H\* He He He He

SGN

FORMAT: FUNCTION : Comments:

Returns the sign of X.

SGN (0) = 0; if X>0 then SGN (X) = 1, otherwise SGN (X) = - 1. X may be real or integer or hex.

|  |  |  |
| --- | --- | --- |
| 1 0 0 | ft = 1 9 |  |
| 1 1 0 | P r i n t | s 9 n < fi |
| 1 2 0 | ft = 9 |  |
| 1 3 0 | Pr in t | s g n C fi |
| 1 4 0 | fi = - 1 0 |  |
| 150 | P r i n t | s 9 n < fi |
| 1 6 0 | e n d |  |

Example:

Returns the sine of X.

The range of values for X is ±2.8 E 16.

SIN

**FORMAT: FUNCTION : Comments: Example:**

|  |  |
| --- | --- |
| 1 0 0 | fl = s i n |
| i i 0 | Pr i n t |
| 120 | e n d |
| r u n |  |
| 0.8 6 | 6 0 2 5 4 |
| Rea d | y |

|  |  |
| --- | --- |
| SQR  FORMAT: | SQR (X) |
| FUNCTION : | Returns the square root of X. |
| Comments: | X can be a real number or an integer. |
| Example: | 1 0 0 fi = 2 110 B = s qr< fl >   1. 0 Print " SQR<2 > 2. 0 P r i n t. " S 0 R < 2 ) \* S Q R 3. 0 e n d |

|  |  |
| --- | --- |
| TAN |  |
| FORMAT: | TAN (X) |
| FUNCTION : | Returns the tangent of X. |
| Comments: | The range of values for X is ±2.8 E 16. |
| Example: | 1 88 X = 23 118 fi = t a n< X> 12 8 P rint ft 138 end |

TIME

FORMAT: TIME

FUNCTION : Returns the amount of time since the system was powered up. Value is in seconds.

Comments : The value returned is in seconds.

Example : i 00 rl I N = i n t < t ime/60>

1 1 0 S E C = t i rn e m o d 6 0 t 2 0 P r i n t M I N \* " M I N " \*' 5 E C \* 11 S E C 13 0 e nd

Converts a character string into its numeric equivalent. The string must be a legal number in BASIC-F.

1 0 0 ft $ = " 1 0 0 "

VAL

**FORMAT: FUNCTION : Comments: Example:**

110 B $ ="5 5"

12 0 Print, fi $ + B $ 1 3 0 P r i n t v a 1 < fi > + u a 1 ( B $ > 14 0 e nd

VARPTR

FORMAT : VARPTR ({X | X$})

|  |  |
| --- | --- |
| FUNCTION : | Returns the actual memory address of a variable. |
| Comments: | This function is required when machine language subroutines are to access BASIC-F variables. After the ad dress has been obtained it can be passed to the subroutine by any of several methods. |
| Example: | 10 0 A': = 2 5 5 110 B = u a r P tr (fi'»> 11! 0 C = P1 e a k 8 13 0 Pri n t. C 140 end |

VPEEK

FORMAT: VPEEK (X)

|  |  |
| --- | --- |
| FUNCTION : | Returns the contents of video memory at location X. |
| Comments: | Displays the data stored in video memory address X. |
| Example: | i 0 0 p r i n t " "  1 1 0 P r i n t c u r so r < 0 ? 0 > 5 " fi "  1 2 0 U = v Peek ( & 3 8 0 0)  1 3 8 P r i n t " fi S C I I < " c h r $ ( U > " > = "  i 4 p n d |

XCHG

FORMAT: XCHG (X)

|  |  |
| --- | --- |
| FUNCTION : | Swaps the order of the upper and tower bytes of X. |
| Comments: | Using XCHG moves original bits 0-7 (upper byte) to new bits 8-15 (tower byte) and original bits 8-15 (tower byte) to new bits 0-7 (upper byte) |
| Example: | 1 9 0 X = 12 3 110 fi y. = X c h '3 < X >   1. 0 Pr i n t ft y. 2. 0 e nd |

1. **Applications Section**

This section contains a set of useful programs which will run on your M5. They have all been tested by our staff. These programs make use of the special features of the M5 and BASIC-F. Each program has a statement of general function, a description of how to run the program with a sample run, a description of how the program runs and the program listing itself.

* 1. **Loan Repayments**

This program calculates the time needed to repay a loan. Given the amount of the loan, the interest rate, the number of payments made per year and the amount of each payment, the program will return the length of time that it will take until the loan is completely repaid. The program calculates the time according to the following equation:

"V ioq (1 + 4,) /N

where Y = term of payment in years P = principal i = interest rate

N = number of payments per year R = amount of each payment.

This could be used to calculate the length of time it would take to pay off a mortgage. How many years would it take to pay off a mortgage of $20,000 at 18% by quarterly payments of $1,000?

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | cl s |  |  |  |  |  |  |  |
| 2 0 7 |  |  |  |  |  |  |  |  |
| 3 0 | r e m c o m P u t- a | | t h e | ] e n g t h | o t | | t i rn e | |
| 4 0 7 | r e | P a y a | I o a n |  |  |  |  |  |
| 50 7 |  |  |  |  |  |  |  |  |
| 6 0 | 1o cate | 5,5 |  |  |  |  |  |  |
| 7 0 | Print- | "Ter m | o f a | 1 o a n " |  |  |  |  |
| 8 0 | Print |  |  |  |  |  |  |  |
| 9 0 7 |  |  |  |  |  |  |  |  |
| 1 0 0 | r em inP ut- t | | h e d | at a for |  | the | | c a |
| 1 1 0 |  |  |  |  |  |  |  |  |
| 120 | inPu t- | " r e 9 u 1 a r | | P a y rn e n t |  | ii | ; R |  |
| 1 3 0 | input | " P r i n | ciPa | 1 " \* P |  |  |  |  |
| 1 4 0 | i n P u t | " a n n u a 1 i | | merest |  | r | at | 0 " |
| 150 | inPut | " n u m b | e r o | f P a y rn e | n t | | •5 | P 0 r |
| 1 6 0 | 7 |  |  |  |  |  |  |  |
| 170 | r e rn P | e r f o r rn | c o rn P u t a t i o n | | | 3 |  |  |
| 1 8 8 |  |  |  |  |  |  |  |  |
| 1 9 0 | V =- | o g < 1 - < | P + C I | i 0 0) / < | N + R > | | | j / |
|  | 1 0 0 / N | ) \* N) > |  |  |  |  |  |  |
| 2 0 0 |  |  |  |  |  |  |  |  |
| 210 | r e m c | o rn p u t e | years and | | m o | | nths | |
| 2 2 0 |  |  |  |  |  |  |  |  |
| 2 3 0 | M = i n t | <V\* 12 + 0.5> | | s V 0 = l n t | ( | M | / i | 2 > |
| 240 | M = M- V | 0 \* 1 2 |  |  |  |  |  |  |
| 2 5 0 | P r i n t | "Ter rn | — il a i I ~ ? '» | 0 ; " year | | 5 | >i  y |  |
| 2 6 0 | Pr in t | " a n d " | ; M ■ " | m o n t h s |  | ii |  |  |
| 2 7 0 | P r i n t |  |  |  |  |  |  |  |
| 2 8 0 |  |  |  |  |  |  |  |  |
| 2 9 0 | i n P u t | " Once | rn o r | 0 ? < y / n |  |  | ii ■ | V $ |
| 3 0 0 | if 1 e | f t $ ( V T | ,1) = | " y " the n | |  | g 0 | t 0 |
| 3 8 5 | 7 |  |  |  |  |  |  |  |
| 310 | Prin t |  |  |  |  |  |  |  |
| 3 2 0 | Print | " d o n e | >1 |  |  |  |  |  |
| 33 0 | e n d |  |  |  |  |  |  |  |

i o n

; N

+1

ears

* 1. **Initial Investment**

This program calculates the investment necessary to provide a stated future value in a. specified time period. Suppose you want to earn $10,000 from interest over 5 years, how much money would you have to put in the bank to start with? For any given investment, interest rate and time period the program calculates the required-initial investment. The program requires that you enter the time period, the goal total, the interest rate and the number of times that the interest is compounded in one year. The program bases its calculations on the following equation:

p = (1+i/N)NY

where P = initial .investment T = future value

N = number of times compounded per year Y = number of years i = interest rate

The interest rate must be entered as a whole value: for instance 8.5% is entered as 8.5. The number of years can include fractional parts, such as 5.5 for five and one half years.

