Introduction to BASIC Programming

COMX 35 is a trademark of COMX World Operations Ltd

ClibPDF - www.fastio.com

© COMX WORLD OPERATIONS LTD., 1983

All rights reserved. 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 COMX WORLD OPERATIONS LTD.

The information in this document has been carefully checked and is believed to be entirely reliable. No responsibility, however, is assumed for inaccuracies. Furthermore, such information does not convey to the purchaser of the product described any licence under the patent rights or copyrights of COMX WORLD OPERATIONS LTD. or others.

Printed In Hong Kong

CONTENTS

ClibPDF - www.fastio.com

ClibPDF - www.fastio.com

ClibPDF - www.fastio.com

PREFACE

USING THE COMX 35

Preface

Congratulations! You are now the proud owner of a COMX 35 computer. Together with this manual, you now have the keys to the Magic Kingdom of Computing.

The COMX 35 is a powerful, yet easy to use, computer and this guide will show you how. This manual serves three purposes,

(1) ... to enable you to have fun with the COMX 35 without going through the whole book. Every step will be explained, and you will learn by "hands-on" experience (chapters 1, 2 and 3),

(2) ... to provide you with some fundamental computer concepts, and a comprehensive reference guide to the COMX 35 BASIC language (chapters 4 and 5),

(3) ... to provide you with many programming examples illustrating the more advanced aspects of the art of programming (chapters 3, 5 and 6).

To turn the computer on and connect it to your TV, follow the steps in Chapter 1. The COMX 35 computer has been especially designed to be easy to use, or as the computer experts say, "It is user friendly."

The fundamental commands the COMX 35 obeys are described in Chapter 2. Follow the step-by-step guidance, you will soon be in the world of computer arithmetics, color graphics, sound effects and video games.

Chapter 3 deals with elementary programming, including teaching the computer how to make decisions, and to obey a section of program repeatedly. Programming exercises with answers to selected problems are provided. As a budding programmer, you will be guided to form good programming habits, such as using "REM" or "comment statements" to record the thinking behind each section of your program.

If you aspire to be a competent programmer, you should pay attention to Chapters 4, 5 and 6. Concepts such as algorithm, use of subroutines and comprehensive information about the COMX 35 BASIC, graphics, and sound effects are presented.

ClibPDF - www.fastio.com

The COMX 35 is a source of fun and games. But, it is also a useful tool at the office, at home, and in the school. The COMX 35 and this Manual have been designed to serve also as a self-learning package for BASIC PROGRAMMING and basic computer concepts.

To program a computer is to write a sequence of instructions for the computer to obey. Solving problems by programming is rapidly becoming a basic skill, which, like reading, writing and doing arithmetic, is to be included in the "survival kit" of a school curriculum. The COMX 35 offers something to everyone — the hobbist, the businessman or the student.

Programming helps to sharpen the mind and promotes logical thinking. Color graphics and computer music fire the imagination. The rapid response of the COMX—35 will enhance your creativity.

ClibPDF - www.fastio.com

CHAPTER ONE

GETTING STARTED

CHAPTER 1 GETTING STARTED

1.1 Unpacking the COMX 35

The COMX 35 package consists of

(1) ... the COMX 35 computer with built-in keyboard and joystick,

(2) ... a power cord with a plug at one end, and an AC/DC converter at the other. The converter is intended to be plugged into a household AC socket on the wall. Please check that your AC wall outlet voltage rating matches the input rating on your converter.

(3) ... a video cable with a plug at one end, and a connector at the other for plugging into the antenna input of the TV,

(4) ... a pair of cables (with two plugs on each end) for connecting COMX 35 to an audio cassette recorder, and

(5) ... a user manual entitled "Introduction to BASIC Programming".

In addition to the above, you will need,

(a) ... an ordinary home TV. (COMX 35 sold in Europe and Hong Kong are designed for TV sets using the UHF PAL system, whereas COMX 35s sold in the U.S.A., Canada and Mexico are designed for TV sets using the VHF NTSC system.)

(b) ... an ordinary audio cassette recorder (but NOT a stereo recorder) which is equipped with earphone and microphone jacks and some cassette tapes for storing your programs.

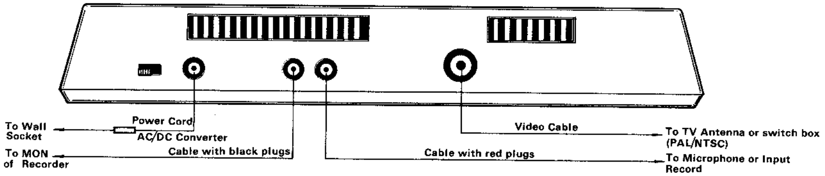
Note: Like all delicate electronic equipment, your COMX 35 computer should be stored and operated in a cool and dry environment. High humidity is especially harmful to your computer. If your COMX 35 is to be stored for a prolonged period, place it in a plastic bag together with a bag of silica gel.

ClibPDF - www.fastio.com

1.2 Connecting up the TV and the Cassette tape recorder

The television set is used as the display device. Input into the computer (by pressing the keys on the keyboard) will be displayed (or echoed) on the screen in one color, and output from the computer (e.g. results from computations, graphics, etc) in another color. The cassette tape recorder is used for the storage of programs and data. When the computer is switched off, any programs or data stored temporarily in the computer memory will be lost. If you wish to store the programs or data you have been working on, record them onto a cassette tape.

To connect the TV to the computer, connect the plug end of the video cable into the jack or socket labelled TV PAL (or TV NTSC), at the back of the COMX 35, and the connector end into the antenna input socket (usually located at the back) of your TV set. (An external RF modulator is NOT needed as there is one built-in inside the COMX 35.)



This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type-tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

Reorient the receiving antenna
Relocate the computer with respect to the receiver
Move the computer away from the receiver
Plug the computer into a different outlet so that computer and receiver are on different branch circuits.

If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful:

"How to Identify and Resolve Radio—TV Interference Problems"

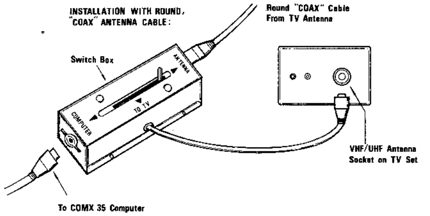
This booklet is available from the U.S. Government Printing Office, Washington, DC 20402 Stock No. 004-000-00345-4.

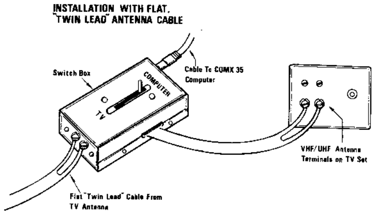
WARNING: THIS EQUIPMENT HAS BEEN CERTIFIED TO COMPLY WITH THE LIMITS FOR A CLASS B COMPUTING DEVICE, PURSUANT TO SUBPART J OF PART 15 of FCC RULES. ONLY PERIPHERALS (COMPUTER INPUT/OUTPUT DEVICES, TERMINALS, PRINTERS, ETC.) CERTIFIED TO COMPLY WITH THE CLASS B LIMITS MAY BE ATTACHED TO THIS COMPUTER. OPERATION WITH NON-CERTIFIED PERIPHERALS IS LIKELY TO RESULT IN INTERFERENCE TO RADIO AND TV RECEPTION.

To connect the cassette recorder, use the cable pair provided. One cable has a red plug at both ends. The other has a black plug at both ends. Connect one red plug to the EAR or EARPHONE or MON or MONITOR jack on the recorder, and the other red plug into the jack at the back of COMX 35 marked CASSETTE IN or EAR. Connect one black plug to the MIC or MICROPHONE jack on the recorder, and the other black plug to the jack marked CASSETTE OUT or MIC. Connect the cassette recorder's power cord into a wall socket, and it is now ready for use. We do not recommend using batteries with the cassette recorder, as they may weaken during playback or recording, producing unacceptable results. In addition, always unplug the cassette cables before disconnecting the power of your cassette recorders.

ClibPDF - www.fastio.com

1.3 Turning the power on


INSTALLATION WITH ROUND, "COAX" ANTENNA CABLE:
Round "COAX" Cable From TV Antenna
Switch Box
ANTENNA
TO TV
COMPUTER
VHF/UHF Antenna Socket on TV Set
To COMX 35 Computer


INSTALLATION WITH FLAT, "TWIN LEAD" ANTENNA CABLE
Switch Box
COMPUTER
TV
Cable To COMX 35 Computer
VHF/UHF Antenna Terminals on TV Set
Flat "Twin Lead" Cable From TV Antenna

Firstly, you MUST check the following:

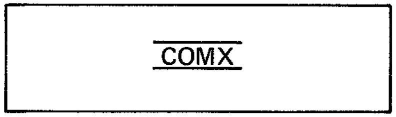
(a) ... make sure that the power on-off switch at the back of the COMX 35 is in the OFF position,

(b) ... check that the power converter voltage rating matches the output voltage of the AC outlet on your wall.

Now, connect the end of the power cord with a plug into the jack at the back of the COMX 35 labelled POWER, and the other end, where the converter is, into the AC outlet. Switch the power switch to the on position. The lighting up of the red POWER light, and the sounding of a few audible notes indicate that COMX 35 is turned on.

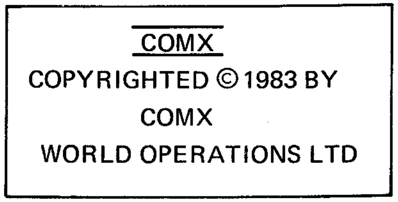
Select and fine tune CHANNEL 36(UHF) for PAL systems or, CHANNEL 3 (VHF) for NTSC systems on the TV (the channel may vary with different local frequencies) and tune the TV until the following multi-colored COMX 35 messages appear. For the adjustment of your COMX 35 computer, refer to Section A.5.

Fig. 1. 3. 1 COMX 35 turn-on messages (to begin computing, just press any key)



COMX

(a)



COMX
COPYRIGHTED © 1983 BY
COMX
WORLD OPERATIONS LTD

(b)

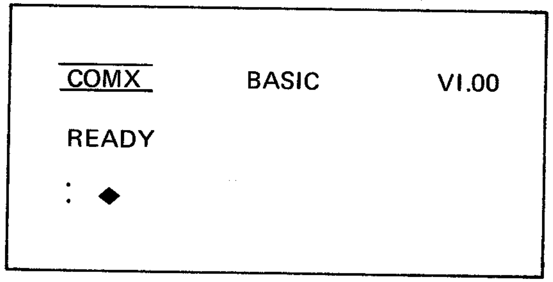
Tuning the TV set to the frequency of the COMX 35 TV output has to be done slowly. As you turn the tuning knob on the TV set, the TV picture disappears, and then the TV gives a "snowy" display. Continue to adjust the tuner such that the "snowy" display suddenly clears up, and the messages in Fig. 1. 3. 1 appears. Now, adjust the contrast, brightness and color controls of the TV to give you a clear, color display. The sound volume of the TV may be turned to the minimum since the sound effects come from a built-in loudspeaker inside the COMX 35 and not from the TV loudspeaker.

The turn-on messages (a) and (b) alternates indefinitely. The colors of message (a) will change. This may be used as a "test pattern" for best color adjustment.

(Note: Before leaving the quality assurance section of the factory, each COMX 35 is adjusted to give the best color. If this has been disturbed, adjustment may be made with a small screw-driver through the tiny holes located at the bottom side of COMX 35. The "test pattern" is intended for such adjustment. If in doubt, consult your dealer.)

To begin a computing session, press any key, except the space bar, and the turn-on messages will be replaced by the following "READY" message.

Fig. 1. 3. 2 COMX 35 "READY" message



COMX BASIC VI.00

READY

: ◆

Note that the screen background is black in color, the computer output on the screen is in cyan (light blue) color, and the diamond shaped "cursor" is in magenta (pink).

The cursor indicates the position of the next character to be printed if a key is pressed.

ClibPDF - www.fastio.com

The colon: is referred to as the "prompt" symbol (or simply the prompt) for the COMX 35 BASIC. It serves to remind the user that the computer is waiting for the user to input something.

1.4 The COMX 35 keyboard

On the COMX keyboard there are 55 keys including the space bar arranged in a way similar to typewriter keys.

The SHIFT KEYS allow for nearly twice as many characters with the same number of keys. If a key is pressed, the lower symbol will be displayed on the screen. If the same key is pressed while the shift key is held down, the upper symbol or a graphic symbol is displayed.

Unlike a typewriter keyboard, the COMX 35 keyboard does not have lower-case letters. Also, the letter "O" and the number zero "0" are differentiated by the presence of a slash in the latter. It is important not to confuse "O" with "0" and "L" with "1".

It is important to understand the functions of those keys not found on a typewriter. At this stage, you may not fully appreciate them, but they are useful and important.

The key marked CR (Carriage Return) causes the cursor to return to the leftmost position of the screen in the following row. (It also sends a message to the computer to signify the end of a command or instruction. See also section 2.1.) The CR key is also referred to as simply the RETURN key.

The RT or RESET resets the computer to the same conditions as when the computer was powered on. Resetting will clear the screen, and cause the computer to output the messages shown in Fig. 1. 3. 1. It should be noted that whenever the reset key and the space bar are pressed simultaneously, everything in memory is cleared. This means that all programs and data previously in memory are lost. Avoid pressing reset unless all wanted programs and data have been saved by recording them onto cassette tapes (see section 3. 13). Again, to start a computing session, just press any key except the space bar. It is to be noted that to use the RT key, it is necessary to type and hold down the "space bar" at the same time. This is to prevent resetting by accident. After pressing these two keys, and not until the keys are released, will the COMX 35 display the start-up message.

ClibPDF - www.fastio.com

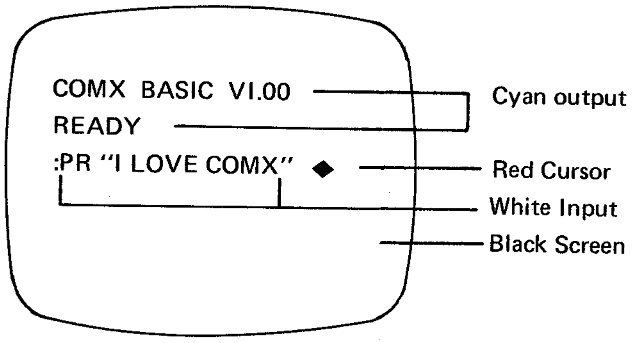
The ESC or ESCAPE Key enables you to stop the execution of a program (to "escape" from a program). When the ESC key is pressed, the message "READY" is displayed showing that the computer is ready for a new BASIC command.

Pressing the CNTL or CONTROL key by itself does not put any character on the screen or do anything else. If the CNTL key is held down and another appropriate key is pressed, the computer responds by performing certain actions or printing certain characters. For example, pressing C while the CNTL key is held down will cause the computer to ignore the line being typed (see section 2. 1). The cursor will move to the leftmost position on the next line. This function is useful if a user makes a mistake in a line and wishes to start again on the next line. The functions of the CNTL keys are further discussed in sections 2.1 and 3.15.

1.5 The COMX 35 screen

When power is first turned on, the computer will output characters in cyan (light blue) for computer generated messages or computations and the keyboard input characters will be displayed in white against a black screen (background) color. This assumes that you are using a color TV and the color/brightness/contrast controls are properly adjusted. The screen displays 24 lines with 40 characters per line.

As will be shown in section 2.3, the colors of the screen and characters can be changed using COMX 35 BASIC commands.



```
COMX BASIC VI.00
READY
:PR "I LOVE COMX" ◆
```

CHAPTER TWO

GETTING FAMILIAR WITH SOME BASIC COMMANDS

ClibPDF - www.fastio.com

CHAPTER 2 GETTING FAMILIAR WITH SOME BASIC COMMANDS

At the end of this Chapter, you will be able to:

(a) ... use the COMX 35 as if it is an advanced calculator,

(b) ... use the more common "BASIC" commands,

(c) ... know what to do if mistakes are made, and how to correct them,

(d) ... use the "BASIC" commands for creating color graphics and for making music,

Using the COMX 35 to do the job of a calculator is certainly under-utilizing the immense power of the COMX 35, but it is a good way to learn the BASIC language. You can use this mode of operation to do fairly complex calculations or help your kid brother or sister to check his or her arithmetic homework. When you are familiar with the BASIC commands, you can then use a sequence of these commands, i.e. a program, to solve a complex problem requiring many steps. Thus, this Chapter prepares the groundwork for the following chapters on programming.

ClibPDF - www.fastio.com

2.1 Using the COMX 35 to print a message

Turn on the computer to obtain the welcome message. Refer to section 1.5 if you encounter difficulties.

Type on the keyboard,

```
PRINT "HI COMX 35"
```

followed by pressing the key marked CR. Remember to hold the shift key down when you type the quotes ("). Pressing CR causes the cursor to move to the left margin of the screen. Pressing CR also tells the computer that you have come to the end of your command. The screen should look as follows:

You type	PRINT "HI COMX 35"
COMX 35 responds	HI COMX 35

COMX 35 SUMMARY

To print a message, use the command PRINT followed by the message included in a pair of quotes. End the command by pressing the key marked CR.

You need not type PRINT, using the abbreviation PR will have the same effect.

Typing PR followed by pressing the CR key will move the cursor to the next line. This is called a "newline" operation.

COMX 35 footnote: Correcting mistakes or Editing

If you discover a typing mistake BEFORE you type CR:

(1) ... Pressing the DEL (for delete) key will move the cursor backward (called backspacing), and the character moved over will also be deleted. To make corrections, delete wrong characters by the DEL key and retype.

(2) ... Alternatively, hold the CNTL (for control) key down while pressing C. This has the effect of moving the cursor to the leftmost position on the next line. This makes the line (containing the errors) invalid, and the COMX 35 will ignore this line.

Try typing,

```
PRINT "TEST"
```

but before pressing CR, hold the CNTL key down while pressing C. Note that no message is printed since the incomplete statement is simply ignored.

If you discover a mistake AFTER you have typed CR:

If you misspell the word "PRINT", you will get the following error message:

```
ERR CODE 30
READY
: ◆
```

The error message with code number 30 means "Unacceptable last character in PRINT statement". The presence of : indicates that no harm is done, and you can retype your command. To understand why this happens, we can examine an error producing print statement such as PRNNT "HI", where an "N" has by mistake been typed instead of an "I". The computer will still recognize the first two letters as "PR" which has the same meaning as "PRINT". The computer then interprets the "NNT" as a variable and the "HI" as a literal, but there is no separator such as a comma or semicolon between them. It is the lack of a separator that cause the error.

If you miss the first or the last quote, the computer will print:

```
ERR CODE 22
READY
: ◆
```

ClibPDF - www.fastio.com

The error message with code number 22 means "missing quote".

Other error messages and their meanings are given in the Appendix.

If you did not get any response, perhaps you forget to end your command by pressing CR.

Examples

Now, try the following commands to see if you get the corresponding outputs:

you type→
```
PRINT "MONEY CAN'T BUY ME LOVE – BEATLES"
MONEY CAN'T BUY ME LOVE – BEATLES
```

you type→
```
PRINT "TO ERR IS HUMAN, TO REALLY MESS THINGS
UP YOU NEED A COMPUTER"
TO ERR IS HUMAN, TO REALLY MESS THINGS UP
YOU NEED A COMPUTER
```

you type→
```
PRINT "A CHILD MISEDUCATED IS A CHILD LOST –
JOHN F. KENNEDY"
A CHILD MISEDUCATED IS A CHILD LOST–JOHN F.
KENNEDY.
```

Repeat some of the above examples using PR instead of PRINT and note that the computer responses are identical.

2.2 Using the COMX 35 as a calculator

The COMX 35 can be used as a super calculator.

Try this on the COMX 35

```
PRINT   5 + 6
```

and press CR. The COMX 35 will respond with:

```
11
```

The COMX 35 can do 5 arithmetic operations:

(a) ... ADDITION, indicated by +.

(b) ... SUBTRACTION, indicated by –. Try,

```
PRINT   34 – 16
```

to get 18. Don't forget to press CR at the end of your comand.

(c) ... MULTIPLICATION, indicated by *.

Try →
to get →

```
PRINT  7 * 8
56
```

(d) ... DIVISION, indicated by /.

Try →
to get →

```
PRINT   56/7
8
```

(e) ... EXPONENTIATION. It is sometimes necessary to multiply a number by itself for a given number of times.

Try →
to get →

```
PRINT   2 * 2 * 2 * 2 * 2
32
```

However, instead of writing the above, you can represent the same by the following:

ClibPDF - www.fastio.com

Try → to get →

```
PRINT  2↑5
32
```

The upward arrow ↑ is typed by pressing the key ≙ while holding down the shift key. When computer languages like "BASIC" were being designed, the authors decided to use special symbols for certain functions, so that the user would not accidentally use the wrong keys, and these special symbols had to be found on an ordinary computer keyboard like the COMX 35. Thus, the X in (3 X 5) became *, so that multiplication would not be confused with the letter "X". Division became "/" instead of " ÷ " as there is no " ÷ " on a regular keyboard. The "+" and "–" symbols were left untouched. Since exponentiation is shown in an expression by character placement such as "10^2", and most computer displays cannot show "super-script" characters, a symbol is required. The "↑" was chosen as it points up showing that the next character is a superscript or an exponent (e.g. 10^2 becames 10 ↑ 2).

All of the above examples involve two numbers and one operation. The COMX 35 can also solve more complicated problems involving many numbers and many different operations.

= shift tegelijk met (

Try → to get →

```
PRINT  (5 + 6) * (9 – 7)
22
```

Try → to get →

```
PRINT (7 – 4) * (19 – 14) /
(8 – 3)
3
```

Try → to get →

```
PRINT (3 ↑ 3 + 2 ↑ 4) / (2 * 5)
4.3
```

The computer will do certain operations first before others e.g. like the ordinary rules of arithmetic, it will do * and / first before + and –. A complete set of similar rules are given in section 5.3. At this stage, use parentheses "(   )" as often as you like to tell computer which operations are to be done first.

COMX 35 SUMMARY

The COMX 35 can solve complex arithmetic problems involving +, –, *, / and ↑ . Always press CR at the end of your comand. Use the computer notations * and / instead of X and ÷ . Use parentheses wherever necessary. For more detailed information, see section 5.3.

ClibPDF - www.fastio.com

2.3 Changing colors

The COMX 35 screen display is multi-colored. The keyboard input of the user is displayed (or echoed) in white, and the computer output and error messages are in cyan (a light blue). The diamond shaped cursor is in magenta (pinkish). The overall effect is to increase readability, and to facilitate "dialogue" between the user and the computer and generally enable the user to program faster with fewer errors. These features are especially important for first time users. The color combination described above is the "default" scheme, i.e. it is the scheme normally used by COMX 35 unless it is specifically instructed to use other colors. Other color schemes are possible through the use of the COLOR, SCREEN and CTONE commands described below.

2.3.1 Experiment with the COLOR command

The COLOR command is in the form,

```
COLOR (X)
```

Where X is a numeric expression which determines the color combination to be used for user input and computer output characters. The color combinations for different values of X are given in Table 2.3.1.

Now, type the command

```
COLOR (10)
```

followed by pressing CR, and note that user input characters have turned magenta and the computer outputs have turned blue. Try some other color commands with the "10" replaced by any number from 1 to 12 inclusive. Observe the color changes and compare them with Table 2.3.1. Note that COLOR (12) corresponds to the default character colors.

Table 2.3.1 COLOR Schemes for COLOR (X)

X	COMPUTER RESPONSE		KEYBOARD INPUT	
1	BLACK	(0)	GREEN	(59)
2	RED	(30)	YELLOW	(89)
3	BLUE	(11)	CYAN	(70)
4	MAGENTA	(41)	WHITE	(100)
5	BLACK	(0)	BLUE	(11)
6	RED	(30)	MAGENTA	(41)
7	GREEN	(59)	CYAN	(70)
8	YELLOW	(89)	WHITE	(100)
9	BLACK	(0)	RED	(30)
10	BLUE	(11)	MAGENTA	(41)
11	GREEN	(59)	YELLOW	(89)
12	CYAN	(70)	WHITE	(100)

NOTE: The bracketed numbers are the luminances in %.

2.3.2 Experiment with the SCREEN command

The SCREEN command which is in the form,

```
SCREEN (X)
```

where X is a numeric expression (varying from screen 1 to 8 inclusively) which determines the color of the background. The different screen colors corresponding to different values of X are given in Table 2.3.2.

Now, try the command

```
SCREEN (2)
```

ClibPDF - www.fastio.com

followed by pressing CR, and note that the screen color changes from black to green. Replace the "2" by numbers from 1 to 8 inclusive, and compare the screen color observed with Table 2.3.2. Note that SCREEN (1) corresponds to the default screen color. Note also that if either the input or output characters have the same color as the screen, the characters would not be visible.

The brightnesses or luminances of different COMX 35 colors are also given in Table 2.3.1. In general, the best result is obtained with bright characters against a dark background (e.g. COLOR (12) and SCREEN (1), or COLOR (12) and SCREEN (3)); dark characters against a bright background (e.g. COLOR (5) and SCREEN (2)) may also be used; COLOR (12) and SCREEN (7) is also a pleasing combination.

TABLE 2.3.2 Screen color for SCREEN (X)

X	Color	
1	BLACK	(0)
2	GREEN	(59)
3	BLUE	(11)
4	CYAN	(70)
5	RED	(30)
6	YELLOW	(89)
7	MAGENTA	(41)
8	WHITE	(100)

Note: The bracketed numbers are the luminances in %.

2.3.3 The CTONE command

This is in the form,

```
CTONE (X)
```

where the numeric expression X is either zero or non-zero. If X is non-zero, the color-tone effect is turned on such that the screen color remains unchanged but the color of the characters displayed is similar to the screen color with a different degree of BRIGHTNESS. If X is zero, the color-tone effect described above is turned off.

2.4 Producing sound effect

In the COMX 35, high quality sound effect is generated by a built-in speaker. The "music command" produces the 7 music notes over 8-octave frequency ranges, and 16 volume levels. The "noise command" produces Gaussian white noise over 8 frequency ranges, and 16 volume levels. In addition, there is the "tone command" which is like the music command (i.e. over 8 octaves and 16 volume levels) except that for each octave, instead of the 7 music notes, the frequency may be varied in 128 steps. Using suitable combinations of these three commands, a musical tune can be synthesized, and realistic sound effects can be generated to synchronize with pattern and color changes. As these are all programmable, the COMX 35 output display is therefore not just numbers and texts, but can include animated pictures accompanied by music and sound.

2.4.1 Experiment with the MUSIC command

The MUSIC command is in the form,

```
MUSIC (X, Y, Z)
```

where X, Y and Z are numeric expressions. X determines the music note and can vary from 1, 2 ... to 7. Thus

ClibPDF - www.fastio.com

1 = DO
2 = RAY
3 = MI
4 = FA
5 = SO
6 = LA
7 = TI

Y determines the Octave and can vary from 1, 2, ... to 8 with 8 being the octave with the highest upper frequency.

Z determines the amplitude and can vary from 0, 1, to 15 with 15 being the loudest.

Now, try

```
MUSIC (1, 1, 1)
```

followed by pressing CR to get a weak "DO" over the lowest octave.

Try

```
MUSIC (1, 1, 2)
```

and note the increase in volume.

Try

```
MUSIC (2, 1, 2)
```

and note the change to a "RAY".

Try

```
MUSIC (2, 2, 2)
```

and note the change to a higher octave.

Notice that once a music command is issued, the note will continue until it is replaced by another music command. To turn-off the music note, type

```
MUSIC (0, 0, 0)
```

Of course, always remember to terminate a command by pressing CR.

2.4.2 Experiment with the NOISE command

The NOISE command is in the form,

```
NOISE (Y, Z)
```

Y determines the frequency range of the Gaussian white noise, and can vary from 1, 2, ... to 8.

Z determines the amplitude and can vary from 0, 1, 2,... to 15.

Try

```
NOISE (1, 1)
```

followed by pressing CR, and note the noise effect.

Try

```
NOISE (1, 2)
```

and note the increase in loudness (or amplitude).

Try

```
NOISE (2, 2)
```

and note the increase in pitch (or frequency).

To turn-off the noise operator, type

```
NOISE (0, 0)
```

Again, always remember to terminate a command by pressing CR.

ClibPDF - www.fastio.com

Gaussian white noise, is noise that can be heard between stations when turning an "FM" radio. This type of noise, which contains many different frequencies is very useful in producing sound effects.

2.4.3 Experiment with the TONE command

The TONE command sends out a continuous tone. It is in the form,

TONE (X, Y, Z)

where X determines the frequency, and can vary from 1,2, ... to 128.

Y determines the octave, and can vary from 1, 2, ... to 8.

Z determines the amplitude and can vary from 0, 1, ... to 15.

Table 2.4.3 shows the relationship between the actual output frequencies and the parameter X in the TONE command. This table refers to the 4th octave (i.e. Y = 4). Frequency will be doubled for a higher octave and halved for each lower octave.

Table 2.4.3

Parameter X	Freq. (Hz)	Music note	Parameter X	Freq. (Hz)	Music note
6	482	B	9	452	$B^b/A^\#$
2	426	A	4	410	$A^b/G^\#$
8	382	G	1	363	$G^b/F^\#$
5	341	F	9	321	E
3	303	$E^b/D^\#$	7	287	D
2	270	$D^b/C^\#$	6	257	C

2.4.4 The VOLUME command

This is in the form,

VOLUME (X)

where the integer X varies from 1 (softest) to 4 (loudest). This "MASTER VOLUME CONTROL" command will affect all MUSIC, NOISE and TONE commands issued after it. All sound commands preceding the VOLUME command will not be affected.

2.5 Using the cassette recorder for storing programs and data

2.5.1 Transferring a program from the computer to the recorder using PSAVE

Firstly, connect up the cassette recorder for subsequent use as described in section 1.2.

Let's assume that you have a program in the computer memory which you wish to save for future use. Type LIST to have a quick check at the program in memory to see that there are no unwanted fragments of a program. Adjust the recorder's volume control to an appropriate setting. For the COMX 35,setting the volume to mid-level will usually work. Now, proceed to do the following:

(1) ... Insert a blank cassette tape into your recorder and rewind the tape. (If your recorder is provided with a tape position counter, set it to zero. Recording on used tape will of course erase information previously recorded.)

(2) ... Start recording on the tape by pressing both "PLAY" and "RECORD" down.

(3) ... Type PSAVE for "program save" and press CR to start transferring the program onto tape.

(4) ... When (3) is finished, the message "Ready" followed by ": ◆ " reappears. Remember to note the tape position corresponding to the end of the recorded program for future reference. Step (3) is further detailed below. When PSAVE is keyed in followed by CR, the cursor should disappear, and a long high-pitch note marking the beginning of the program (called the program header) can be heard.

ClibPDF - www.fastio.com

We should start the cassette recorder first to ensure that the blank "leader" tape will not be recorded on. This leader tape, used in most commercial tapes will not record any signals and thus part of your program or data could be lost if we started to recorder right from the start.

Then, a noisy note follows, indicating the transfer of the first page (256 bytes) of the program. This is followed by a shorter high-pitch note, the page header, marking the beginning of the second page, and then another noisy note indicating the transfer of the second page of program. The high-pitch page header and the noisy note alternate until the entire program is transferred. The end of the transfer is marked by a high-pitch note followed by a low-tone. By getting accustomed to the above sequence of sound, the user can tell whether normal transfer is taking place.

The COMX 35 is so designed that the sound signal is sent out by the computer to the recorder via the cable connected to the two MIC jacks. The signal goes through the recorder and returns to the computer via the cable connected to the two EAR jacks. If there is a wrong and/or loose connection, or if the recorder is not turned-on, the sound signal path is disrupted and cannot be heard. This serves to alert the user to check the connections.

* It is to be noted that not all recorders will produce the above sound sequence as they record. If this is the case, do not worry, you can still use your recorder with the COMX 35, but no sound will be heard.