1. 0 C 1 S
2. 0 1 o c a t e j > 5
3. 0 p ri nt "Initial i nme stme n t
4. 0 p r i n t 50 ’

|  |  |  |
| --- | --- | --- |
| A 0 | r e rn i n P u t d a t a f o r c o rn P u tat i o n s | |
| 7 0 ? |  |  |
| 8 0 | Pr in t |  |
| 9 0 | i n P u t | " firnou n t. o f 9 oa 1 " ' T |
| 1 0 0 | inP u t | " # o f cornP oun d s P er year 11 5 H |
| i i 0 | i n P u t | " # of y ears "5V |
| 120 | i n P u t | " N o rn i n a 1 interest rate " ; I |
| 130 |  |  |
| 1 4 0 | r e rn Per for rn c o rn P u t a t i o n s | |
| 1 5 0 | ,1 |  |
| 1 6 0 | I = I / N | /10 0 |
| 170 | ,1 |  |
| 1 3 0 | P = T < | i + n < n \* v > |
| 1 9 0 |  |  |
| 2 0 0 | P r i n t |  |
| 2 1 0 | P r i n t | "Initial in y est rn ent require d |
| 2 2 0 | Pr i n t | " t o ac hi e u e 9oa 1 total is i" |
| 23 8 | P r i n t | i nt <P\* 10 0 + 0.5 >/10 0 |
| 2 4 0 | Pr in t |  |
| 2 5 0 | P rint |  |
| 2 6 0 | i n P u t | " U n e rn o re t i rn e ? ( y n ) " ' V $ |
| 27 0 | it 1 e | f t $ < V $ , 1) = " y " t h e r, o t o 1 0 |
| 2 8 0 | P rint |  |
| 29 0 | p r i n t | " d o n e " |
| 3 0 0 | e n d |  |

* 1. **Regular Deposits**

This program calculates the amount of each deposit required to reach a goal total within a specified time period. Let us assume that you wish to save $5,000 to buy a car. You wish to buy the car at the end of one year and you want to know how much money you need to put in the bank each month. Or let’s say that you wish to save up the money for a down payment of $10,000 on your home. You want to save the money over three years. How much must you put in the bank each month in order to get the desired total at the right time?

To use this program you must enter the values of the total, the number of years and the interest rate. You must also specify the number of times per year that you will make deposits. This allows you to skip one or more months. For example, you may wish to make 11 deposits per year, skipping December as your other expenses may be high that month. Or you may decide not to make payments over a vacation month but spend the money on your holiday. The interest rate that you enter must be expressed as a whole value, such as ‘10’ for 10%, or ‘4.5’ for four and a half percent. Years may be expressed as fractions as well, for example six months would be entered as ‘.5’ for half of one year. The calculation for regular deposits is based on the following equation:

R = t((1-H/N)Nn y -1)

where:

R = amount of regular deposit T = future value i = interest rate N = number of deposits per year Y = number of years.

1. 0 C 1 S
2. 0 ’
3. 0 r

e m c omP u t e the deposit s re 4uired e m t o a-c h i e m e a d e s i r e d t o t a 1 a t e m s o m e f u t u r e d a t e

1. 0 r
2. 0 r
3. 0 ’
4. 0 1 o c a t a 0 ? 5
5. 9 0 ’

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 8 0 P | rint | “ R e 9 u 1 | ar DePosit- sched y | 1 | e " |  |
| 9 0 ’ |  |  |  |  |  |  |
| 1 0 0 | P r i n t |  |  |  |  |  |
| 185’ |  |  |  |  |  |  |
| 1 1 0 | r e rn i n P u t- n | | e c e s s a r y da t a |  |  |  |
| 12 0’ |  |  |  |  |  |  |
| 130 | i n P u t | " i..j h a t. | is the desired 9 | ij | a 1 11 5 | T |
| 1 4 0 | i n P u t | " w h a t. | is the interest | r | a t e " | : I |
| 158 | i n P u t. | " H o u | many d ePosits Per |  | year | H ; N |
| 1 6 0 | i n P u t | " H o y | m a n y ye ar s to 9o a |  | d a t | 0 “; |
| 170’ |  |  |  |  |  |  |
| 1 8 0 | r e m P | erf o r rn | t h e c o rn P u t a t i o n s |  |  |  |

1. 0 0 I = I / N / 1 0 0

2 10 R = T \* I •••- < <1 + 1 > A < N \* V ) - 1 > 2 2 0 ’

2 3 0 2 4 0’

r e m r o u n d o ft t o c ents R = i n t- < R \* 1 0 0 + 0.5 > / 1 0 0

250 2 6 0 ’

270 Print

2 S 0 P r i n t " E a c h d e P o s i t rn u s t be $ " \* R

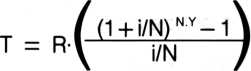
1. 9 0 ’
2. 0 0 P r i n t
3. 10 i nP u t "fi n other r u n < y/n > " \* V %
4. 2 0 if 1e f t $< V $ 7 1> = "y" the n ot o 1 0 3 3 0 ’

340 end

* 1. **Future Value of Regular Deposits (Annuity)**

This program calculates the total amount which will be saved if deposits are made regularly. Assume that a payment is made to an interest bearing account each month or at some other interval. What is the total amount in the account at any given time? For example, assume that you put $75 from your pay into a special ‘rainy day’ account each month. How much will you have saved after a year and a half, or after five years? Or assume that your company matches your $50 savings with a benefit in a company savings plan at 8%, how much will you save in three years? This program can perform the necessary calculations. You must enter the amount of each deposit, the number of deposits per year, the number of years and the interest rate. The number of years may contain a fractional part: for example three and a half years is entered as ‘3.5’ years. The interest rate is entered as a whole value. For example, seven and eight tenths of a percent is entered as 7.8’ for the program.

This program assumes that interest is compounded with each deposit according to the following equation:



where:

T = total value after Y years R = amount of regular deposit N = number of deposits per year Y = number of years i = nominal interest rate

As an exercise, modify this program to calculate the result with interest compounded over different time periods. Many accounts today compound interest on a daily basis, rather than monthly. This involves changing the value of Y for years and the way the interest value is used.

|  |  |  |
| --- | --- | --- |
| 1 0 | Pr int c | in r f ( 1 |
| 2 0 7 |  |  |
| 3 8 | 1 o c a t e | 0 j 5 |
| 4 8 | Print " | F u t u r |
| 5 0 | Print " |  |
| 6 0 | P r i n t |  |
| 7 0 7 | 1 o a d d | a t- a t |
| 8 8 ? |  |  |
| 1 0 0 | i n P u t | " fi m o u |
| 1 1 0 | i n P u t | " n o rn i |
| 1 2 0 | i n P u t | " n u rn b |
| 130 | i n P u t | " n u rn b |
| 1 4 0 |  |  |
| 1 5 0 | r e rn c a | 1 c u 1 a |
| 1 6 0 |  |  |
| 1 7 0 | I = I ••••■ H / | 1 0 0 |
| 1 3 0 |  |  |
| 190 | r e rn c a | 1 c u 1 a |
| 2 0 0 |  |  |
| 210 | T = R + < < | I + 1 > |

e a n n u 11 y

its

< a n n u i t- y > "

li ■ li

P ri nt "F u tu r e va 1ue = $ " \*

2 2 8 7 2 30 240 25 0 2 6 0 270 2 8 0

1. 9 0
2. 0 0 310

P r i n t i n t < T \* 1 0 0 + 8 . 5 > / 1 8 8

P r i n t

i n P u t " M o r e d a t a ? < y / n )

if left $<V$, 1 > = " y" th e n P r i n t

P r i n t " d o n e " e n d

i 0

* 1. **Remaining Balance on a Loan**

This program calculates the balance remaining on a loan after a specified number of payments has been made. You must input the amount of each payment, the number of payments per year, the amount of the principal, the annual interest rate, and the payment number from which you wish to calculate the remaining balance. The interest rate is entered as a whole value, thus 17% is entered as 17 and 6.5% is entered as 6.5.

For example, if you have a loan of $10,000 at 12.5% interest and your payments are $200.00 per month, how much will you still have remaining to pay after the 11th payment in the third year?

|  |  |  |
| --- | --- | --- |
| 1 0 C | 1 s |  |
| 2 0 7 |  |  |
| 3 0 r | arn comPuts | |
| 4 0 7 | o n | a 1 o |
| 5 0 7 |  |  |
| 6 0 ] | ocat a | 5,5 |
| 7 0 P | r i n t | " R a m a |
| 3 0 P | r i n t- |  |
| 9 0 7 |  |  |
| 1 0 0 | r a m i n P u t | |
| l i 07 |  |  |
| 1 2 0 | i n P u t | " r a g |
| 130 | i n P u t | " P r i |
| 1 4 0 | i n P u t- | n u m |
| 1 5 0 | inPu t | " an n |
| 1 6 6 7 |  |  |
| 1 7 0 | r a rn P | a r f o r |
| 1 3 0 7 |  |  |
| 1 9 0 | I = I 1 0 0 | |
| 2 0 0 7 |  |  |
| 2 1 0 | i n P u t- | "Las |
| 2 2 0 | i n P u t  > W 1 | " w h a |
| 2 3 0 7 |  |  |
| 24 0 | r a ni i | n it i a |
| 2 5 0 7 |  |  |
| 2 6 0 | B 0 = P |  |
| 27 0 |  |  |
| 2 3 0 | r e m a | c c u m u |
| 290 7 |  |  |
| 3 0 0 | •for J 1 = 1 t | |
| 3 1 0 | I 1 | = i n t- ( |
| 32 0 | fi = | R-I 1 |
| 3 3 0 | BO | = B 0 - fi |
| 34 0 | n a x t | J 1 |
| 3 5 0 7 |  |  |
| 3 6 0 | P r i n t- | " R a rn |
| 370 | Pr in t | i n t < |
| 3 3 0 | P r in t |  |
| 3 9 0 7 |  |  |
| 4 0 0 | i n P u t- | " 0 n c |
| 4 1 0 | if la | f t $ ( V |
| 42 0 | P r i n t |  |
| 43 0 | Pr in t | " d o n |
| 14 g | a n d |  |

o a n

c i ? a 1 " ; P

ar o f paymants Par year " 'N

1. Prime Factors

The following program calculates the prime factorization of a given number. Prime factorization is the expression of a number as the product of its prime factors. The method is to check all possible factors from 2 up to the square root of the number. In each case each factor is removed until the next factor is required or the factorization is complete. When this program detects a prime, it prints out that a prime has been found. The program, as given, does not make use of indentation. As an exercise, rewrite the program using indentation. Also, the program contains a branch at 230, out of the FOR NEXT loop. This is particularly bad programming style. As an exercise, rewrite this program without branching out of a loop.

To run the program, simply type RUN and enter the number that you want factored. After the factors have been printed, the program will ask for another number to factor. When you are done, ask for 0 to be factored, and the program will halt. As an exercise, modify the program to print out all the prime factors of the numbers from one to 1000. Another good exercise would be to change the program to display the current factor being tested at the top of the screen. Factors already found could be displayed near the middle of the screen. When all factors have been found, the program should wait till the user is done, then continue. This will require that you use the cursor control abilities of the M5, under BASIC-F.

The following short program is a stripped-down version of the previous prime factors program. This program is used to generate prime numbers only. As an exercise, see how many primes your M5 can generate using this program. Can you think of any ways to make this program faster? As an exercise modify this program to print out only twin primes, i.e. those pairs of primes with only one number between them, like 5 and 7 or 17 and 19.

10 0 rem 1 10'

120 els

prime factors

1. 0’
2. 0 locate 4 > 5

145 Print "Prime Factor s"

147 Print

150 i n P u t " l.iJ h a t n u m b e r d o y o u w ant f a c tore d "; N

160'

170 if N = 0 then end

18 0 if N = 2 then Print "2 is Prime": g o t o -3 0 0

|  |  |  |
| --- | --- | --- |
| 190 |  |  |
| 2 0 0 | M = 2: SW = 8 |  |
| 210 | I N = i n t < s q r < N + 0.5 > > |  |
| 22 0 | for L = M to- IN |  |
| 230 | if N / L = i n t < N L) then -3 o t o 2 | 5 8 |
| 240 | n ext L |  |
| 245 | if S W = 0 then Print N \* " is 3 0 0 | P r i m e |
| 247 | Print N : -3 o t o 3 0 0 |  |
| 250 | S W = 1 |  |
| 260 | if IN <L then 9ot o 2 90 |  |
| 270 | Print L; " \* " \* |  |
| 280 | M = L: N = N/L : 9 ot o 210 |  |
| 290 | Print L |  |
| 30 0 | PrintiPrint: goto 158 |  |
| 120 | c 1 s |  |
| 145 | Print "Primes" |  |
| 147 | H = 1 |  |
| 150 | N = N + 2 |  |
| 2 1 8 | I N = i n t < s 9 r < N + 0.5 > > |  |
| 22 0 | for L = 3 to IN |  |
| 230 | if M L = i n t. < N .••" L) t h e n 9 o t | o 1 5 8 |
| 24 0 | n ext L |  |
| 245 | Print N ? |  |
| 30 0 | goto 150 |  |

1. Long Number Arithmetic

This program allows your M5 to do simple arithmetic on very large numbers. Normally the size of the numbers that can be handled by your M5 under BASIC-F is limited by the BASIC-F interpreter. Integers have a limited size. With this program you can represent numbers of arbitrary size and do addition, subtraction and multiplication with them. The only limit on the size of the numbers is the memory capacity of your computer. In other words, except for the space occupied by the program itself, you can use the entire memory of your M5 to store 3 numbers using this program. This means that you can store incredibly large numbers.

The program works by breaking the large numbers into components which are small enough to be held in normal integer variables. These small parts of your number are stored in arrays. The arrays are A, B, and C and are declared in line 210. Array C is twice as large as A and B in order to accept large products of A\*B. You can change the size of A,B, and C by changing the values in lines 180 and 160. M must always be 2\*N + 1. Also, you can control the largest value in each part of the number by changing line 200. If you set this value (P) to the maximum power of 10 that can be represented on the M5 in BASIC-F, then you will be able to store the largest number of digits possible with this program. The program itself is extremely modular, and each section does only one task. The first routine is used to read in numbers from the keyboard. When using the program, zeros must be entered if the first part of the number is not used. For example, the current listing of the program requires numbers broken into 5 parts. If you want to enter the number 100, the first four parts must all be 0, and they must be entered. As an exercise you might try to modify the program so that only the last number need be entered, to save time. The program has four more subroutines. Each of them is called with a GOSUB. The first routine is used to print the long numbers on the screen. The second is used for addition, the third for subtraction and the fourth for multiplication. There is no routine for division in the program. Notice that division can be done by repeated subtraction. As an exercise add a new subroutine to perform division by calling the subtraction routine until the remainder is smaller than than the divisor. Print out the quotient and the remainder. As another exercise, you may wish to convert the input routine to use DATA statements, to save typing. Another interesting addition would be the ability to perform several operations, by reading in an expresion and then executing it one step at a time.