2.5.2 Saving Data Using DSAVE

To save data instead of program, proceed with (1) and (2) above except that for (3), instead of PSAVE, use DSAVE for "data save". The content of that part of the computer memory reserved for data (numbers, strings and/or arrays) will be stored on the tape.

The same sound sequence as described in the previous section indicates proper functioning of the loading process.

2.5.3 Transferring programs from the recorder to the computer using PLOAD

To transfer or load a computer program stored on a cassette tape into the computer memory, firstly, connect up the recorder as described in section 1.2. Then proceed as follows:

(1) ... Insert the cassette tape into your recorder, and locate the beginning of the program.

ClibPDF - www.fastio.com

(2) ... Make sure that any program and/or data now in memory is no longer wanted, or has been saved. This is because step (3) will clear the entire user memory and any program and/or data in memory will be lost.

(3) ... We assume that the computer is powered up, and the "READY" message followed by ": ◆ " is displayed. Now, type PLOAD, and press the PLAY button. When a high pitch note is heard, press the CR key to start loading the program. The cursor will disappear, followed by the sequence of sound described in section 2.5.1.

(4) ... When the loading of the program is completed, the message "READY" followed by ": ◆ " will reappear.

2.5.4 Loading Data Using DLOAD

The DLOAD (for data load) command loads from tape any previously stored data (resulting from a DSAVE). The procedure is similar to that described in section 3.13.3 except that DLOAD is used instead of PLOAD. The data is automatically placed at the end of existing program memory space and overwrites any existing data.

2.5.5 Further Points Related to DLOAD

In connection with the use of the DLOAD command, three points must be considered:

(1) ... If the user memory space contains both a program and data, any editing of the program will wipe out the data. Therefore, before any editing is done, do a DSAVE to save the data (if desired), edit the program, and do a DLOAD to return the data.

(2) ... Note that the act of doing a DLOAD automatically dimensions any arrays within the stored data.

(3) ... Strings are placed at the end of array space. If additional arrays are dimensioned after the strings have been generated, the strings will be wiped out by the growing array space. For this reason, all array dimensioning should be done before any strings are generated.

2.5.6 Further tips on saving or loading programs on tapes

In order to prevent the loss of valuable programs and data, the following points ought to be observed:

(1) ... Save more than one copy by repeated use of PSAVE and DSAVE, and for programs representing many hours of work, save the same in another tape.

(2) ... Listen to the sound sequence which provides a fair indication of a successful transfer.

(3) ... After step (1), load the saved program back to the computer and LIST. If loading cannot be completed successfully, the original program may remain intact and attempt to save the program may be repeated.

(4) ... Handle the tapes carefully to aviod scratches. Also, avoid putting the tapes near strong magnetic fields. Keep the tapes away from the TV. See also Appendix A.3 on "Twenty important cassette recording guidelines."

ClibPDF - www.fastio.com

CHAPTER THREE

WRITING SIMPLE PROGRAMS

ClibPDF - www.fastio.com

CHAPTER 3 WRITING SIMPLE PROGRAMS

3.1 Your first COMX 35 BASIC program

We shall introduce you to programming by the use of examples. You are invited to type in sample programs, and then observe the computer response. The first program is intended to show you the main structure of a program, and the use of certain BASIC commands. Type the followings:

PROGRAM 3.1

```
:  NEW
:  10      PR      "HI COMX 35"
:  20      PR      2 + 3                    The program
:  30      PR      (5 + 6) * (9 – 7)
:  40      END

:  RUN                                      The RUN command

:  HI COMX  35                              Computer response
   5
   22
   READY

:  ◆
```

The screen display is divided into three main parts,

(i) the program itself

(ii) the RUN command telling the computer to obey or run the program instructions, and finally

(iii) the output by the computer resulting from obeying the program instructions.

Let's examine further the program itself.

NEW is a command (or statement) telling the computer to erase all programs or data previously entered.

ClibPDF - www.fastio.com

Each line of instruction in the program begins with a line number, 10, 20 etc. The computer will obey the instructions in sequence i.e. in the order of the line numbers, that is from the smallest to the largest number.

If you examine instructions 10, 20 and 30, you may notice that you have met them before in Chapter 2. However, in this case, the instructions are obeyed as a group (after you type RUN), instead of being obeyed one instruction at a time. This is the main difference between using COMX 35 as a computer to run programs and using COMX 35 as a calculator.

The command END indicates the end of a program. When the computer encounters END, it terminates program execution.

RUN is a command to tell the computer to begin program execution. The computer searches for the lowest line-number and begins executing each line in numerical order.

COMX 35 Footnote: More On Line-Numbers

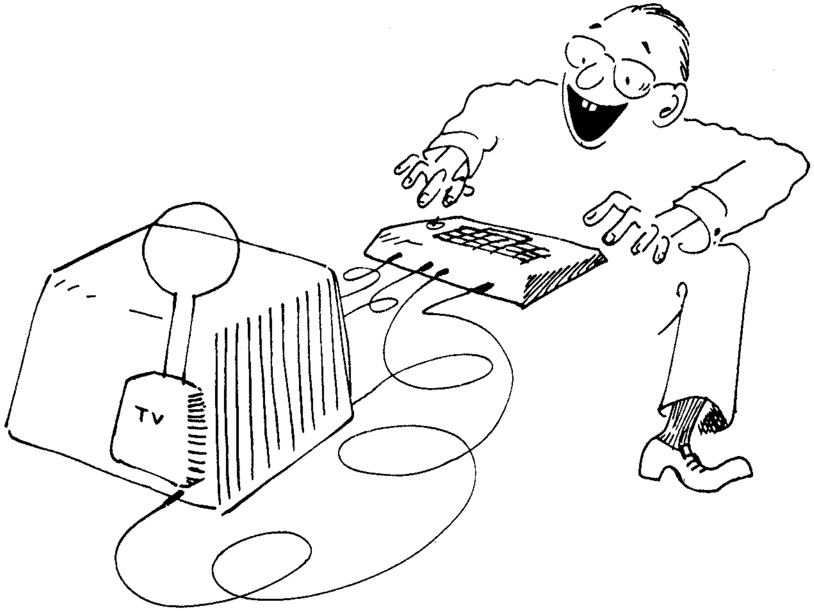
COMX 35 recommends you to begin your program with line-number 10 and goes up to 20, 30, 40 ... etc, i.e. leaving gaps for 9 more instructions. This will allow you to insert extra lines without the need to change the original line numbers. If you add line 15, this instruction will be obeyed before line 20.

When you begin an instruction with a line-number, you are in fact telling the computer not to execute it until you type RUN. You are using the computer in the program mode. If you do not use any line-numbers, as in Chapter 2, execution will begin as soon as you type CR. You would then be in the calculator mode.

The line-number also serves as the name (or label) for the instruction. Line-numbers are also used in GOTO statements and in program editing. See sections 3.2 and 3.3.

COMX 35 Summary: What is a program?

A program is a group (or a sequence) of instructions each beginning with a line-number. The computer will normally obey (or execute) these instructions in sequence i.e. in numerical order of the line-numbers. The command NEW at the beginning of the program will erase all previous programs, and the command END causes the computer to terminate execution. The command RUN tells the computer to begin execution starting with the instruction with the lowest line-number. Commands are often referred to as statements. Thus, we speak of the statement END, the statement RUN etc.



ClibPDF - www.fastio.com

3.2 Teaching the COMX 35 to jump (using the GOTO statement)

As explained in section 3.1, the computer will normally obey instructions in sequence. However, the computer may be told to obey instruction out of sequence using the GOTO statement. Try inserting the statement "15 GOTO 30" into the program in section 3.1 you should get,

PROGRAM 3.2

```
NEW
10      PR       "HI COMX  35"
15      GOTO     30
20      PR       2 + 3
30      PR       (5 + 6) * (9 – 7)
40      END
```

Additional GOTO statement

RUN

```
HI COMX  35
22
```

Note: Statement 20, and hence the output "5" has been skipped.

COMX 35 summary: GOTO Statement

The statement "GOTO n" tells the computer to execute the statement with the line-number n next, instead of the one immediately following. The statement is said to be an "unconditional branch" since the computer is told to jump or branch off to another statement always, under all conditions i.e. unconditionally. If the line-number n does not exist, an error message is generated. "n" can be an expression. Thus, "GOTO A–B" and "GOTO 50 * (A + B)" are valid statements.

3.3 Displaying and editing a program

The following session assumes that you have entered PROGRAM 2 in section 3.2 into the computer. If you have not, do so now before proceeding.

To tell the computer to display a program, try,

LIST

followed by CR and the computer will respond by displaying the latest program entered, in this case, program 2 as follows:

```
10      PRINT      "HI COMX  35 "
15      GOTO       30
20      PR         2 + 3
30      PR         (5 + 6) * (9 – 7)
40      END
```

To replace a line, type the line-number of the line to be replaced, followed by the new instruction. The following replaces line 15. Try,

```
15      GOTO       40
LIST
```

The computer will respond by:

```
10      PRINT    "HI COMX 35"
15      GOTO     40
20      PRINT    2 + 3
30      PRINT    (5 + 6) * (9 – 7)
40      END
```

This replaces the old line

To delete a line, type the line-number followed by CR. The following will delete line 15. Try,

```
15
LIST
```

A line-number followed by CR

The computer will respond by:

```
10      PR       "HI COMX  35 "
20      PR       2 + 3
30      PR       (5 + 6) * (9 – 7)
40      END
```

ClibPDF - www.fastio.com

To insert a line between line 20 and line 30, choose a line-number, say 25, and type,

```
25      GOTO     40
LIST
```

The computer will respond by:

```
10      PR       "HI COMX  35 "
20      PR       2 + 3
25      GOTO     40
30      PR       (5 + 6) * (9 – 7)
40      END
```

The new line has been inserted

COMX 35 footnote: More on LIST

```
LIST
LIST    n
LIST    n,    m
```

The statement LIST (with nothing following) will cause the computer to display or list the entire program. "LIST n" will only list line numbered n. "LIST n, m" will start listing at the line n and end at the line m inclusive. Both "n" and "m" may be an arithmetic expression.

If, at any time, the expression equals a number which is a non-existent line number, then the nearest line number after will be listed.

COMX 35 summary: Editing a program

To edit a program is to correct or to modify it. We can replace a line, delete a line or insert a line by making use of the line numbers. Always LIST the program, or that part of the program which has been edited, to make sure that the expected change has been made (see also section 3.11).

3.4 Write a program which will accept data from the keyboard (using the INPUT statement)

Beginning with the last program in section 3.3, try

```
 5      INPUT     A, B       Insert the new INPUT statement
10                           Delete line 10
20      PRINT     A+B
25                           Delete lines 25 and 30
30
40      END
```

```
LIST
```

After the editing, the computer will respond by

PROGRAM 3.4

```
 5      INPUT     A,B
20      PRINT     A+B
40      END
```

Line 5 is the input statement and A and B are the "not yet specified" or "unknown" quantities called the variables. When the computer encounters the input statement, it stops and outputs a question mark "?" on the screen. It then waits for the user to respond. The user should type in as many numbers as there are variables. These numbers specify the values of the "unknown quantities" or "variables".

For A = 2, B = 3, try

```
RUN
?        2,3        User input data for A and B in
                    response to "?"
5                   Computer output, equal to A+B.
```

For A = 1012, B = 4517, try

```
RUN
?        1012,    4517      Another set of data for A and B.
5529                        A+B
```

ClibPDF - www.fastio.com

You have now a program which will give you the sum of two numbers when the values for the two numbers are supplied.
Try other values of A and B.

Try Now, replace line 20 by a more complicated expression.

```
20     PRINT     (A * A + B * B) ↑ 0.5
LIST
```

The program becomes

```
 5     INPUT     A, B
20     PRINT     (A * A + B * B) ↑ 0.5
40     END
```

Which when executed, gives

```
RUN
?      3,4
5
```

Some of you may recognize that the above program is an application of Pythagoras' Theorem which you are most likely to come across in your geometry lessons. If you wish to know more about the Theorem, please read the following footnote. Meanwhile, the important point is to note the supreme usefulness of these 3-line program.

Line 5 can be modified to accept data for more than two variables. Line 20 can be replaced by a complicated algebraic expression of these variables. When the program is executed, and values are assigned to the variables in response to "?", the computer will output the result of the expression. You can repeat with other sets of values without the need to type in the expression again. You input the data, the computer will execute the program to give you the answer. This is the essence of power of a computer program, and may be illustrated as follows:

Input Data
Computer Program
Output Answer

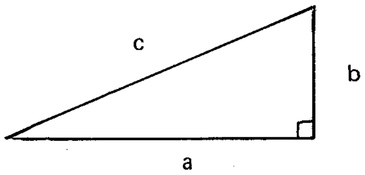
ClibPDF - www.fastio.com

When the computer is used without the line-numbers (as in Chapter 2, in what is called the calculator mode), you have to provide the computation steps for every set of data.

COMX 35 footnote: Pythagoras' Theorem

If the hypotenuse (or slanting side) of a right-angled triangle is of length c, and the lengths of the two other sides are a and b, as shown, according to Pythagoras' Theorem,

$$c = (a^2 + b^2)^{½}$$



i.e. given a and b, c may be calculated as above. To do so using BASIC, we have to rewrite the algebraic expression on the left into a BASIC expression i.e. (A*A+B*B) ↑ 0.5. The following section gives more examples of how to rewrite an algebraic expression into a BASIC expression.

3.5 Algebraic expressions and BASIC expressions

You are probably familiar with certain algebraic formulas e.g. the formulas for calculating the circumference and area of a circle. However, since the computer can only recognize BASIC expressions, these algebraic formulas would have to be rewritten into the equivalent BASIC expressions before being input into the computer. For example, "ab", "a ÷ b", "a^b" would have to be written as "a * b", "a/b" and "a ↑ b" respectively. The following table gives more examples:

Table 3.5 Converting algebraic expressions into BASIC expressions

Description	Algebraic expression	BASIC expression
1. Area of a circle, radius R	πR^2	PI*R ↑ 2
2. Circumference of a circle, radius R	$2\pi R$	2*PI*R
3. Area of a triangle, given height H, and base B.	1/2 HB	H*B/2
4. Area of a triangle, given three sides A, B, and C and S=1/2(A+B+C)	$\sqrt{S(S-A)\ (S-B)\ (S-C)}$	S = (A+B+C)/2 (S*(S–A)*(S–B) *(S–C)) ↑ 0.5
5. The hypotenuse C of a right-angled triangle given the other sides A and B	$C=(a^2+b^2)^{\frac{1}{2}}$	C = (A*A+B*B) ↑ 0.5 or C=(A ↑ 2 + B ↑ 2) ↑ 0.5

6. The solutions for X, given a quadratic equation, $AX^2 + BX + C = 0$	$X=\dfrac{-B\pm\sqrt{B^2-4AC}}{2A}$	X=(–B+(B*B–4*A *C) ↑ 0.5)/(2*A)
		and
		X=(–B–(B*B–4*A*C) ↑ 0.5)/(2*A)

7. "Principal plus interest" at a yearly compound interest rate of R%, Principal P and after n years	$P\ (1+R)^n$	P*(1+R)↑n

Beginning with the 3-line model program 3.4 of section 3.4, and the rightmost column of the Table 3.5, try writing programs for some or all of the seven problems.

3.6 Getting the COMX 35 to make decisions

Type the following program for computing the area of a triangle, RUN, and input 3 sets of data as follows:

PROGRAM 3.6

```
NEW
10        INPUT A, B, C
20        If A < 0      THEN GOTO 50
25        S = (A + B + C)/2
30        PRINT (S*(S–A)*(S–B)*(S–C)) ↑ 0.5
40        GOTO  10
50        END
RUN
?    7, 8, 9
26.8328
?    12, 14, 17
83.0267
?    –1, 2, 3
READY
:    ◆
```

When the computer encounters the "GOTO 10" statement in line 40, it will always branch back to statement 10 and ask for another set of input data. This can go on indefinitely (provided your data is correct, e.g. any two sides of the triangle is greater than the third side), and with this little program, you can compute the area of all the triangles in the world! The problem is how to tell the COMX 35 to stop! That is the purpose of line 20. This new statement asks the computer to examine the value of A, and if A is less than zero, then branch to 50 to terminate execution. This GOTO statement is obeyed only if a certain condition (in this case, A< 0) is true. This "IF ... THEN GOTO" statement is called a conditional branch i.e. branch if certain condition is true.