Remember, the numbers must each contain five groups of four digits. The possible operations are +, -, and \*.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 0 0 | r 0 n't n u rn b 0 r 0 P 0 r a t i 0 n s |  |  |
| 1 1 0 |  |  |  |
| 120 | c 0 n s 0 1 0 ? ? ? ? 3 |  |  |
| 130 | c 1 s |  |  |
| 1 4 0 | 7 |  |  |
| 150 | i 0 c a t 0 4 ? • j |  |  |
| 1 6 0 | Print- "Lon 9 n umber " 0P E R A T I | | O N S " |
| 1 7 0 |  |  |  |
| 1 3 0 | L = 4 : N = 4 : M = 9 |  |  |
| 1 9 0 |  |  |  |
| 2 0 0 | P = 1 0 0 0 0 |  |  |
| 210 | d i rn A < N > > B < N > > C < M > |  |  |
| 220 | 7 |  |  |
| 23 0 | P r i n t i p r i n t |  |  |
| 24 0 | Pr i n t "Enter the first | n u rn ber" | |
| 25 0 | 9 o s u b $L 0 A D J re rn read | a n u | m b e r |
| 2 6 0 | for 1=0 to N : A < I>= C( I | > : n e x | t I |
| 2 70 | ,1 |  |  |
| 28 0 | Print "Enter the s e c on d | n u rn b | er " |
| 29 8 | 9 o s u b $ L 0 A D |  |  |
| 30 0 | f o r I = 0 t o N ! B ( I > = C < | I > : n | ext I |
| 318 |  |  |  |
| 32 0 | i n P u t " i.i.l h a t o P e r a t o r ? " | ? 0 P $ |  |
| 33 0 |  |  |  |
| 3 40 | for 1=0 t o H: C(I)= A( | I > : n | ext I |
| 35 0 | 9 o s u b $ S H O l.'j: r e rn Print a | |  |
| 3 6 0 | Print |  |  |
| 3 70 | P r i n t O P $ : re rn Pr in t | oPerat or | |
| 3 8 0 | for 1=0 to N: C(I)= B( | I > : n | e x t I |
| 39 0 | 9 o sub $ S H 0 l.‘J J r e rn P r i n | t b |  |

400 Print 410 Print "=" 4 20 7

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 430 | for I= 0 to | M: C | < I | ) = 0 : | n e x t |
| 44 0 | r e rn cal 1 a P P r | oPr i | a t- | 0 0 P |  |
| 450 | 7 |  |  |  |  |
| 4 6 0 | if 0P$="+" | t h e n | g | o s u b | $ A D D |
| 470 | i f O P $ = " - " | t h e n | 9 | o s u b | $ S U B |
| 430 | if OP $ ="\*" | then | \*3 | o s u b | $ M U L |
| 4 9 0 | 9 o s u b $ H 0 l.i.l |  |  |  |  |
| 500 | P r i n t- ; Print | " d o n | 0 H |  |  |
| 5 1 0 | 7 |  |  |  |  |
| 520 | e n d |  |  |  |  |
| 53 0 | 7 |  |  |  |  |

1 0 0 0 ■$ L 0 N D 10 10 '

1 0 2 0 re m r e ad a nu m ber

Print "enter 5 9rouPs of 4 dibits each "

Print

1 040

10 50 10 60 1 0 70 1 0 8 0

1. 0 9 0 110 0
2. 0 0 0 2 0 10 2 020 2 0 3 0 20 40

2 0 50

20 6 0

2070 20 80

1. 0 9 0 2100 2110
2. 0 0 0 3010 3 0 20 3030 3040 3 0 50 30 6 0 3 0 7 0 30 3 0 3 0 9 0 3100
3. 118 3120
4. 0 0 0 4 0 10 4 0 2 0 40 30 40 40 40 50 4060 4070 4 0 3 0 4 0 90 410 0
5. 110
6. 0 0 0 5 0 1 9

**for I= N to 0 steP-1 inPut C < I > next I return**

J

**$SH0W S U = 0**

**for K = M to 0 steP-1**

if C < K)= 0 a nd S W = 0 then goto $ I 1 if S W = 0 then SW=l:Print mi d f(num $

**< C(K>) f 2) S :9oto $11**

**if C ( K) = 0 then Print 11 0 0 0 0 " \* : g o t o "11"**

Pr int r i g h t $ < " **0 0 0 0** "+mid$<n u m **$■** ( **C (.**

K ) '> , 2 ) ,4 >

$ I 1

**next K**

if SW = 0 then Print "0"; **return**

?

$ A D D

r e m addition

**C fl R R V = 0 for 1=0 to N**

**C < I) = I > + B < I ) + C A R R V CflRRV = 0**

if C CI)> = P then **C**(I>=C<I>-P:**CflRRV** next **I K = N + 1**

**C < K > = C fl R R V**

return

?

$ SUB ?

re m subtracti on

B R R W = 0

for 1=8 to N

C < I > = fl < I) - B < I > - B R R W B R R W = 0

if C <I)< 0 then C <I)= C < I > + P: B R R W=1 n e x t I return

$ MIJ L

r e m m u 1 t i P 1 y

5020 50 30 50 4 0 5 0 5 0 5 0 6 0 5 0 7 0 5 0 8 0 5 0 90 5 1 0 0

5110 5120 5 130 5140

CPRRV = 8 **•for 1 = 0 t o N** CPRR V = 0 **■for J = 0 t- o N**

C<I+ J>=C<I+ J> + H ( I > B < J> +CPRRV C P R R V = 0

if C (. I + J)> P then CPRRV = i n t<C(I+ J >/P CC I- : C < I + J > = C ( I + .J > - C p R R V + P n e x t- J

**C < I + N + 1) = C P R R V** n e x t I return

CODE INFORMATION CHARACTER CODES

Following are the ASCII representations of all characters stored and displayed on the M5 computer.

To use this appendix, find the character you want to display. Then look at the row of numbers and letters across the top and find the one that lines up with your character. Now look to the left at the leftmost column of numbers and letters for the corresponding number or letter. Combine these two numbers or letters. The one you found first is followed by the second.

Let’s look at three examples. Verify they’re correct in the table below.

|  |  |
| --- | --- |
| Character | Character  code |
| $ | &24 |
| H | &48 |
| + | &2B |

9

Pi

**I**

BCDEF



H

B

D

E

F

DEC (HEX) CODE

1. (20)
2. (21) !
3. (22) "
4. (23) #
5. (24) $
6. (25) X
7. (26) &
8. (27) ’
9. (28) (
10. (29) )
11. (2A) \*
12. (2B) +
13. (20 ,
14. (2D) -
15. (2E) .
16. (2F) /
17. (30) 0
18. (31) 1
19. (32) 2
20. (33) 3
21. (34) 4
22. (35) 5
23. (36) 6
24. (37) 7
25. (38) 8
26. (39) 9
27. (3A) :
28. (3B) 5
29. (30 <
30. (3D)
31. (3E) >
32. (3F) ?
33. (40) 3
34. (41) A
35. (42) B
36. (43) C
37. (44) D
38. (45) E
39. (46) F

|  |  |  |
| --- | --- | --- |
| DEC | (HEX) | CODE |
| 71 | (47) | G |
| 72 | (48) | H |
| 73 | (49) | I |
| 74 | (4A) | J |
| 75 | (4B) | K |
| 76 | (40 | L |
| 77 | (4D) | M |
| 78 | (4E) | N |
| 79 | (4F) | 0 |
| 80 | (50) | P |
| 81 | (51) | Q |
| 82 | (52) | R |
| 83 | (53) | S |
| 84 | (54) | T |
| 85 | (55) | IJ |
| 86 | (56) | V |
| 87 | (57) | w |
| 88 | (58) | X |
| 89 | (59) | Y |
| 90 | (5A) | z |
| 91 | C5B) | [ |
| 92 | (50 | \ |
| 93 | (5D) | ] |
| 94 | (5E) | /\ |
| 95 | (5F) |  |
| 96 | (60) | t |
| 97 | (61) | a |
| 98 | (62) | b |
| 99 | (63) | c |
| 100 | (64) | d |
| 101 | (65) | e |
| 102 | (66) | f |
| 103 | (67) | g |
| 104 | (68) | h |
| 105 | (69) | j |
| 106 | (6A) | j |
| 107 | (6B) | k |
| 108 | (60 | l |
| 109 | (6D) | m |

DEC (HEX) CODE

1. (6E) n
2. (6F) o
3. (70) p

1'13 (71) q

1. (72) r
2. (73) s
3. (74) t
4. (75) u
5. (76) v
6. (77) w
7. (78) x
8. (79) y
9. (7A) z
10. (7B) <
11. (70 !
12. (7D) >
13. (7E) ~
14. (7F) <
15. (80) ■
16. (81) O
17. (82) ♦
18. (83) \*
19. (84) -v
20. (85) x
21. (86) T
22. (87) |
23. (88) -
24. (89) H
25. (8A) h
26. (8B) +
27. (80
28. (8D)
29. (8E)
30. (8F)
31. (90)
32. (91) -
33. (92) -
34. (93) ■
35. (94) I

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DEC | (HEX) | CODE | DEC | (HEX) | CODE | DEC | (HEX) | CODE |
| 149 | (95) | 1 | 192 | (CO) |  | 235 | (EB) | \* |
| 150 | (96) | 1 | 193 | (Cl) | !\*i. | 236 | (EC) | A |
| 151 | (97) | ■ | 194 | (C2) | £ | 237 | (ED) | y |
| 152 | (98) | i | 195 | (C3) | f | 238 | (EE) | k. |
| 153 | (99) | yr | 196 | (C4) | Yl | 239 | (EF) | r |
| 154 | (9A) | a | 197 | (C5) | \* | 240 | (FO) | — |
| 155 | (9B) | ► | 198 | (C6) | 4 | 241 | (FI) | \_ |
| 156 | (90 |  | 199 | (C7) | K | 242 | (F2) | a. |
| 157 | (9D) | \ | 200 | (C8) | 6 | 243 | (F3) | m |
| 158 | (9E) |  | 201 | (C9) | 4 | 244 | (F4) | 1 |
| 159 | (9F) | J | 202 | (CA) | ij. | 245 | (F5) | 1 |
| 160 | (AO) | \ | 203 | (CB) | 0 | 246 | (F6) | 1 |
| 161 | (Al) | fi | 204 | (CC) | ».i | 247 | (F7) | ■ |
| 162 | (A2) | t | 205 | (CD) |  | 248 | (F8) | V |
| 163 | (A3) | 5 | 206 | (CE) | l | 249 | (F9) | ,« |
| 164 | (A4) | .V | 207 | (CF) |  | 250 | (FA) | x |
| 165 | (A5) | ft | 208 | (DO) | %. | 251 | (FB) | H |
| 166 | (A6) | ft | 209 | roi) |  | 252 | (FC) | A |
| 167 | (A7) | S | 210 | (D2) |  | 253 | (FD) | 'A |
| 168 | (A8) | 0 | 211 | (D3) | o | 254 | (FE) | k |
| 169 | (A9) | i=i | 212 | (D4) | u. | 255 | (FF) | r |
| 170 | (AA) | U | 213 | (D5) |  |  |  |  |
| 171 | (AB) | 8 | 214 | (D6) | £ |  |  |  |
| 172 | (AC) | •J | 215 | (D7) | I |  |  |  |
| 173 | (AD) |  | 216 | (D8) | 0 |  |  |  |
| 174 | (AE) | ♦ | 217 | (D9) | 0. | DEC-\*- | denotes base 10 | |
| 175 | (AF) | ■ | 218 | (DA) | A | HEX-\*- | denotes base 16 | |
| 176 | (BO) | 1 | 219 | (DB) | 2 |  |  |  |
| 177 | (Bl) | % | 220 | (DC) | i |  |  |  |
| 178 | (B2) | £ | 221 | (DD) |  |  |  |  |
| 179 | (B3) | 0 | 222 | (DE) | P- |  |  |  |
| 180 | (B4) | 6 | 223 | (DF) | \* |  |  |  |
| 181 | (B5) |  | 224 | (EO) | JS |  |  |  |
| 182 | (B6) | ¥ | 225 | (El) | • |  |  |  |
| 183 | (B7) | I | 226 | (E2) | 9 |  |  |  |
| 184 | (B8) | t | 227 | (E3) | ♦ |  |  |  |
| 185 | (B9) | ft | 228 | (E4) | S |  |  |  |
| 186 | (BA) |  | 229 | (E5) | / |  |  |  |
| 187 | (BB) |  | 230 | (E6) | \ |  |  |  |
| 188 | (BC) |  | 231 | (E7) | X |  |  |  |
| 189 | (BD) | H | 232 | (E8) | zz |  |  |  |
| 190 | (BE) | ¥ | 233 | (E9) | A |  |  |  |
| 191 | (BF) | 6 | 234 | (EA) | \* |  |  |  |

COLOR CODES

|  |  |
| --- | --- |
| Color | Color code |
| No color | 0 |
| Black | 1 |
| Green | 2 |
| Light green | 3 |
| Deep blue | 4 |
| Light blue | 5 |
| Deep red | 6 |
| Cyan | 7 |
| Red | 8 |
| Light red | 9 |
| Deep yellow | A |
| Light yellow | B |
| Deep Green | C |
| Purple | D |
| \* Gray | E |
| White | F |

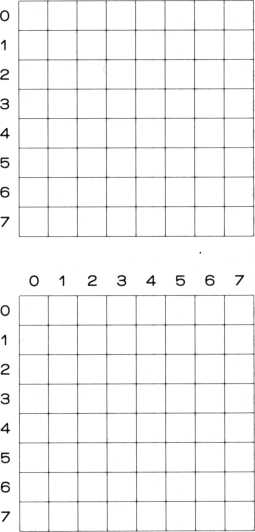
CONTROL CODES

These are functions that control the screen, cursor and a few other specialized functions. When using control functions directly after a READY prompt, press the CTRL key and the control key simultaneously. But when using control codes in a program (for example, in a PRINT statement), first press the CTRL and SHIFT keys before pressing the control key. Also enclose the control character in double quotes.

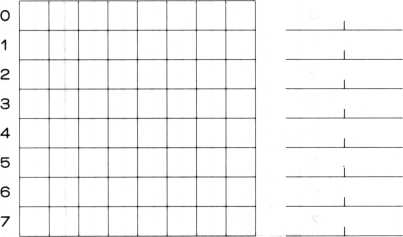
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Keyboard  Key | Base  10 | Base  16 | Function | Program Usage Display |
|  | 0 | 00 | Ignore |  |
| A | 1 | 01 | Ignore | (3 |
| B | 2 | 02 | Return cursor to beginning of current line |  |
| C | 3 | 03 | Scroll screen display down | a |
| D | 4 | 04 | Shift screen display left | IS |
| E | 5 | 05 | Scroll screen display up | is |
| F | 6 | 06 | Shift screen display right | is |
| G | 7 | 07 | Bell | [9 |
| H | 8 | 08 | Backspace | 11 |
| I | 9 | 09 | Tab the cursor eight spaces | tl |
| J | 10 | 0A | Move cursor down one line | y |
| K | 11 | 0B | Move cursor to home position | i a |
| L | 12 | OC | Clear screen display | IS |
| M | 13 | 0D | Same as RETURN key |  |
| N | 14 | 0E | Move cursor to beginning of next line | tsi |
| 0 | 15 | OF | Change to standard mode | UJ |
| P | 16 | 10 | Change to insert mode | is |
| Q | 17 | 11 | Change to multi-color mode | (El |
| R | 18 | 12 | Change to Gil graphics mode | IS |
| S | 19 | 13 | Change to Gl graphics mode | a |
| T | 20 | 14 | Return to text mode | u |
| U | 21 | 15 | Change to visible screen | m |
| V | 22 | 16 | Alternates between the visible and invisible screens, input is sent to the displayed screen | iy |
| w | 23 | 17 | Same as RETURN key | 161 |
| X | 24 | 18 | Delete characters to the right of cursor |  |
| Y | 25 | 19 | Alternates between the visible and invisible screens only | UH |
| z | 26 | 1A | Writes input to the alternate screen | a |
| [ | 27 | 1B | Ignore |  |
|  | 28 | 1C | Right arrow |  |
| ] | 29 | 1D | Left arrow |  |
|  | 30 | 1E | Up arrow | T |
|  | 31 | 1F | Down arrow | l |