We can also say that the COMX 35 is making a decision as to what to do next depending on certain conditions.

ClibPDF - www.fastio.com

COMX 35 footnote: IF statement

The symbols,

>	greater than
<	less than

are called relational operators. Other relational operators are,

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

In many versions of BASIC, the "IF ... THEN GOTO" statement is the only form of the "IF" statement. In COMX 35 BASIC, however, "IF ... THEN" may be followed by a group of statements (separated by colons). This is a very important advantage as the flexibility helps to make the program easier to be understood, or more "structured".

Line 25 is an "assignment" statement. The value resulting from evaluating the right-hand-side expression is given or assigned to the variable on the left.

Using the second form of the IF statement, the above program can be rewritten as follows, but giving identical results.

```
NEW
10        INPUT     A, B, C

20        IF A > 0        THEN     S = (A + B + C)/2:
          PRINT (S*(S–A)*(S–B)*(S–C))↑ 0.5: GOTO 10

30        END
```

Line 20, consisting of many statements separated by colons, will be executed only if A > 0, i.e. if A is greater than 0. If A < 0, line 20 is skipped, and line 30 terminates execution.

3.7 Forming good programming habits

Let's improve program 3.6. Type the following lines.

PROGRAM 3.7

```
NEW
1      REM THIS IS A PROGRAM TO COMPUTE THE AREA
       OF A TRIANGLE
5      PRINT "THIS PROGRAM COMPUTES THE AREA OF
       A TRIANGLE"
10     INPUT "PLEASE INPUT THE LENGTH OF THE SIDES
       A, B, C" A, B, C
20     IF A < 0 THEN GOTO 120
30     S = (A+B+C)/2
```

ClibPDF - www.fastio.com

```
40    IF S < A THEN GOTO 100
50    IF S < B THEN GOTO 100
60    IF S < C THEN GOTO 100
70    T = (S* (S–A) * (S–B) * (S–C)) ↑ 0.5
80    PRINT "THE AREA OF THE TRIANGLE =";T
90    GOTO 10
100   PRINT  "INCORRECT  DATA:  TWO  SIDES  OF  THE
      TRIANGLE NOT LONGER THAN THE THIRD"
115   GOTO 10
120   PRINT "END OF SESSION" : END
RUN
```

```
THIS PROGRAM COMPUTES THE AREA OF A TRIANGLE
PLEASE INPUT THE LENGTHS OF THE SIDES
A, B, C? 12, 13, 14
THE AREA OF THE TRIANGLE = 72.3079
PLEASE INPUT THE LENGTH OF THE SIDES
A, B, C? 12, 13, 26
INVALID DATA: TWO SIDES OF THE TRIANGLE NOT
LONGER THAN THE THIRD
PLEASE INPUT THE LENGTHS OF THE SIDES
A, B, C? –1, 2, 3
END OF SESSION
```

Let's examine what you have done.

Line 1 beginning with the keyword REM (short for REMARK) which allows you to insert after it any remarks to explain what the program is doing or how it works. REM statement may be used liberally anywhere in a program. It is listed (by LIST) but is ignored by the computer during program execution. It is intended for the reader of the program more so than for the computer.

Line 10 is similar to the INPUT statement previously used except that the pair of quotes allows you to insert a message which will be printed before the question mark "?" prompts you to input data.

Similarly, line 80 is similar to the PRINT statement previously used except that the message within quotes is followed by a variable T separated by the semi-colon. This allows you to insert a message to explain the meaning of T.

Lines 40, 50 and 60 cause the error message in line 100 to be printed if the data is incorrect. Such is the case of the second set of input data where 12 + 13 < 26.

Compare program 3.7 with program 3.6, the former is seen to have many more words, messages and explanations (referred to as "documentation"). These are intended to make both the program and the output more meaningful and readable. For a short program, and when the thinking behind the program is still fresh in your mind, such "documentation" may not seem necessary. For a long program, and for somebody who picks up a program for the first time, or even for yourself after a few weeks, "documentation" would be much appreciated. Now that you are budding programmer, forming good habits by fully documenting your program with "REM messages" is most desirable.

3.8 Simple programming exercises with solutions

Remember our motto? "Learn programming by doing"? We are convinced that after you have followed this manual and try all steps on the COMX 35, you will have much appreciation of what computing is about, and will be in a better position to understand more formal textbooks. "Thorough understanding comes from experience". The following exercises are intended to provide you with opportunities to gain more experience.

Compound Interest

Statement of the problem

Given, P = the principal or the original amount ($)
R = the annual interest rate(%)
Y = number of years
T = the number of times in a year that interest is compounded e.g. if compounded daily,
T = 365

You are requested to write a program to output,

F = the principal plus interest, or the final amount ($)

ClibPDF - www.fastio.com

Suggestion

You may wish to write your own program before you look up the solution. You learn more that way.

Solution

```
10    REM        THIS PROGRAM COMPUTES THE PRINCIPAL PLUS
20    REM           INTEREST F, GIVEN,
30    REM              P = THE PRINCIPAL ($)
40    REM              R = ANNUAL INTEREST RATE (%)
50    REM              Y = NUMBER OF YEARS
60    REM              T = NUMBER OF TIME/YEAR INTEREST IS
                              COMPOUNDED
70    REM              Input P = – 999 to stop.
75    PRINT      "THIS PROGRAM COMPUTES PRINCIPAL PLUS
                    COMPOUND INTEREST"
77    PRINT
78    PRINT
80    INPUT      "PRINCIPAL" P
85    IF P = –999  THEN  GOTO  210
90    INPUT  "ANNUAL  INTEREST" R
100   INPUT "NUMBER OF YEARS" Y
110   INPUT  "NUMBER OF TIMES INTEREST IS COMPOUNDED IN
                A YEAR" T
115   REM    Lines 120 to 150 detect errors in the input data
120   IF  P  <  0  THEN  GOTO  190
130   IF  R < 0  THEN  GOTO  190
140   IF  Y < 0  THEN  GOTO  190
150   IF  T < 1  THEN  GOTO  190
160   F = P* (1 + R/(100 * T)) ↑ (Y * T)
165   PRINT
170   PRINT      "PRINCIPAL PLUS INTEREST = $"; F
180   GOTO       77
190   PRINT
195   PRINT      "INCORRECT DATA TRY AGAIN"
200   GOTO       77
210   PRINT
215   PRINT      "END OF SESSION" : END
```

```
RUN

THIS PROGRAM COMPUTES PRINCIPAL PLUS COMPOUND INTEREST

Principal  ?  1000
Annual Interest?  15
Number of years?  5
Number of times compounded in a year?  365

Principal plus interest = $2115.99

Principal?  15000
Annual Interest?  10
Number of years?  3
Number of times compounded?  –12

Incorrect data try again

Principal?  15000
Annual Interest?  10
Number of years?  3
Number of times compounded?  12

Principal plus interet = $20222.6

Principal?  –999

END OF SESSION
```

Comment on the solution

Your program will not be the same as the solution given. Where they are significantly different, pause to think which is better. Don't be surprised if yours is!

Carbon dating

When an ancient object e.g. a fossil or a mummy is found, it is often of interest to determine the age or date of the object. One method is by carbon dating.

ClibPDF - www.fastio.com

COMX 35 footnote: More on carbon dating

Scientist determines the percentage P of carbon 14 (a radioactive isotope) in the object relative to that in the atmosphere, and knowing the rate of decay R of radioactivity, the age of the object T in years can be determined as follows:

$$\frac{P}{100} = e^{-RT} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(1)}$$

R itself is determined from H, the half-life, which is the time in years required for the carbon isotope to lose half of its radioactivity. Thus,

$$R = \log e^{2}/H \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(2)}$$

Combining (1) and (2),

$$T = -H \log\left(\frac{P}{100}\right) / \log e^{2}$$

$$= H\,\mathrm{Abs}\left(\log \frac{P}{100}\right) / \log e^{2}$$

H equals 5730 years. Therefore given P, one can compute T.

Statement of the problem

Given P, the % of "carbon 14" in an object, scientist can compute the age T (in years) of the object by the following formula,

$$T = 5730\ \mathrm{Abs}\left(\log \frac{P}{100}\right) / \log e^{2}$$

You are asked to write a well-documented program accepting the input P from the user, and outputing the age of the object T.

Solution

```
10        REM        THIS PROGRAM DETERMINES THE AGE OF
20        REM              AN OBJECT BY CARBON DATING.
```

```
30        REM                        P =  THE PERCENTAGE OF
40        REM                             CARBON 14 IN THE OBJECT
50        REM                             RELATIVE TO THAT FOUND IN
60        REM                             THE ATMOSPHERE
70        REM                        T =  AGE OF OBJECT IN YEARS
80        REM      TYPE P = –999 to stop.
90        PRINT    "CARBON DATING"
100       PRINT
105       INPUT    "PERCENTAGE LEVEL OF RADIOACTIVITY" P
110       IF  P = –999  THEN  GOTO  180
120       IF P < 0      THEN  GOTO  160
125       IF  P > 100   THEN  GOTO  160
130       T=5730 * ABS (LOG(P/100))/LOG(2)
140       PRINT "AGE OF OBJECT IN YEARS ="; T
150       GOTO   100
160       PRINT  "INCORRECT DATA: TRY AGAIN"
170       GOTO   100
180       PRINT  "END OF SESSION" : END
```

```
RUN

CARBON DATING

PERCENTAGE LEVEL OF RADIOACTIVITY?  52
AGE OF OBJECT IN YEARS = 5405.78
PERCENTAGE LEVEL OF RADIOACTIVITY?  120
INCORRECT DATA: TRY AGAIN
PERCENTAGE LEVEL OF RADIOACTIVITY?  80
AGE OF OBJECT IN YEARS = 1844.65
PERCENTAGE LEVEL OF RADIOACTIVITY?  –999
END OF SESSION
```

3.9 Going around in loops (use of the FOR/NEXT loop)

You may recall in section 3.4 how a three-line program can compute the areas of all the triangles in the world! The trick was to use the GOTO statement to branch back to the beginning of the program. There are other ways to tell the computer to obey a block of instructions repeatedly. Such a block is called a loop. We talk about going round the loop (i.e. a block of instructions) n times or simply to loop for n times.

ClibPDF - www.fastio.com

Try the following:

```
10      FOR I = 1 TO 100 STEP 1
20      PRINT  "COMX  35";
30      PRINT  "    IS CLEVER"
40      NEXT  I
RUN
```

What happens? You are filling your screen with rows and rows of the message "COMX 35   IS CLEVER", one hundred times to be exact.

The FOR statement in line 10 and the NEXT statement in line 40 work together as a "FOR/NEXT" pair. The pair tells the COMX 35 to repeat obeying the instructions in-between (line 20 and line 30) 100 times i.e. for I = 1 to I = 100 using a step-size of 1.

Try again,

```
10        FOR I = 0 TO 2000 STEP 2
20        PRINT I;
30        NEXT
RUN
```

The screen is filled with "024 ... 2000". In this example, the step size is 2 and therefore 1001 numbers are printed. A line by line analysis is as follows:

Table 3.9 A line by line description of the action of a FOR/NEXT loop

line number	action
10	Set I = 0
20	Print 0
30	Increase I by 2, (the step size), and go back to 10
10	I = 2
20	Print 2
30	Increase I by 2, go back to 10
10	I = 4
20	Print 4
30	Increase I by 2, go back to 10
10	I = 6
	•
	•
	•

This will continue until and including i = 2000, and the looping will stop. Note that in this example, the PRINT instruction to be obeyed repeatedly, namely line 20, changes with I.

The step size can be negative. Try

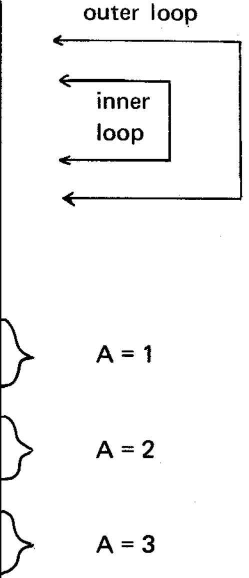
```
10          For A = 11 to 8  Step –1
20          PRINT A
30          NEXT A
40          PRINT "FINISH"
RUN
11
10
 9
 8
FINISH
```

If the step size is not specified, a step size of 1 is assumed. Also, it is possible to have a FOR/NEXT loop within another FOR/NEXT loop, a situation described as "nested" loop.

ClibPDF - www.fastio.com

Try

```
10        FOR  A = 1  to 3
20        FOR  B = 1  to  6        step 2
30        PRINT  A, B

40        NEXT  B
50        NEXT  A
60        PRINT  "FINISH"
RUN

1         1
1         3
1         5
2         1
2         3
2         5
3         1
3         3
3         5
FINISH
```



The inner loop is repeated three times for each time the outer loop is gone through, i.e. for each value of A. Follow the above program, noting how the values for A and B change, in particular, when A changes from 1 to 2, B begins at B = 1.

ClibPDF - www.fastio.com

As a final example, try

```
10          FOR  A = 1 to 10
20          IF A = 4  THEN A = 9
30          PRINT  A
40          NEXT  A
50          PRINT  "FINISH"

RUN
 1
 2
 3
 9
10
FINISH
```

Note how the IF-statement in line 20 reduces the looping from 10 to 5 times.

3.10 Breaking a complicated program into blocks (using subroutines) — GOSUB and RETURN

Imagine you are to write a program which will enable you to play a game similar to the "Space Invader". Being able to write a game program of such sophistication has made many very rich and famous. With the clever COMX 35 BASIC, COLOR graphics and sound effect, you are in fact well equipped to do this. It takes learning, a mixture of fun, hardwork and imagination. How should you begin?

One way is to identify the various jobs which has to be done repeatedly e.g. job 1 to draw the space-ship at any position on the screen, job 2 to draw the missile-launcher, job 3 to create the color graphics and sound effect when a space-ship is hit, job 4 to create the color graphics and sound effect when the missile-launcher is hit, job 5 to determine whether a space-ship has been hit, etc.

To simulate a moving space-ship, draw a space-ship (by calling upon job 1) at its original position, but using the color of the background. This will erase the original space-ship. Now, draw a space ship slightly displaced from its original position and along the direction of flight. If this is repeated many times, and rapidly, you have a moving space-ship, but job 1 will have to be activated many times indeed.

Now, job 1 is done by a block of program. If we insert this block of program in places wherever a space-ship is to be drawn, the program will be very long indeed. Another way is to store a copy of the job 1 program (called the subroutine) somewhere, and insert a "goto subroutine" or "GOSUB" statement in the main program wherever a space-ship is to be drawn. This is illustrated in fig. 3.10, where the numbered arrows correspond to the steps below.

Step 1 when the computer encounters "GOSUB 5000" in line 110, it goes or transfers control to the beginning of the subroutine at line 5000.

Step 2 Instructions in the subroutine is obeyed until the last instruction RETURN is reached.

Step 3 Control is returned to line 120, immediately below the "GOSUB" statement in line 110.

Step 4 Execution of the main program is continued.

Step 5 The computer encounters another "GOSUB 5000" in line 210 and as in step 1, control is transferred to the beginning of the subroutine.

Step 6 Same as step 2.

Step 7 Control is returned to line 210.

Step 8 Execution of the main program is continued.

ClibPDF - www.fastio.com

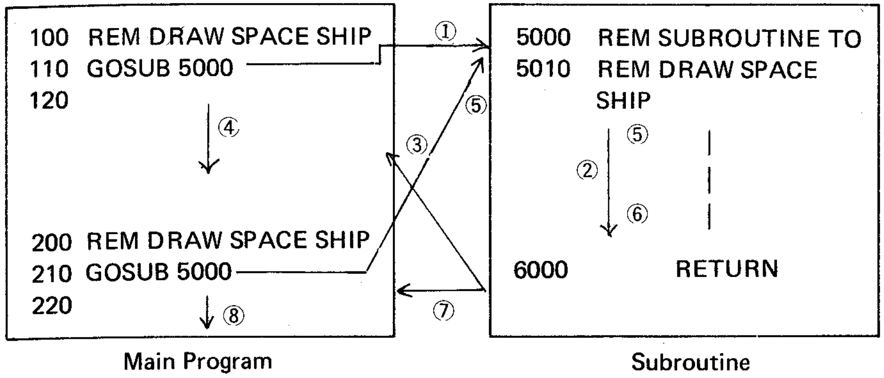


Fig. 3.10 A step-by-step explanation of the action of a main program calling a subroutine by GOSUB

Note that in "GOSUB N", N is the line-number of the first instruction in the subroutine. Note also that a subroutine always ends with a RETURN statement.