**Coding sheet for 8x8 pixel pattern code**

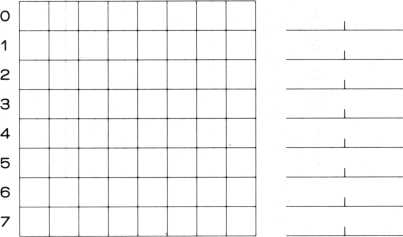
0 1 2 3 4 5 6 7



0 1 2 3 4 5 6 7

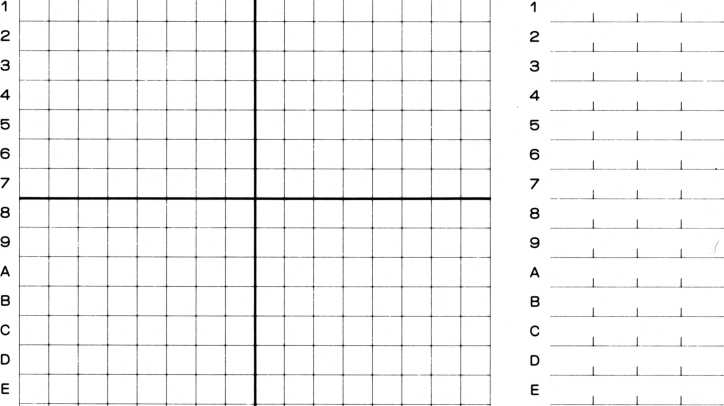


0 1 2 3 4 5 6 7



**Coding sheet for 16** x **16 pixel pattern code**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 1 | \_j | I |  |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | i | \_i | L |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 | i | \_j | I |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 3 |  | [ | I |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 4 | i | \_l | I |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 5 | i | \_j | I |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 6 | i | \_l | I |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 7 | i | \_| | I |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 8 | i | \_j | I |  |
| 9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 9 | i | \_| | I |  |
| A |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | A | i | \_j | . l |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | B | i | I | I |  |
| C |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | C | i | I | l |  |
| D |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | D | i | I | I |  |
| E |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | E | i | I | I |  |
| F |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | i | I | I |  |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |  |  |  |  |  |

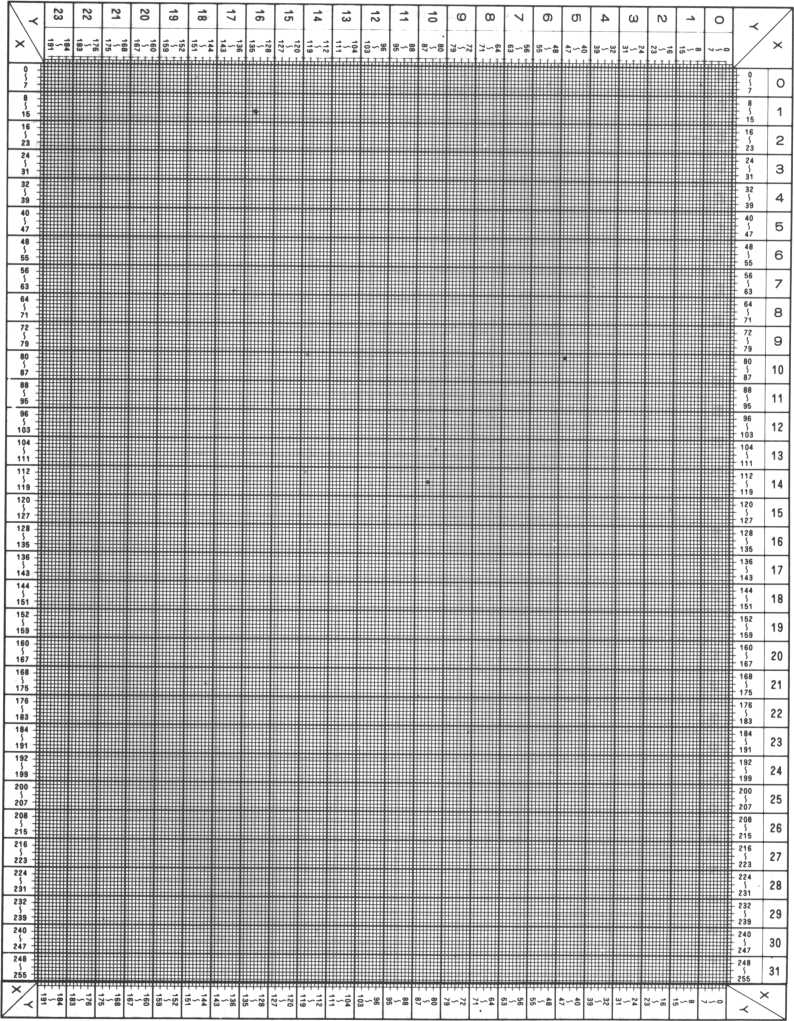


CRT LAYOUT COORDINATES

**CRT Screen Layout Sheet (Characters)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| £ | O | T" | CM | CO | \* | IT) | (0 | N | CO | 0) | o | - | CM | CO | •XT | in | CD | Cm | CO | 05 | o | CM | CM  CM | CO  CM | % |
| m |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | m |
| o  CO |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | o  CO |
| CO  CNJ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 05  CM |
| CO  CM |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | CO  CM |
| cm  CM |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Cm  CM |
| CO  CM |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | CO  CM |
| m  CM |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | in  CM |
| CM |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | xj-  CM |
| CO  CM |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | CO  CM |
| CM  CM |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | CM  CM |
| CM |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | CM |
| O  CM |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | CD  CM |
| a> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 05 |
| CO |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | CO |
| fv |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| CD |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | CO |
| in |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | LO |
| xf |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x» |
| CO |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | CO |
| CM |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | CM |
| - |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | £ |
| o |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | o |
| 0) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0) |
| GO |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | CO |
| IS |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | IS |
| (0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
| 1C |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| CO |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | CO |
| CM |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | CM |
| - |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | - |
| o |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | o |
| xy /y | O | r- | CM | CO |  | 10 | 0 | Is | 00 | 0) | o | £ | CM | CO | XT | in | CD |  | CO | 05 | o  CM | CM | CM  CM | CO  CM | y  >-\ |

CRT Screen Layout Sheet (Pixels)



KEYBOARD KEY CODES

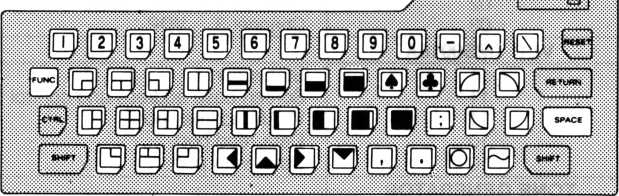
Key Codes

|  |  |
| --- | --- |
| / C |  |
| ®©©©@®©@@l©®l©(« —0 ® © © © © © © ® ® © © © H® ©©©©©© ©©©©©(“  HO [33] (34] [^5) [36) [37] (38^1 [39) (40) (45) [46] 57) ( | 5 □  l 8 1 ■8 |
| These key codes are ascertained using INKEY(1) |  |

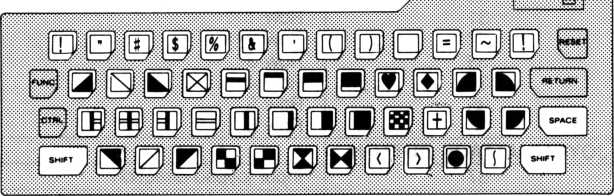
The codes of the shaded keys are known using INKEY(1). If more than two of these keys are pressed simultaneously, the sum of their key codes is returned. For example, if the FUNC and CTRL keys are pressed at the same time, “3” is returned (2 + 1).

|  |  |  |
| --- | --- | --- |
| Alphabetic characters with SHIFT key not pressed | |  |
|  | . S©©0©©-© © Ql©©®{D ^ ^ 1 - ; ©0©:©D00 Qtd : |
| Alphabetic characters with SHIFT key pressed  / msMssssms | |
|  | R | A; ‘slfpl [f) G-. Ik1 ' j,: IK- fU 1+ 1 T) C~1  ©©§:®jg ® ® mb© ® q |
|  | |  |

Graphic characters available with SHIFT key not pressed



Graphic characters available with SHIFT key pressed



POUT I/O TABLE

|  |  |
| --- | --- |
| POUT Number | Summary |
| Z80CTC |  |
| 01 | Channel # 1.... peripheral timer |
| 02 | Channel #2 ... I/O clock |
| 03 | Channel 3 ... VDP |
| VDP TMS9918A |  |
| 11 | Status port |
| 11 | Screen base address and control port |
| 11 | VRAM address port |
| 10 | Data read port |
| 10 | Data write port |
| TONE GENERATOR |  |
| 20 | Tone generator control |
| KEYBOARD |  |
| 30 | Row 0 |
| 31 | Row 1 |
| 32 | Row 2 |
| 33 | Row 3 |
| 34 | Row 4 |
| 35 | Row 5 |
| 36 | Row 6 |
| JOYPAD/ATTACK BUTTON |  |
| 37 | Input joypad direction |
| RESET/HALT KEY |  |
| 50 | Reset/Halt key data port (bit 7) |
| CASSETTE |  |
| RECORDER |  |
| 50 | Output port |
| 50 | Input port |
| 50 | Output port |
| PERIPHERAL |  |
| I/O |  |
| 40 | Data output |
| 50 | Strobe output |
| 50 | Printer busy |

MEMORY MAP

Memory map

Online main memory 0000

M5

(8 kB internal ROM)

2000

External ROM cartridge  
(maximum 20 kB)

ROM area

4000 6000

7000

8000

4kB internal RAM

4KB external  
RAM

Extended external RAM  
(maximum 32 kB)

RAM area

FFFF

**VRAM (video RAM) memory map Layout I**

0000 2000 2800 3000 3800 3B00 3B80 3C00 3F00 3F80 4000

Free area

Sprite pattern code table (2048  
bytes—both screen buffers 0 and 1)

Character pattern code table (2048  
bytes—screen buffer 0)

Character pattern code table (2048  
bytes—screen buffer 1)

Character to ASCII relationship table  
(768 bytes— screen buffer 0)

Sprite attribute table (128 bytes—screen buffer 0)

Character color table (32 bytes—screen buffer 0)

Character to ASCII relationship table  
(768 bytes— screen buffer 1)

Sprite attribute table (128 bytes—screen buffer 1)

Character color table (32 bytes—screen buffer 1)

Layout 1 remarks—applicable when the 8 expanded screens are used (uses addresses &0000 to &1 FFF)

**Layout II**

0000

1800

2000

3800

3B00

3B80

3C00

3F00

3F80

4000

Gil mode color table (6 kB)

Sprite pattern code table (2 kB)

Gil mode pattern code table (6 kB)

Pattern code table (screen buffer 0)

Sprite attribute table

Character color table

\*

Pattern code table (screen buffer 1)

Sprite attribute table

Character color table

\*

Note: \* signifies color table in other  
than Gil mode

ERROR CODES

|  |  |  |
| --- | --- | --- |
| ERROR  CODE | ERROR SUMMARY | REASON |
| ERR 1 ERRNF | FOR .. NEXT error | • FOR-NEXT does not correspond |
| ERR 2 ERRSY | Syntax error | • Non-existent command |
| ERR 3 ERRRG | Subroutine error | * CLEAR used in the subroutine * Jumped to subroutine using a GOTO * GOSUB-RETURN does not correspond |
| ERR 4 ERROD | READ error in DATA statement | * Insufficient data * Missing DATA statement |
| ERR 5 ERRIF | Variable type mismatch | • Wrong type of value given for statement variable |
| ERR 6 ERROV | Overflow | • When multiplying — answer is correct but beyond the negative limit. |
| ERR 7 ERROM | Memory exhausted | * Program is too long * Too many variables (reduce the number) * Too many subroutines (reduce the number) |
| ERR 8 ERRUL | Missing line number | • Missing destination for GOTO or GOSUB |
| ERR 9 ERRBS | Array variable error | * Error in array statement * Letter accompanying array variable outside scope of the statement |
| ERR 10 ERRDD | Array variable error | • Same variable set twice |
| ERR 11 ERRDZ | Division by 0 | • Divided by zero |
| ERR 12 ERRID | Inappropriate direct execution statement | • Wrong direct execution command (execute wrong program) |
| ERR 13 ERRTM | Inappropriate data item | • Characters were provided when numerics were expected, or vice versa |
| ERR 14 ERROS | Stack overflow | * Stack space exhausted * No stack area left for the PAINT statement |

|  |  |  |
| --- | --- | --- |
| ERROR  CODE | ERROR SUMMARY | REASON |
| ERR 15 ERRST | Character string length error | * Character string too long or becomes too long during calculation * Substituted a character string larger than the left hand variable |
| ERR 16 ERRUD | Array variable error | • Used an array variable which has not yet been allocated |
| ERR 17 ERRDL | Redundant label | • Used the same label more than once |
| ERR 18 ERRTR | Tape read error | * Tape read error * Reset during tape read operation |
| ERR 19 ERRDM | Wrong screen display mode | • Wrong screen mode chosen from Gl, Gil, text or multi-color. |
| ERR 20 ERRSP | Sprite error | • Tried to see or move a sprite that has been erased |
| ERR 21 ERRNS | Stack error | • Does not occur. |
| ERR 22 ERRUR | REPEAT..UNTIL error | •REPEAT..UNTIL does not match |
| ERR 23 ERRTO | Timeout error | * INPUT statement timed out * Response not received from floppy disk in time. |
| ERR 24 ERRRE | RESUM error | • Executed RESUM when no error occurred |
| ERR 25 ERRDF | INPUT error | • Pressed RETURN key without keying in any data |

|  |  |  |
| --- | --- | --- |
| ERROR  CODE | ERROR SUMMARY | REASON |
| Error  30 | SIO communications error | • Wrong input data |
| Error  100 | Channel unavailable | • All channels in use—appears when OLD, VERIFY, SAVE, or LIST are attempted |
| Error  101 | Specified channel not open | • Channel may not be OPENed for data input |
| Error  102 | Channel already in use | • You have tried to allocate more than one I/O device to a channel |
| Error | Specified device already in | • You have tried to use multiple chan |
| 103 | use | nels with the ACMT |
| Error  104 | Improper file name | * Non-existent device * File name too long |
| Error  105 | Improper access | * PUT or GET used with a device which cannot PUT or GET * You have randomly accessed a device which only handles sequential access |
| Error  106 | Wrong file | • A file name already used cannot be reused on the same disk |
| Error | Communications process | • Information transfer to disk not pro |
| 107 | error | ceeding correctly |
| Error  131 | Improper drive number | • The FD-5’s drives are numbered 0 and 1 |
| Error  132 | Incorrect file name | • Zero-character file names are not permitted with the FD-5 |
| Error  151 | Exceeded record | • You have tried to read or write across more than one record |
| Error  152 | Data finished | • You have attempted to read out data which has not been input |
| Error  154 | No space on the disk | • There is no empty field on the disk; file may not be expanded. Unnecessary files may be erased with KILL |
| Error  155 | Too many files | • The disk file capacity has been exceeded. Capacity is 108 files |
| Error  156 | Record not finished | • You have randomly accessed a record before access to the previous record has been completed |

|  |  |  |
| --- | --- | --- |
| ERROR  CODE | ERROR SUMMARY | REASON |
| Error  160 | No empty channel | • No more than four files may be simultaneously OPENed with the FD-5 |
| Error  170 | No relevant file | •The specified file is not on the disk |
| Error  171 | File already in use | • Specified file is in use on another channel |
| Error  172 | File already exists | •The file is already on the disk |
| Error  180 | Readout prohibited | • You have tried to read a protected file |
| Error  181 | Entry prohibited | • You have tried to enter to a protected file |
| Error  182 | Erase prohibited | • You have tried to erase a protected file |
| Error  190 | Wrong disk | • The file has been transferred to another disk |
| Error  191 | Panic | • Unexpected fault during normal operations. Transfer from the disk has possibly gone haywire |
| Error  214 | Abnormal sounds | • Mechanical problem with reading to or writing from the disk |