To illustrate the use of subroutines, we shall write a program to find the greatest common divisor & GCD (also known as the highest common factor, HCF) of three integers, A, B and C. We shall need a subroutine to find the GCD of two numbers. Let the GCD for A and B be Y. Then, we call the subroutine again to find the GCD of Y and C. The result is then the GCD of A, B and C.

GREATEST COMMON DIVISOR

```
10      REM      THIS MAIN PROGRAM
20      REM      THE GCD OF INTEGERS
30      REM      A, B, C
40      REM
42      PRINT    "THIS PROGRAM FINDS THE GCD"
43      PRINT
50      INPUT    "A", A
55      IF  A  <  0  THEN  GOTO  150
60      INPUT "B",B
70      INPUT "C",C
80      X = A
90      Y = B
100     GOSUB    1000
110     X = C
120     GOSUB    1000
130     PRINT    "THE GREATEST COMMON DIVISOR ="; Y
140     GOTO     50
150     PRINT    "END OF SESSION"; END
1000    REM      THIS SUBROUTINE FINDS THE GCD OF
1010    REM      TWO INTEGERS X AND Y
1020    REM      AND THE GCD = F
1030    REM
1040             Q = INT (X/Y)
1045             R = X – Q * Y
1050             IF R = 0 THEN GOTO 1100
1060             X = Y
1070             Y = R
1080             GOTO 1040
1100             RETURN
```

3.11 Further editing tips

As we attempt to write longer programs, mistakes are likely to be made more often, and the editing facilities described in sections 2.1 and 3.3 may not be adequate. In addition to the use of the "DEL" key, "CNTL", and "line numbers" for editing (see section 2.1 and section 3.3), COMX 35 in fact provides more sophisticated facilities for correcting errors in a program.

This is done through the use of the command EDIT. This has the form

```
EDIT  X
```

where X is an integer denoting the line number of the statement requiring correction.

In response to Edit X, line X is displayed. The computer is now said to be in the "edit mode" (i.e. under the control of the EDITOR instead of in the "COMX 35 BASIC mode" which puts COMX 35 under the control of the BASIC interpreter (see section 4.3)). The user can now move the cursor (by pressing the space key) to a position under the line displayed that is one character before (i.e. to the left of) the character to be modified. At this point, three EDIT options are available. The user may type I (for Insert), D (for Delete) or C (for Change). After the option is selected, the option character (I, D or C) is displayed and the cursor is now directly under the character to be modified.

The Insert option allows characters to be entered or inserted immediately after the position specified by I. The Delete option removes character from the line sequentially for each pressing of the space key i.e. pressing space key n times will delete n characters. The Change option replaces sequentially each character with the new one that is typed after C.

After the user has performed the desired modification, typing CONTROL S (holding CNTL key down then press S) causes line X to be retyped with the modification incorporated. Note that in one pass, only one option (I, D or C) is allowed. If necessary, the user, however, can exercise one of the 3 options as many times as is needed for all corrections to be made. When this is done, press CNTL S again to "exit" from the EDIT mode and to indicate that the editing of line X is completed.

The above may be illustrated as follows where "u" denotes the space key. Type,

```
10        PRINT        "HELLO"
```

ClibPDF - www.fastio.com

To edit line 10, type EDIT 10 to get,

```
: EDIT  10
10          PRINT              "HELLO"
◆- - - - - - - - - - - -→◆
```

where the line to be edited is 10. Now, position the cursor beneath the " getting ready for the next step.

To insert SAY before HELLO in line 10, select the I option, and type SAY followed by SPACE to get,

```
:   EDIT     10
10          PRINT              "HELLO"
                               ISAYu
```

Now type CNTL S to get

```
10          PRINT              "SAY HELLO"
```

Type CNTL S again to exit from EDIT mode.

To delete SAY, and change HELLO to JERRY, select D option, followed by 4 spaces to get

```
:   EDIT     10
10          PRINT              "SAY HELLO"
                              D u u u u
```

Note that for each space after D, one character is deleted. Press CNTL S to display the corrected line. Since the editing is not completed yet, we do not exit edit mode at this point. We proceed and position the cursor under " in the partially corrected line, and choose Change option,

```
10          PRINT              "HELLO"
                               CJERRY
```

Pressing CNTL S will display,

```
10          PRINT              "JERRY"
```

ClibPDF - www.fastio.com

Note that the 5 characters HELLO have been changed or replaced by another 5 characters JERRY. Now that the editing job is completed, press CNTL S again to exit the edit mode. The computer will respond by

```
READY
:  ◆
```

showing that control is now returned to the COMX 35 BASIC interpreter.

For an error to occur in a short simple line, retyping the entire line using the same line number is the simplest way to do editing. For a long line, there is the possibility of introducing new errors whilst attempting to correct the old ones. In this case, the use of EDIT is preferred.

Pressing CNTL C while in the EDIT mode will abort the edit mode and return to BASIC mode leaving the line being edited unchanged.

3.12 Creating Special Graphical Characters

3.12.1 Built-in "Graphical Characters" – using the function CHR$

To have a preview of the built-in graphical characters, run the following program.

Program 3. 12. 1

```
10        FOR  I = 0  to  127
20        PR  I,  CHR$ (I); "          ";
25        IF   I = 127      THEN     PR
30        NEXT

RUN
```

The application of the function CHR$(X) is detailed in section 5.5. Briefly, CHR$(X) is a function which returns a character for each integer X given. Corresponding to each key, there is an integer or code (the ASCII code to be exact where ASCII stands for "American Standard Code for Information Interchange") and a character e.g.

the key A has a code number of 65 (65 in decimal, or 41 in hexadecimal), and pressing key A will normally output the character A. Hence "PR CHR$ (65)" is equivalent to PR "A". The above program will thus display all the 128 characters corresponding to the code numbers 0 to 127. Now, when the COMX 35 is first turned-on, characters having decimal codes from32 to 95 and 160 to 223 give the standard characters (i.e. SP !, etc. through 1,2, ... 9, A, B, C ... Z, etc.) as shown in Appendix A.2. The keys having decimal codes 0 to 31, 96 to 127, 144 to 159 and 224 to 255, give built-in graphical characters, e.g. PR CHR$(21) will print the hexagonal shape in cyan color.

TRY,

```
PR   CHR$   (111, 112, 113)
```

and press CR. The three built-in characters printed side-by-side make up the picture of an aeroplane.

Draw a longer plane by,

```
PR CHR$ (111, 112, 112, 112, 113)
```

The same picture can be obtained by holding down the shift key and press keys O (once), P (3 times) and Q (once).

3.12.2 User-defined characters — using the command SHAPE

It is possible to redefine the character printed by the pressing of a key e.g. it is possible to redefine the character for decimal code 65 such that pressing key A will give, for example, a chinese character instead of the letter A.

This is done by using the command SHAPE which has the form (for TV using the PAL system),

```
SHAPE (X, "18 hex numbers")
```

where X is an integer corresponding to the decimal code of the key. Within the pair of double quotes are 18 hexadecimal numbers which together specify the color and shape of the character. Try,

```
10 SHAPE  (65, "48487E6A7E48484800")

RUN
```

(Note: SHAPE can also be used in the direct-execution mode without the line-number.)

Pressing key A will cause the Chinese character 中 to be printed in magenta. In fact, notice that all letter A's on the screen have been replace by 中 after the execution of the above SHAPE command!

Try the following examples, you will appreciate the magic done by these commands.

Example 1

```
SHAPE (153, "8F8D8C5C7E7E7E5C00")
```

An apple will be printed when you "PRINTED CHR$ (153)."

Example 2

```
SHAPE (119, "84848CBF9E9C94A400")
```

Whenever you press the letter "W" with the SHIFT key being held down, the computer will respond by giving you a green star.

ClibPDF - www.fastio.com

Example 3

```
SHAPE (20, "00000000DFFFFFDF00")
SHAPE (21, "00FCFCFCFFFFFFFF00")
SHAPE (22, "00000000FEFFFFFE00")
```

After pressing CNTL-T, CNTL-U, and CNTL-V, you can get a submarine on the screen.

For TVs using the NTSC system, instead of using 18 hex numbers, use 16 hex numbers to specify the shape.

COMX 35 Footnote on Hexadecimal Numbers

Hexidecimal requires 16 distinct characters, and therefore, 6 characters in addition to 0, 1, ... 9 are needed to represent "10", "11", ..., "15". These are usually taken to be A, B, C, D, E, F. Hexadecimal is related to binary using four-bit groups. Thus,

111 / 1010 / 0001 / 0010 / 0001

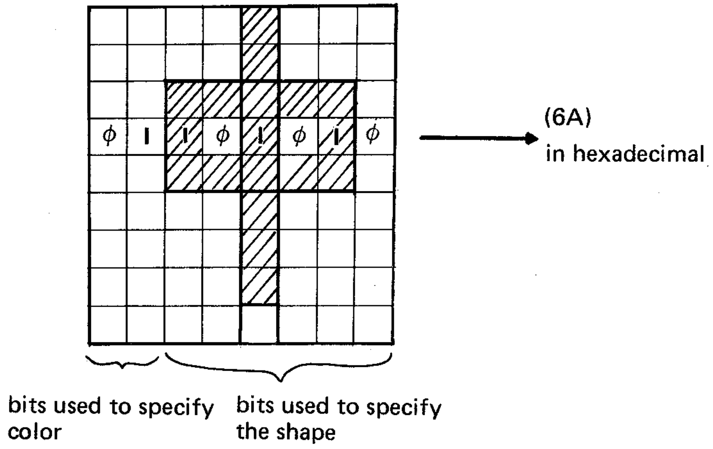
becomes

7 A 1 2 1

in hexadecimal.

To determine the 9 pairs of hex numbers, consider a rectangle of 8 units by 9 units as follows:

Fig. 3.12 Writing the SHAPE command for the character ф



Consider each row of 8 squares. The leftmost two squares, which are to be filled in by 1's and 0's, determine the color of the remaining 6 squares in accordance with Table 3.12.2. Each of the rightmost 6 squares of each row is represented by 1 if that square is "painted" or "filled", by 0 if it is not (i.e. if it has the same color as the background). Thus, the first two columns of squares (from the left) are used to define colors, whereas the rightmost 6 columns of squares (6 x9 squares) are used to define the shape. Each row is coded into two hex numbers. With there being 9 rows, 18 hex numbers are needed to define both the colors and the shape. For example, consider the 4th row from the top of Fig. 3.12, if the magenta color is chosen, the leftmost two bits should be "01" (see Table 3.12.2). The remaining six bits are "101010" as shown in Fig. 3.12, coding a "filled square" into "1", and an "empty square" into "0". The eight bit pattern "01101010" may be represented by the two hex numbers 6A. Hence, the 4th pair (corresponding the 4th row) of hex numbers is 6A (see Fig. 3.12).

Note that for PAL system, the resolution per character is 6 x 9 dots. For NTSC, only 8 pairs of hex digit is required giving6 x 8 dots if the hex string is longer than required, the rest is ignored. In

ClibPDF - www.fastio.com

both cases, each character can be multi-colored. Note also that string variables can be used instead of literals when defining a shape, also the user-redefined characters or symbols can be printed like any other normal characters.

The default character set will be restored during power-up or system reset (i.e. pressing of RT and space bar simultaneously)

Table 3.12.2 Colors of one row of squares as a function of the first two bits

Left most two bits	Color (if output by computer)	Color (if input from keyboard)
0 0	BLACK	RED
0 1	BLUE	MAGENTA
1 0	GREEN	YELLOW
1 1	CYAN	WHITE

3.13 Time-controlled subroutines — use of the commands TIMOUT and TIME

The commands TIMOUT and TIME are used jointly to enable the programmer to call a subroutine after a specified lapse of time. The "alarm clock" is set by the command,

```
TIME (X)
```

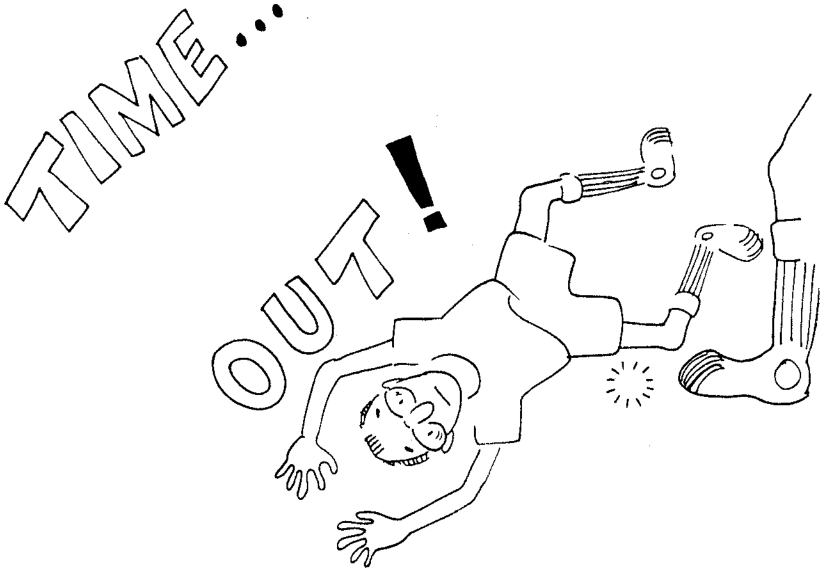
where X is an integer. When the above instruction is executed, the "clock" starts ticking. When X units of time has elapsed, the command,

```
TIMOUT Y
```

is activated and the computer is directed to jump to a subroutine beginning with the line number Y.

In the PAL version of the COMX 35,, 50 units is equivalent to 1 second.

In the NTSC version of the COMX 35,, 60 units is equivalent to 1 second.



ClibPDF - www.fastio.com

The typical structure of a program involving a time-controlled subroutine, and the sequence of actions are illustrated in Fig. 3.13.

Step 1:

The instruction TIMOUT (1000) is encountered. The computer takes note of the beginning address of the subroutine, but there is no other action as yet.

The instruction TIME (150) is executed, and the "clock" is set to tick.

Step 2:

The computer proceeds as usual.

Step 3:

When 150 units of time (i.e. 3 seconds for PAL machines or 2.5 seconds for NTSC machines) have elapsed after the execution of statement 100, the "clock alarm" will cause the computer to jump to the beginning of the subroutine.

Step 4:

The subroutine is being executed. It is usual to find the "clock" being set inside the subroutine but this is not mandatory. Statement 1990 shows that the "clock" is set to give an "alarm" after 100 units of time.

Step 5:

Upon hitting RETURN, control is returned to statement 20 immediately after TIMOUT.

Step 6:

The computer will proceed as usual until 100 units of time have elapsed after statement 1990, when again control is passed to the subroutine.

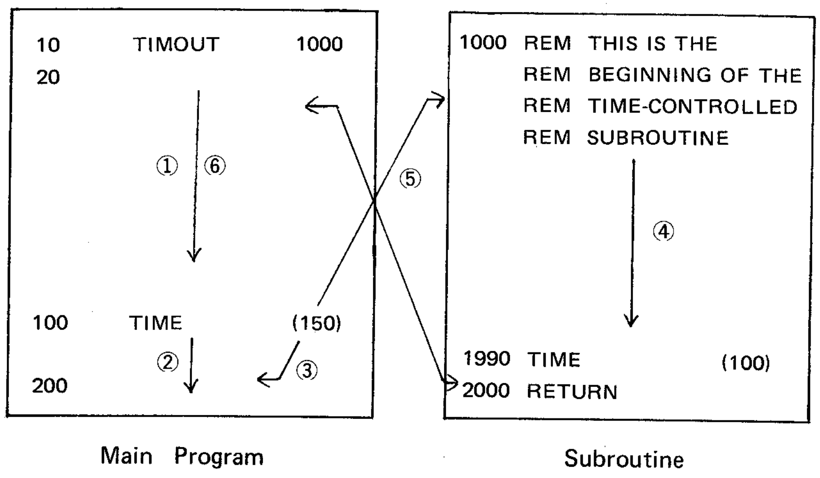


Fig. 3.13 The typical structure and sequencing of a program using a time-controlled subroutines.

3.14 Functions of the Control (CNTL) key

If certain keys are pressed when the control (CNTL) key is held down, the computer will take certain action, but no character will be displayed on the screen.

CNTL C

As discussed on section 2.1, if key C is pressed while CNTL key is held down (an operation abbreviated as "CNTL C") prior to pressing CR, the line being typed will be ignored, and the cursor will jump to the leftmost position on the next line. CNTL C enables the user to instruct the computer to ignore a partially typed statement.

CNTL R

CNTL R will cause the computer to repeat the last statement input on the screen. This facility is specially useful in the situation when the new statement to be typed in differs slightly from the last statement. Modifying the second copy of the last statement by the DEL key and by retyping, and then press CR will send the new statement to the computer.

ClibPDF - www.fastio.com

Cursor movement (CNTL I, K, M, L)

Pressing keys I, K, M and L while holding the CNTL key down will move the cursor position as follows:

I : Cursor moved up by one row (↑)
M : Cursor moved down by one row (↓)
K : Cursor moved to the right by one column (→)
J : Cursor moved to the left by one column (←)

Note: total number of moves must not exceed 95 when executed directly from keyboard.
Use PRINT CHR$(128) to move cursor up in program.
Use PRINT CHR$(130) to move cursor down in program.
Use PRINT CHR$(129) to move cursor right in program.
Use PRINT CHR$(131) to move cursor left in program.

3.15 Real-time control of a program — using KEY and the JOYSTICK

KEY

This function returns the ASCII code of a key pressed. Its use may be illustrated by the following example,

```
10     IF  KEY = 65  THEN  GOTO  30
20     GOTO  10
30     PRINT  "KEY A PRESSED"
40     END
```

When the function KEY is encountered, the keyboard is scanned, and the ASCII code of the key pressed is returned. If this is equal to 65 (which is the ASCII decimal code for key A), the message "KEY A PRESSED" is printed, and the program is terminated. If any key other than key A is pressed, the program goes round a loop, scanning the keyboard repeatedly.

The above program illustrates the tyical situation the function KEY is used. It is most often used in an "IF-statement" located within a loop, such that the keyboard is scanned repeatedly, and certain action will be taken if a specified key is pressed. This enables the pressing of the specified key to control the flow of a program.

(Note: the following information may become useful for a more detailed understanding of the KEY function. When a key is pressed, its ASCII code is sent to a register or latch. Functions like KEY (or INPUT) will read the content of this latch, and then clear the latch. If no key is pressed, KEY will return the value 0.)

JOYSTICK

The built-in joystick is a device which controls 4 keys. It will input into the computer one of 4 codes according to which direction the joystick is pressed. The 4 codes (which will not be displayed) have ASCII decimal codes of 136, 137, 138, and 139, corresponding respectively to the up, right, down and left positions of the joystick. If the statement, "IF KEY = 136 THEN ..." appears somewhere within a loop, whenever the joystick is flicked to its "up" position, (which has an ASCII decimal code of 136), the condition within the IF statement is satisfied. This enables the joystick to control the flow of the program.

ClibPDF - www.fastio.com

Try,

```
10        IF  KEY = 136   THEN  GOTO  30
20        GOTO  10
30        PRINT   "JOYSTICK IS ACTIVATED"
40        END
```

The program will print a message when the joystick is flicked to the "up" position.

ClibPDF - www.fastio.com

CHAPTER FOUR

FUNDAMENTAL COMPUTER CONCEPTS

CHAPTER 4 FUNDAMENTAL COMPUTER CONCEPTS

4.1 What is a computer?

A computer is a device capable of accepting, processing and outputing information. The information is in the form of electrical signals and represents alpha-numeric data (i.e. text, words and numbers). By processing, we include mathematical and logical operations, sorting, retrieval and rearrangement. The output "processed" information has three main categories of application, (a) data processing (including business and accountancy application, record keeping, information retrieval, management information system to facilitate decision making), (b) scientific computation and engineering design, and (c) industrial control in which the input information is picked up and converted by sensors or transducers into electrical signals and processed in a prescribed manner. The output signals are then used to activate various electro-mechanical devices.

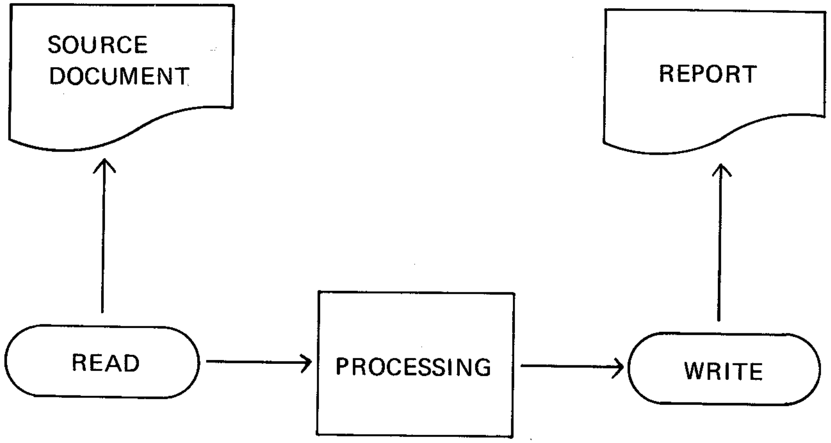


Fig. 4.1 The Computer as an information processor.

ClibPDF - www.fastio.com

4.2 The main parts of a computer

Problems may be solved by the computer or by a human operator equipped with a calculator, pencil and paper. It is instructive to compare these two situations, and in so doing, identify the subsystems of a computer. The comparison is shown in Table 4.2.

Table 4.2 Problem solving by a human operator and by the computer — a comparsion of their requirements

	Human operator	Computer
1.	Problem sheet — for recording the specification of the problem.	Input device
2.	An algorithm — an ordered sequence of steps specifying the procedure for solving the problem.	Program storage
3.	Data which when processed according to the prescribed algorithm, yields the required answers.	Data storage.
4.	The "calculator" — which performs the basic operations.	The arithmetic/ logic unit (ALU)
5.	The human operator — which controls the execution of the algorithm	The control unit.
6.	Pencil and paper — for recording the intermediate results.	Temporary (scratch—pad) storage.
7.	Result sheet for recording the final results.	The output device.
8.	A library of reference books.	Mass storage.

Table 4.2 shows that the computer consists of

(a).. A subsystem for doing arithmetic/logic operations and another subsystem for controlling the sequencing of each step. The former is called the arithmetic-logic unit or the ALU and the latter, the control unit. In the computer, these two subsystems are often grouped together in one central processing unit or the CPU. The CPU is likened to the human operator equipped with a calculator.

(b).. A device for storing the program, the input data and the intermediate results. In the computer, these are stored in the main memory. The main memory is likened to the pencil and paper used by the human operator.

(c).. Input and output devices for the computer to communicate with the the outside world, enabling the computer to accept a statement of the problem and the associated data (e.g. from the keyboard), and to output the results (e.g onto the TV screen).

(d).. Another device for storing a vast amount of information which is less frequently used. Such mass-storage devices (e.g. the floppy disk or the cassette tape) are slower than the main memory, but can store a large volume of information. It is likened to the library of reference books which the human operator may consult occasionally.

Therefore the main subsystems of a computer are: the CPU, the main memory, the input/output devices and a mass-storage device.

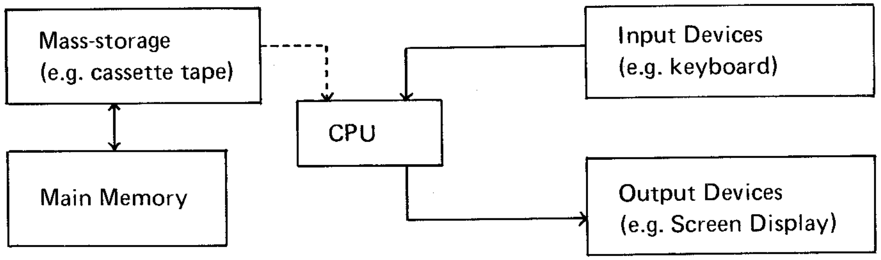


Fig. 4.2. The main subsystems of a computer

4.3 Computer software

By "software", we refer to the programs or the step-by-step procedures given to the computer telling it how to perform different tasks. There are three main types of software.

ClibPDF - www.fastio.com

(a).. Application software, for solving specific scientific or data-processing problems e.g. programs for doing general ledger, inventory control, and for designing the structure of a building. These are acquired (or written) by the user.

(b).. System software, which makes it more convenient for the user to operate the computer, e.g. programs which can perform a variety of tasks such as enabling the user to save a program on the cassette tape, and subsequently to load it back to memory.

(c).. Language compilers and/or interpreters, which enables the user to write his application programs in a language like his natural language. Inside the computer, programs and data are represented by a large number of devices which can exist in two and only two "states" designated by 1 and 0, (like a switch which is either opened or closed). The computer can directly interprets these "one-zero" patterns. For the user, however, writing his programs using only 1's and 0's is an extremely difficult, if not an impossible task and the resulting programs would not be easily "readable". He would prefer to express his problem-solving procedure in a language similar to his natural language, such as BASIC, which would have to be converted into "one-zero" (or binary) pattern before it can be executed by the computer. Such conversion is effected by software referred to as "compilers" or "interpreters". A compiler translates the entire program written in higher language into a machine-language version which is then executed. An interpreter translates and executes each source program statement before proceeding to the next statement.

ClibPDF - www.fastio.com

CHAPTER FIVE

COMX 35 BASIC REFERENCE GUIDE

CHAPTER 5 COMX 35 BASIC REFERENCE GUIDE

This chapter is intended to be used as a reference manual. It provides comprehensive information on the COMX 35 BASIC Language. Users not familiar with BASIC and reading this chapter for the first time may find it dry and difficult to follow. As you become more familiar with BASIC, you will find this chapter an invaluable source of information.

5.1 Programs, statements, expressions and functions

A PROGRAM is a collection of computer-executable statements which tell a computer how to solve a problem. It embodies a methodology in the form of a step-by-step procedure called the ALGORITHM and DATA upon which the algorithm operates. Thus "program = data + algorithm".

A STATEMENT (which is preceded by a line number) consists of BASIC keywords (e.g. END, NEXT), expressions and functions.

FUNCTIONS are BASIC keywords which return values e.g. LOG(X) SIN(X) etc. A list of functions are given in section 5.4. The bracketed quantity is called the ARGUMENT of the function. The keyword defines a rule by which the argument may be transformed into a value called the value of the function.

AN EXPRESSION, which is short for arithmetic expression, may consist of anything from a single number, a single variable, a function or a group of operands (numbers, variables and functions etc) tied together with arithmetic operators. Some examples are:

```
8
5
A
3* sin (45) 0.5
3* (2 + A) / (B + C)
```

5.2 Numbers and variables

COMX 35 handles both integers and floating-point numbers. Both types are stored as 32-bit signed numbers.

ClibPDF - www.fastio.com

FLOATING POINT NUMBERS can have values ranging from −0.170141 x 10^{39} to +0.170141 x 10^{39}. The fractional part of the mantissa is accurate to six digits.

INTEGERS range from −2147483647 to +2147483647. They are accurate over the entire range.

All numbers entered from the keyboard are stored in the form they are entered. If entered without a decimal point, they are stored as integers; if entered with a decimal point or with the "E" notation, they are stored as floating point.

5.3 Operators

COMX 35 operators can be divided into five categories: arithmetic, assignment, relational, logical and miscellaneous.

Arithmetic operators

The following list gives the arithmetic operators with their symbols and the priority or order each will be acted upon in a mathematical expression.

Unary minus	:	–	: first priority
Exponentiate	:	↑	: first priority
Multiply	:	*	: second priority
Divide	:	/	: second priority
Add	:	+	: third priority
Subtract	:	–	: third priority

Operators on the same level of priority are acted upon from left to right. The symbol "–" is also used to denote negative numbers. Some examples illustrating the exercise of priorities follow.

```
4*3↑2=36   (3↑2 will be acted upon first before 4*(3↑2))
4/2*3= 6
4+2*3-5= 5
```

Assignment operator

The assignment symbol is the equal sign "=". When this symbol is used for the purpose of assignment, it takes on the meaning "takes the value of" rather than "equals". Some examples are:

```
LET A=5
LET B=B+A*2
```

The keyword LET is optional and may be omitted. The second example may be interpreted as: evaluate B+A*2 and use the result to replace B.

POKE (expr1, expr2)

This statement, which should be used with extreme caution, is used to write any location in memory with any data. The location in memory is defined by expr1 and the data is defined by expr2. For example, POKE (5000,255) will place the hexadecimal equivalent of 255 (FF) in memory at the decimal address 5000. A more common and much safer approach is POKE (■ 1F23, #FF) which places the hexadecimal data FF at hexadecimal location 1F23.

Relational operators

The relational operators are listed below.

Equality: =
Inequality: < >
Greater Than: >
Less Than: <
Greater Than or Equal To: > =
Less Than or Equal To: < =

Expressions that contain relational operators are evaluated to true or false rather than to a numeric value. Some examples are:

```
IF A = 3+B>2 THEN
IF A< > 4 THEN
IF A*2<=B/4 THEN
```

ClibPDF - www.fastio.com

Logical operators

The logical operators are AND, OR, XOR NOT. It should be noted that the logical operators convert the expressions to integer before performing the indicated operation and that the expression following the NOT operator must be within parentheses. Further, the logical operators have the same level of priority as + and –.

Some examples are:

```
IF (A AND B) <  >C THEN
FOR I=(A AND B) TO (A OR B) STEP A/B
PR (A XOR B)
GOTO NOT (A)
```

In the first three examples, the parentheses are inserted for clarity. They may be omitted.

Miscellaneous operators

The miscellaneous operators are listed below.

:	separates statements on the same line
;	PRINT delimiter
,	PRINT delimiter
"	string delimiter
(	group terms
)	group terms
\. . . .\	8-bit binary values
#	8-bit hexadecimal values
■	16-bit hexadecimal values

Some examples are:

```
PRINT A,B: "YES":GOTO 10
LET A = (3 + 4)/2
LET B = # F8: C=■01F3
IF A <  > \10011010\ THEN GOTO 100
```

Note 1 Use ; in PRINT statement to print the next item directly after the last without any spaces in between.

Note 2 Use , in PRINT statement to print the next item at the beginning of the next printing-zone. One printing-zone is eight characters wide.

Note 3 Use " to indicate the limits of a string of characters and therefore cannot be used as a member of the string.

5.4 Mathematical Functions

The term function is defined in section 1. The argument of a mathematical function may be a number, a variable or an expression. Functions are classified into (i) built-in functions, and (ii) derived functions. Derived functions are those which may be obtained from an expression involving the built-in functions. For example, SIN(X) and COS(X) are built-in functions, whereas TAN(X) is a derived function, since TAN(X) can be computed as SIN(X)/COS(X).
The preset state of COMX 35 assumes all angles to be expressed in radians. The following examples assume this state. To change this to degrees, type DEG followed by CR. To revert back to radians, type RAD followed by CR.

5.4.1 Built-in Functions

These consist of ABS, ATN, COS, EXP, FNUM, INT, INUM, LOG, MOD, PI, RUN, SGN, SIN, and SQR.

ABS (expr)

This function returns the absolute value of the expression

For example

```
PR ABS (–10*2)
```

will print

```
20
```

ATN (expr)

This function which has the same meaning as ARCTAN (expression) returns the angle (normally expressed in radians) whose tangent is equal to the value of the expression. This function is a floating-point function.