**Abvr. Page**

Remark statement or label name

**Definition**

$

ABS (F)

ASCII (F)

ATN (F)

AUTO BCOL CALC (F)

CALL CDBL (F)

CHAIN CHR$ (F)

Cl NT (F)

CLEAR

CLIST

CLOSE

CLS

COLOR

CONSOLE

CONT

COS (F)

CURSOR (F)

DATA

DEL DIM DRAW END ERR (F)

ERRL (F)

ERRL$ (F)

EVENT

EVENT ON/OFF EXE (F)

EXP (F)

FOOL FIX (F)

FOR..TO.. [STEP] FRE (F)

GCOPY

GET

GINIT

GMODE

GMOVE

GOSUB

GOTO

HEX (F)

IF..THEN..ELSE INKEYS INP (F)

INPUT INSTR (F)

I NT (F)

KILL

LEFTS (F)

LEN LEN (F)

LET LIST LN (F)

LOC LOCATE LOG (F)

MAG MIDS (F)

NEW

17

103

104

105

18

19

106

1. 107

21

\*108

1. 22

23

24

25

26

27

28

110

29

30

31

32

33

1. 111 112

113

35

36

114

115

37

116

38

117

39

40

41

42

44

45

46

118

47

119

120

1. 121 122

49

123

50

124

51

52

125

53

54

126

1. 127

56

Returns absolute value of X a

Returns ASCII code for first character of a string as

Returns arc tangent of X at

Automatic line numbering a

Sets screen background color b

Performs BASIC-F operations on an expression given as a ca

string

Transfers program control to a specific machine address ca

Converts an integer to a real number cd

Retrieves program from tape or disk and executes ch

Returns character with internal code X ch

Converts a real number to an integer ci

Clears section of memory for PAINT/character buffer cle

Lists in upper case letters cli

Ends file usage cl

Clears screen

Sets up character color (Gl and Gil modes) col

Enables/disables keyboard function keys cons

Restarts program after STOP or keyboard interrupt c

Returns cosine of X co

Moves curser to specified co-ordinates c

Stores constant information used by program and accessed d

by READ

Deletes lines from current program in memory de

Allocates memory for an array di

Draws a line on the screen dr

Indicates the end of a program and halts execution e

Returns error code of the most recent error

Returns the line number of the most recent error

Returns line label of most recent error

Sets event timer interrupt interval ev

Enables/disables event timer interrupt

Executes a BASIC-F statement having string representation ex

Calculates the function e\* ex

Sets character color or graphics display color fc

Returns integer portion of X fi

Performs many iterations of a section of the program f.~to~s.

Returns information on memory usage fr

Prints the current screen image on the printer gc

Reads data from channel to variable ge

Enters graphic mode gi

Sets up graphics display mode gmod

Moves graphics cursor gm

Transfers control to a subroutine gis

Transfers program control to line number/label in statement g

Returns hexadecimal equivalent of X h

Evaluates the conditional expression if -1. ~ e..

Returns current character from keyboard ink

Inputs a byte from a port

Assigns alphanumeric data from keyboard to variables i

Searches string STR$ for first occurrence of string SUB$ ins

Returns integer portion of a variable i

Deletes files k

Returns left substring of X$ lef

Resets the maximum length of string variables le

Returns length of character string I

Assigns the result of an expression to a variable

Lists a file to another file, the printer or screen I

Returns the natural log of X

Moves sprite-number to specified GR co-ordinates

Moves cursor to specified line and column lo

Returns log X to the base 10 lo

Changes sprite size and format m

Returns substring of X$ between X and Y m

Clears the current program and memory contents

NEXT Ends repeated executions initiated by FOR within a program n 57

**Definition**

**Abvr.**

**Page**

NUM$ (F) Converts numeric value of X to character equivalent nu 128

OLD Reads file from external memory o 58

ON ERROR GOSUB.. Transfers control to line number on error detection 59

ON EVENT GOSUB.. Calls subroutine when event timer interrupt occurs 60

ON..GOSUB.. Evaluates expression and branches to nth line number 61

ON..GOTO.. Evaluates expression and branches to nth line number 62

ON..RESTORE.. Sets data pointer to data group depending on expression 63

OPEN Opens user files op 64

OUT Sends data byte by byte to a port ou 129

PAINT Paints area indicated by GR co-ordinates pa 65

PEEK (F) Returns contents of memory address X 130

PEEKW (F) Returns 16 bits from CPU memory address pe 131

PI (F) Returns the value of Pi 132

PLOT Displays dot at specified GR coordinates pi 66

POKE Writes data directly into specified memory locations 67

POKEW Writes data to CPU memory po 68

PRINT Puts text in screen display buffers p 69

PUT Assigns binary form to a variable and executes it pu 70

RANDOMIZE Resets the seed for the random number generator ra 71

RDSTS (F) Reads statement indicated by cursor rd 133

READ Loads data from DATA statements into variables rea 72

RECORD Record to be next assigned rec 73

REG (F) Returns register value after CALL is executed 134

REM Stores programmer comments within the program 74

RENUM Changes the line numbering of the program ren 75

REPEAT Sets up a loop ending in a logical test rep jq

RESTORE Resets data pointer for data item groups in DATA statements res 77

RESUME Bypasses an error resu 78

RETURN Returns program control after GOSUB has been called re 79

RIGHTS (F) Returns substring from right side of X$ ri 135

RND (F) Returns a random number between 0 and X rn 136

RPT$ (F) Returns a string of repetitions of X$ rp 137

RUN Executes the current program r 80

SAVE Writes to external memory sa 81

SCOD Assigns numeric code to sprite-number sc 82

SCOL Colors sprite-number 83

SG Activates tone and noise generators 84

SGN (F) Returns sign of X sp 138

SIN (F) Returns the sine of X 139

SLEEP Stops execution for specified sleep time si 86

SQR (F) Returns square root of X sq 140

STCHR Assigns pattern-code to character-code stc 87

STEP Stops/resumes execution at statement change ste 88

STOP Halts execution of a program from within a program s 89

SWAP Transfers contents of variables sw 90

TAB Tab over X characters ta 91

TAN (F) Returns tangent of X 141

TAPE Accesses assembler supplied on external memory ta 92

THETA Sets mode for trigonometric functions to degrees/radians th 93

TIME < Returns time since powering up in seconds ti 142

TRACE Displays trace of executed line numbers during program t 94

execution

TYPE Defines the type of a variable ty 95

UNTIL Marks the end of a repeat loop u 98

VAL (F) Converts a character string to its numeric equivalent v 143

VARPTR (F) Returns the actual memory address of a variable var 144

VPEEK (F) Returns contents of video memory at X vp 145

VERIFY Compares programs in memory v 97

VIEW Creates a viewport on the display screen vi 98

VPOKE Outputs data to the video memory vp 99

VSAVE Writes video RAM data to tape vs 100

WAIT Limits time computer will wait for input from keyboard w -j 01

XCHG (F) Swaps order of upper and lower bytes of X xc i46



SORD COMPUTER CORPORATION

SAITO BLDG. 2F, 14-6, KYOBASHI 3-CHOME,

CHUO-KU, TOKYO 104, JAPAN PHONE: (03) 562-6061 TELEX: 2522745 (SORD J)

GDE-0083-1 (87100675)

1513 Printed in Japan June 84’NK