ClibPDF - www.fastio.com

For example

```
PR ATN(1)
```

will print

```
0.7854
```

since TAN (0.7854) = TAN PI/4 = TAN 45° = 1

COS (expr)

This function returns the cosine of the angle determined by the value of the expression (normally expressed in radians). This function is a floating-point function. For example

```
PR COS (PI/3)
```

will print

```
0.5
```

since cos 60° = 0.5

EXP (expr)

This function returns a floating-point number of 2.71828 (e) raised to the power of the value of the expression. For example

```
PR EXP (6/3)
```

will print

```
7.3891
```

since e 6/3 = e^2 = 2.71828^2 = 7.3891

FNUM (expr)

This function rounds the expression to the nearest whole number and converts it to the floating-point mode. It is an exact counterpart to INUM. For example

```
PR FNUM (RND (6))
```

will print a floating point value.

INT (expr)

This function returns the integer part of the floating-point expression truncating out the fractional part. The result is still in the floating-point mode. This function is not to be confused with INUM (described later). For example

```
A=7.9: B=INT(A):PR A,B.
```

will print

```
7.9    7
```

INUM (expr)

This function converts the floating-point expression to the integer mode, rounding to the nearest whole number. It provides a means for forcing a particular function to the integer mode.
For example

```
PR SQR (62.41)
```

would result in

```
7.9
```

and

```
PR INUM (SQR (62.41))
```

would result in

```
8
```

LOG (expr)

This function returns a floating point number the value of which is the natural logarithm of the value of the expression.

For example

```
PR LOG (10)
```

will print

```
2.3026
```

and should be

```
PR LOG (EXP (2))
```

will print

```
2
```

ClibPDF - www.fastio.com

MOD (expr1, expr2)

This function (modulo) returns as its value the following equivalent

```
      expr1  –  (expr1/expr2)*expr2
```

where each expression is first converted to integer. The function defaults to an integer mode value. For example

```
      PR MOD (10,3)
```

would result in

```
      1
```

PI

This function returns as its value 3.14159.

RND

This function returns a random floating-point number greater than or equal to zero and less than or equal to one.

RND (expr)

This function returns a random integer number greater than or equal to zero and less than the value of the expression.

However, if the random integer number generated is assigned to a variable, the variable must be defined as an integer first, (e.g. DEFINT A).

SGN (expr)

The value of this function is either +1, 0, or – 1 depending upon the sign of expr. If expr is positive, the function returns a +1. If expr evaluates to zero, the function returns as its value a 0. If expr is negative, the function returns a –1.

SIN (expr)

This function returns the sine of the angle determined by the value of the expression (normally expressed in radians). This function is a floating-point function. For example

```
      PR SIN (PI/6)
```

will print

```
      0.5
```

SQR (expr)

This function returns the square root of the value of the expression. This function is also floating point. For example

```
      PR SQR (65)
```

will print

```
      8.0623
```

5.4.2 More examples on mathematical functions

The following examples serve to illustrate the power of the COMX 35 BASIC mathematical functions. Arguments of functions can be functions or expressions themselves, leading to an ability to express very complex mathematical equations.

```
      PR SIN (45)
      A = INT (10*SQR (LOG (A*B)))
      C = EXP (10 + A)
      PR ATN (A*SIN(A))
```

5.4.3 Derived functions

The following functions are not built-in COMX-35 BASIC library functions but may be computed from the available built-in functions listed in section 5.4.1 using the following formulas:

ClibPDF - www.fastio.com

```
TAN (X) = SIN(X)/COS(X)
ARC SIN(X)  = ATN (X/SQR(1 – X*X))
ARC COS(X) = –ATN (X/SQR(1 – X*X)) + 1.5708
ARC SEC(X) = ATN (SQR(X*X – 1) + (SGN (X) – 1)* 1.5708
ARC CSC(X)= ATN(1/SQR(X*X)–1) + (SGN(X)–1)*
                1.5708
ARC COT(X)  = –ATN(X) + 1.5708
ARC SINH(X) = LOG(X + SQR(X*X + 1))
ARC COSH(X)= LOG(X + SQR(X*X–1))
ARC TANH(X)= LOG((1 + X)/(1 – X))/2
ARC SECH(X)= LOG ((SQR (1 – X*X) + 1)/X)
ARC CSCH(X)= LOG((SGN(X)*SQR(X*X + 1) + 1)/X)
ARC COTH(X)= LOG ((X + 1)/ (X – 1))/2
COT(X) = 1/TAN(X)
CSC(X)  = 1/SIN(X)
SEC(X)  = 1/COS(X)
COSH(X) = (EXP(X) + EXP(–X))/2
COTH(X)= EXP(–X)/(EXP(X)–EXP(–X))*2 + 1
CSCH(X) = 2/(EXP(X) – EXP(–X))
SECH(X) = 2/(EXP(X) + EXP(–X))
SINH(X)  = (EXP(X)–EXP(–X))/2
TANH(X)= –EXP(–X)/(EXP(X) + EXP(–X))*2 + 1
```

5.5 Strings and string functions

A string is a sequence of characters (any printable characters or non-printable characters). A blank space may be included in a string but the quotation mark " cannot because quotation marks are string delimiters i.e. they are used to mark the beginning and end of a string. For COMX-35 BASIC, each string can contain up to 127 characters.

Strings are used to represent non-numeric data as labels, messages etc. Just as the ability to manipulate numbers enables the computer to do scientific calculations, the ability to manipulate strings enables the computer to do word-processing, record keeping and information retrieval.

The functions for manipulating strings are: ASC, CHR$, FVAL, LEN, and MID$.

ASC (string expr)

This function returns as its value the decimal equivalent of the ASCII value of the first character in the referenced string. The value returns in floating point as a default. If it has been preceded by any integer function it will be converted to integer.

For example

```
A$(5) = "ARITHMETIC"
B=ASC(A$(5))
```

The variable B would contain as its value, 65., the decimal ASCII value of the letter A.

As another example

```
5   A$="TEST"
10  FOR  A  =  1  TO  LEN(A$)
20  PR  ASC(MID$(A$,A))
30  NEXT  A
```

would result in

```
84
69
83
84
```

CHR$ (expr, expr, ...)

Corresponding to each bracketed expression, this function returns a character. Thus, CHR$ performs a function opposite to that of ASC. It evaluates each expression and outputs a character having an ASCII decimal code equal to the value of the expression. For example

```
PR CHR$ (65)
```

would result in

```
A
```

since the ASCII decimal code for A is 65, and

ClibPDF - www.fastio.com

```
PR CHR$ (#42)
```

would result in

```
B
```

since the ASCII code for B in hexadecimal is 42.

```
PR CHR$ (#41, #42,  #43)
```

would result in

```
ABC.
```

For more examples of the use of CHR$, see section 3.12. Appendix A.2 gives the ASCII codes for all built-in characters.

FVAL (string expr)

This function is special to COMX 35. It evaluates the string expression as an arithmetic expression and returns its value. In this manner the user can create mathematical functions. The followings contain three examples of FVAL usage.

```
10 A$="8+4"
20 PR FVAL (A$)
30 PR FVAL (A$+"/2")
40 PR FVAL ("SQR(3)")

:RUN (execute program)

12          (evaluate and print A$)
10          (evaluate and print FVAL ("8+4/2"))
1.73205     (evaluate and print SQR (3))

READY

: ◆
```

The string is limited to about 48 characters. Beyond that length, valuable data area may be clobbered.

LEN (string var)

This function returns as its value the number of characters in the specified string. The value returns in floating point unless it is preceded by an integer number or function, in which case it is automatically converted to integer. The following is an example.

```
A$="ARITHMETIC"
PR LEN(A$)
```

gives

```
10 (the length of A$)
```

and

```
PR 3*LEN (A$)
```

gives

```
30
```

Note that string must be previously defined.

MID$ (string var, expr1)

MID$ (string var, expr1, expr2)

This function is a string function which, when executed, extracts a portion of the specified string. The first term, expr1, defines which character from the left is to start the substring. Expr2 defines the number of characters to be used in the substring. If expr2 is not used, all of the remaining characters are used.

The following is an example.

```
A$="YES"
B$=MID$(A$,2,1)
```

B$ would then contain the letter "E".

```
A$(10)="EXPERIMENT"
PR MID$(A$(10),7,3)
```

ClibPDF - www.fastio.com

would result in

```
          "MEN"
```

being printed.

```
         10 INPUT A$ (1)
         20 IF MID$(A$(1),1,1)="Y" GOTO 100
         30 GOTO 10
        100 END
```

The last example would wait until some word beginning with the letter "Y" was input at which time the program would branch to line # 100.

Common functions like LEFT$ and RIGHT$ of other BASICs can be implemented with the MID$ function in the following way

```
          MID$ (A$,1,N)
```

is the same as

```
          LEFT$(A$,N)
```

and

```
          MID$(A$,(LEN(A$)–N+1),LEN(A$))
```

is the same as

```
          RIGHT$(A$,N)
```

Concatenation of strings

Concatenation is the process of adding two or more strings together such as adding "HELLO" and "JANE" to form another string "HELLOJANE". In concatenation, we use the + (plus) sign and form an assignment statement using strings and/or string variables.

The only limitation on string concatenation is that any string resulting from the addition of two or more strings may not exceed the 127–character limit.

For example

```
          10 A$="NEVER"
          20 B$="THE"
```

```
          30 LET A$(5)="LESS"
          40 LET B$(1)=A$+B$+A$(5)
          50 PRINT B$(1)
```

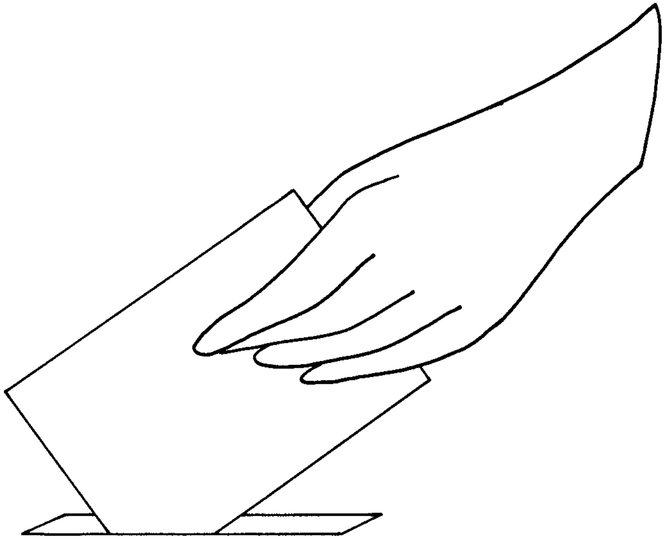
Line number 40 would result in the creation of a new string B$(1) that contains the first three strings placed back to back. "NEVERTHELESS" would be printed at line number 50.

Inputing a string, and using strings in IF statement

Considering the example below:

```
          10 INPUT B$
          20 A$(1)="CONTINUE"
          30 A$(2)="PHASE 1"
          40 A$(3)="PHASE 2"
          50 IF B$=A$(1)+A$(2) GOTO 100
          60 IF B$=A$(1)+A$(3) GOTO 200
          70 GOTO 10
```

The above program would wait at line number 10 for a string to be entered. That string would be stored in B$. It would then be compared to A$(1)+A$(3) and A$(1)+A$(3) with program execution transferred to the appropriate spot depending upon a match or mismatch.



ClibPDF - www.fastio.com

5.6 Arrays

A collection of items, all bearing the same group-name, but each designated by a number (or a set of numbers), is called an array. For example, a_1, a_2, ... a_n = A [1:n] is a one-dimensional array (a list or a vector) of dimension n. Each element in the list, ai is designated by a number i which marks the position of the element in the list relative to the first element in the list.

$$A\ [1{:}m,\ 1{:}n] = \left| \begin{array}{l} a_{11}, a_{12}, \ldots\ a_{1n} \\ a_{21}, a_{22}, \ldots\ a_{2n} \\ . \\ . \\ . \\ . \\ a_{m1}, a_{m2}, \ldots\ a_{mn} \end{array} \right|$$

is a two-dimensional array (or a table). Each element in the table has a row-value i and a column-value j which together specify the position of the element in the array.

COMX 35 is equipped with 26 arrays [A(expr) to Z(expr)] and 26 strings or string arrays A$ to Z$ and [A$(expr) to Z$(expr)]. The 26 numeric arrays may be one or two dimensional arrays. The maximum size of the array is 255 in any dimension. The minimum size of any dimension is 1. The largest array that can exist is G(255,255) and the smallest is G(1). The maximum number of array elements is limited by the amount of RAM that is available. Any array that is used must first be dimensioned by use of a DIM statement, which is explained in section 5.9.

The array space is cleared in several ways: RUN, NEW and CLD. This step is discussed later in the explanation of each of these statements. It is also valuable to know how much memory is being used by each array. Each element in the array is four bytes long. Furthermore, each array has a header of five bytes. Therefore, an array that is dimensioned DIM A (10, 10) contains 100 elements (10 X 10), which is 405 bytes for the array named "A". Similarly, array B, dimensioned as DIM B(20), would contain 20 elements, 80 bytes plus the header and bring the total to 85 bytes. The EOD statement, as explained later, tells how to find out how full the data space is getting.

String capability is also provided by COMX 35. It provides 26 string variables (A$ to Z$). Each string can contain up to 127 alphanumeric characters. Also included are string arrays that enable the programmer to index into a table of strings by means of some numerical argument. A string, therefore, may be referenced to by A$ (N) where N is any expression (or number) with a maximum value of 255. If a number greater than 255 is used, erroneous results could occur.

A string having no argument is assumed to have an argument of zero. For example

A$ is equivalent to A$ (0).

It is also important to note that no dimensioning of any kind for strings is necessary. Memory is automatically allocated each time a string is generated. Also note that the generation of a string C$ (10) does not imply that C$ (1) through C$ (9) exist. Only C$ (10) exists and an attempt to read C$ (1) results in an error message stating the C$ (1) has not yet been generated.

5.7 Commands to control the flow of a program

These are called control statements. They perform conditional and unconditional branching. The control statements include END, EXIT, FOR, GOSUB, GOTO, IF, NEXT, RETURN, and WAIT.

END

This statement (end of program) serves as a stop or end. It terminates program execution and returns COMX 35 BASIC to the direct execution mode. An END statement may be placed anywhere and any number of times in a COMX 35 BASIC program. It may also be deleted completely if desired, if it would have been the last statement in a program.

EXIT expr

This statement is an unconditional branch to a line number defined by expr. It is intended for a premature escape from a FOR/NEXT loop or a subroutine. One precaution should be noted. If subroutines or FOR/

ClibPDF - www.fastio.com

NEXT loops are nested, the EXIT statement is designed to transfer control to a line number within the next level down of nesting. For example, if FOR/NEXT loops are nested four deep and it is desired to prematurely branch outside of the fourth FOR/NEXT loop, COMX 35 BASIC expects to be somewhere in the third FOR/NEXT loop after the EXIT has been executed. An EXIT may then be taken to some line within the second FOR/NEXT loop, and so on. This statement is used instead of the standard GOTO and is designed to clean up all of the modified stack pointers that result from a FOR/NEXT or a subroutine call. An example follows.

```
10 FOR I=1 TO 10
20 FOR J=1 TO 5
30 A=B+C
40 IF A=15 THEN EXIT 60
50 NEXT J
60 PR A
70 NEXT I
```

FOR var = expr1 To expr2 STEP expr3

This statement assigns to a variable the value of expr1. The program continues executing until a NEXT statement is encountered. At that time the value of the variable is incremented by the amount defined by expr3 which may be either positive or negative. Expr2 – var is then evaluated and compared to the sign of the step. It is assumed that zero has a positive sign. If a match in sign does not occur, execution resumes at the first statement after the last FOR statement and continues through the loop. At any time during the loop, the user can modify the value of the variable name by any available means (e.g., a LET statement). It should also be noted that the mode of the variable name sets the mode in which expr1, expr2, and expr3 are evaluated. If the STEP expr 3 is deleted, a step size of one is assumed. (See also section 3.9)

GOSUB expr

This statement (subroutine call) is identical to the GOTO statement with the exception that the program remembers where the GOSUB occurred. When COMX 35 BASIC encounters a RETURN statement, it will return to the statement following the last GOSUB statement executed. In this way subroutines may be nested as deep as memory will allow (the stack grows as the number of nested GOSUB's grows). An example of the GOSUB statement is

```
5 A=2000
10 GOSUB 1000: GOSUB 1000
20 GOSUB A
30 END
1000 PR "DONE––"
1010 RETURN
2000 PR "FINISHED" GOSUB 3000:GOTO 1010
3000 PR "COMPLETE": RETURN
```

This example results in the following being printed.

```
DONE ––
DONE ––
FINISHED
COMPLETE
```

See also section 3.10.

GOTO expr

This statement is an unconditional branch. It transfers execution immediately to the start of the line number specified by the expr. If the line number does not exist, an error message is generated. Examples are as follows.

```
10 GOTO 50
10 GOTO A+B
10 GOTO 100* (A–B)
```

IF statement

This may take one of the two following forms:

IF string reference < > string expr THEN statement

ClibPDF - www.fastio.com

IF expr (relational operator) expr THEN statement

This statement is a conditional statement but in COMX 35 BASIC, it is not only a conditional branch as in some other BASIC's. It tests for a condition, and if the condition is met, a statement or group of statements (separated by colons) is executed. If the condition is not met, then execution continues at the next line number. When string relations are compared, an equal or not equal sign is the only acceptable relation. In the case of arithmetic expressions, the following are acceptable relational operators.

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

The mode of the first expression defines the mode of the second expression. An example of a conditional statement follows.

```
10 IF A=B THEN PR A:WAIT(200):CPOS(0,0):CLS:GOTO 200
20 PR B
```

In this case, if the value of A is equal to the value of B, then the value of A will be printed, a delay will occur, the screen will be cleared, and execution will continue at line #200. If A is not equal to B, then the value of B is printed and execution continues from line #20 onward.

One additional point to be made is that the keyword THEN is optional and need not be used. A common mistake to be avoided is the following.

```
IF A>N THEN 200
```

Anything to the right of THEN, if it appears, is to be an executable statement. The number 200 is not an executable statement. The correct version would be as follows.

```
IF A>B THEN GOTO 200 or
IF A>B GOTO 200
```

Some more examples of acceptable IF statements are given below. Note the multiple conditions in the last example. If any of the conditions are not met, execution continues at the next line number. If all of the conditions are met, execution will finally get transferred to line #500 by means of the GOTO statement. It follows that the GOTO statement could have been any executable statement.

```
10 IF A(2)*B(1)>=C*SIN(A) PR A:GOTO 50
10 IF A$(2);nn "LIST" LET B=3:GOTO 100
10 IF B$(A+B)=MID$(A$,2,4) PR "OK"
10 IF MID$(A$(5),2,4) = "EBCD" PR "MATCH":GOTO 500
10 IF A$=B$+C$ THEN GOSUB 200:CLS:PR "DONE"
10 IF A=INT(S/5) IF B>10 IF C<5 GOTO 500
```

NEXT

NEXT Simple var

This statement closes the FOR/NEXT loop as described in the FOR statement. If the variable name is omitted, the NEXT statement returns to the last FOR statement and continues. If the variable name is included, COMX 35 BASIC will check to see if it matches the variable name used in the last FOR statement. If it does not match, an error message is issued.

RETURN

This statement (return from a subroutine call) marks the end of a subroutine. Execution returns to the statement following the last GOSUB executed.

WAIT (expr)

This statement provides a way to insert a delay in the execution of a program. The length of the delay is directly proportional to the value of the expression. 1 equals 128000 CPU clocks. The delay for an expression of 1 is approximately 6.4 milliseconds (i.e. 128000 x 1/(2 x 10)). An example of one application of the WAIT statement is as follows:

ClibPDF - www.fastio.com

```
10 INPUT A$(1)
20 PR "MESSAGE IS"; A$(1)
30 WAIT (500):CLS:GOTO 10
```

This routine allows the message to be viewed for some period of time (500 x 6.4 ms) before the screen is cleared and execution continued.

5.8 Command Statements

Command statements are system directives. The command statements include CLD, CLS, CPOS, EDIT, EOD, EOP, FORMAT, LIST, NEW, RENUMBER, RUN, RUN+, and TRACE.

CLD

This statement (clear data) when executed erases all strings and arrays. It does so by reseting all string and array pointers to their initial states.

CLS

Clears the screen from the current cursor position. To clear the complete screen type CPOS (0,0):CLS.

CPOS (expr, expr)

This statement (Cursor Position) directs the cursor to a position in the screen specified by the two expressions in bracket. The first specifies the row position (0 to 23) and the record, the column position (0 to 39) of the cursor. CPOS may be used together with CLS to enable part of the screen to be cleared, with PRINT for printing messages and headings beginning from specified position on the screen, or with SHAPE to create color graphics. For example

```
10        CPOS (11, 0)
20        CLS
RUN
```

will clear the lower half of the screen.

```
10        CPOS (11, 12)
20        PRINT "THIS IS A TEST"
RUN
```

will print the message at the centre of the screen.

EDIT expr

This statement opens the line number called by expr and allows the user to modify the line on a character basis. Further details are given in section 3.11.

EOD

EOP

The EOD statement (end of data) prints the hexadecimal address of the end of data (arrays and strings) space. The EOP statement (end of Program) is similar to EOD in that, when executed (normally in the direct execution mode), it automatically prints the hexadecimal address of the end of the current program space. The user can type it at any time during program development to see where the end of program is located in memory. The following example

ClibPDF - www.fastio.com

```
EOP : EOD
```

could result in

```
■ 4411
■ 441A
```

■ indicates that the number following is in hexadecimal.

This example indicates that the end of program is 4411 (HEX) and the end of data is at 441A (HEX). EOD is valid only after data has been generated by running the program (and after DIM has been executed).

FORMAT N

This statement specifies the field size (i.e. the number of places of printed numeric data. Any PRINT statement coming after the FORMAT statement, whether printing the value of a variable, or printing a number, will have the field size specified. For example

```
10    A = 12345.6
20    Format 7
30    PR A
40    PR 67890.1

RUN
```

will give,

```
12345.6
67890.1
```

If line 20 is replaced by,

```
20    FORMAT     6

RUN
```

the result is,

```
* * * * * *
* * * * * *
```

If line 40 is replaced by

```
40    PR      –67890.1
```

the result is

```
* * * * * *
– * * * * *
```

Summing up, if the field size of a positive number is greater than N, N asterisks will be printed. If the number is negative, a negative sign followed by (N–1) asterisks will be printed.

The range of n is from 1 to 15.

FORMAT 0 means that the constraint due to FORMAT is turned-off. For example,

```
10    FORMAT  5
20    PR      123456
30    FORMAT  0
40    PR      123456

RUN
```

will give,

```
* * * * *
1 2 3 4 5 6
```

ClibPDF - www.fastio.com

LIST

LIST expr

LIST expr, expr

This statement (list the program), with no expression, will list the entire program in user space. If one expression is given, only one line, the number of which corresponds to the value of the expression, will be listed. Likewise, if two expressions are given and separated by commas, the listing will start and end at the respective lines corresponding to the values of the expressions. If, at any time, an expression evaluates to a number which is a nonexistent line number, then the line with the nearest line number after the non-existent line will be listed. (see also section 3.3)

NEW

This statement totally initializes COMX 35 BASIC, the user space, and all data space. That is, it erases the user-generated COMX 35 BASIC program and initializes all string and array pointers.

RENUMBER

RENUMBER N

The RENUMBER statement allows the user to renumber the lines within a program to a given increment. If no argument follows the RENUMBER statement, the increment defaults to 10, i.e., line numbers 10, 20, 30, and so forth. When n is given, it becomes the initial line number as well as the increment. The n value that is displayed for numbers above 256 will always be module 256. A warning message indicating the number of "computed branches" is issued. A "computed branch" is a statement requiring the computer to branch or jump to another statement the line number of which is yet to be determined. For example, consider the following program,

```
10   FOR A=1 to 5
15   N(A) = 0
20   NEXT
25   GOTO 10
```

```
RENUMBER    20
```

will give the message

```
0 COMP BR
```

and LIST will give,

```
20   FOR A=1 to 5
40   N=0
60   NEXT
80   GOTO 20
```

Note that the lines have been renumbered. Note also that the statement "25 GOTO 10" has become "80 GOTO 20" since the old line 10 has been renumbered 20. "GOTO 10" is not a "computed branch" (hence the message "0 COMP BR") because the computer is able to determine the line to branch or jump to.

Consider another example, with "GOTO 10" replaced by "GOTO 5*2", (or "GOTO B"),

```
10   FOR A=1 to 5
15   N(A)=0
20   NEXT
25   GOTO 5*2
```

```
RENUMBER 20
```

will give the message,

```
1 COMP BR
```

ClibPDF - www.fastio.com

and LIST will give,

```
20    FOR A=1 to 5
40    N(A)=0
60    NEXT
80    GOTO 5*2
```

The warning message that there is one "computed branch" prints to the fact that one line, on this case, line 80, will give rise to error because of renumbering. "GOTO 5*2" (and "GOTO B") are examples of "computed branch".

RUN

This statement sets COMX 35 BASIC into the program execution mode. COMX 35 BASIC searches for the lowest line number and begins executing each line in numerical order. Before execution begins, however, the RUN statement clears all array and string data space to make room for new data.

RUN expr

This statement starts execution at a line number specified by expr. It is similar to RUN with no trailing expression in that it sets COMX 35 BASIC into the execution mode. If the line number specified by expr does not exist, an error code is generated. In addition, because this RUN statement does not clear the data space, the user can execute a program with data (strings or arrays) generated previously.

RUN+

This statement searches through the user's program and replaces "interpretive branches" with "absolute address branches" and then starts execution. This step greatly enhances speed. After the initial RUN+ cycle, RUN will cause execution in this faster mode. If any program editing occurs, the system automatically goes but to the slower mode.

ClibPDF - www.fastio.com

A program converted by RUN+ should not be saved in this form because of the absolute address assignments.

(Note: When RUN+ is executed, whenever the BASIC interpreter encounters a "branch or goto statement" such as "GOTO 10", it will find the absolute address where the line-number "10" is stored, and hence converts the statement from what is called an "interpretive branch" into an "absolute address branch".

A statement such as "GOTO B" is not an "interpretive branch" since it is known that the line-number to branch to is stored in the location with symbolic address B.)

TRACE expr

If the trailing expression evaluates to anything other than zero, the "trace" action is turned on. That is, each line that is executed by COMX 35 BASIC will transmit the following to the screen:

TR (line number)

The numbers on the screen are the line-numbers of each statement as it is executed. This enables the user to follow or "trace" the flow of the program, and is specially useful in the debugging stage to ensure that the program is doing what is intended.

If a TRACE statement is executed with the trailing expression evaluating to zero, the trace will shut off. These statements may be placed anywhere in a COMX 35 BASIC program. A very useful place for a TRACE statement is in an interrupt routine.

TRY the following with a BASIC program

```
TRACE (1): RUN
```

5.9 Comment and Definition Statements

Comment and definition statements allow the programmer to insert comments into the program and to configure the memory, that is, to define the starting address of the program, to allocate memory for data, and the like. The comment and definition statements include DEFINT, DEFUS, DEG, DIM, FIXED, RAD, and REM.

DEFINT

DEFINT var name

This statement (define integer variables) when executed without any variable name sets all variables (A–Z) and all arrays (A(expr)–Z (expr)) as floating point. If a variable name is included in the DEFINT statement, all variables and arrays from A up to and including the variable name will be set up as integer. For example,

```
DEFINT D
```

defines variables and array names A, B, C and D as integer. A NEW statement will always revert all variables to floating point. It is for this reason that, if integer variables are to be used in a program, a DEFINT statement should appear early in the program.

Note also that if a DEFINT is used with no variable name, it must end in a carriage return. That is, a colon may not be used to concatenate any further commands on the same line.

DEFUS expr

This statement (define the start of user space) is provided to allow the start of program space to be moved further up in memory, as shown in Fig. 5.9.1. It allows the programmer to create a "hole" in memory in which he can store machine language routines. Expr defines where the program space will begin. In COMX 35 BASIC the user space begins at hexadecimal 4400. The expression must evaluate to a number greater than 4400 (HEX). COMX 35 BASIC will round down the expression to give only even-page increments of movement. If an attempt is made to define the user space at an address lower than 4400 (HEX), COMX 35 BASIC will "self-destruct", i.e. since the program overwrites the system area (see Fig. 5.9.1), the BASIC interpreter will no longer work properly. Once a DEFUS statement is executed, the only way to get the program space back to 4400 (HEX) is by another DEFUS to 4400 (HEX). The statement destroys the user program currently in memory.

An interesting feature of moving the start of user program space lies in the PSAVE statement. If a program has been generated at a moved location and it has associated with it some machine language routines, a PSAVE statement will save everything from 4400 (HEX) to the end of program space. Included would be the machine language routines as well as the associated COMX 35 BASIC program. A subsequent PLOAD will load in all of the above, including the machine language routines, and will also redefine the start of user space to where it was when the file was created. No book-keeping is necessary.

DEFUS	■ 6D00	or
DEFUS	■ 6DCC	moves the start of user space to 6D00
DEFUS	■ 4400	moves the start of program space back to its original location

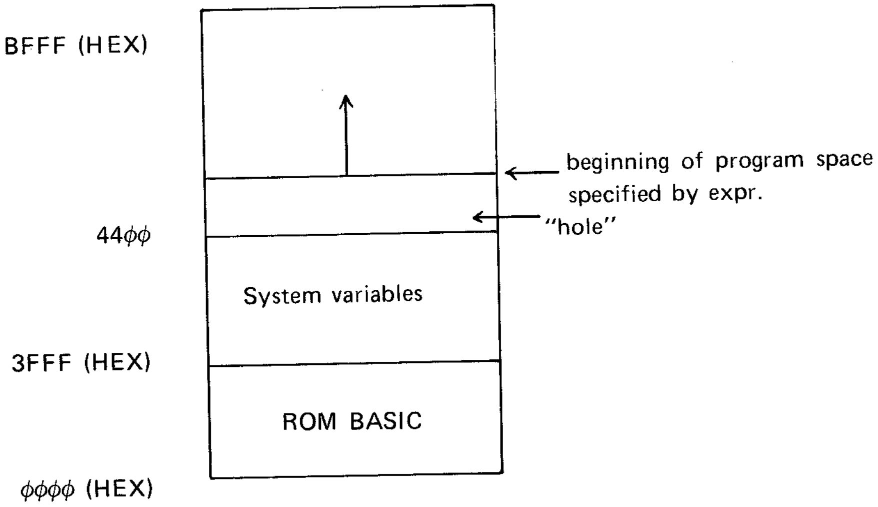


Fig. 5.9.1 Action by DEFUS to move the beginning of program space up

ClibPDF - www.fastio.com

DEG

This statement sets the unit of measure for angular functions to degrees. The preset state of COMX 35 BASIC is radian measure. The complement of this statement is RAD.

DIM var list

This statement (dimension arrays) serves to reserve memory for arrays of numbers. These arrays may be one dimensional or two dimensional. The expression defines how large the array is in one or both dimensions. See the earlier discussion of arrays and their size limitations. The mode of the array (floating point or integer) depends on whether its associated variable name has been defined as either integer or floating point. For instance, if all variables A–Z are floating point, then all arrays will be floating point (independent of the mode of the expression defining its limits). If variables A, B, and C are defined as integer, then arrays named A, B and C (if used) will be defined as integer. Multiple arrays may be dimensioned in the same DIM statement so long as each array is separated by a comma.

If memory is exceeded when an array is being dimensioned, an error message will result. Trying to use any array element that has not been previously dimensioned will also result in an error message. The programmer can use a CLD statement to clear all unwanted data space in order to reclaim all data space.

The following is an example of a DIM statement.

```
DIM A(10, 10), B(20)
```

The above example sets aside space for 100 numbers (10 x 10) with the names ranging from A(1, 1) to A(10, 10) and 20 numbers with names ranging from B(1) to B(20).

The dimension statement can be used in the direct mode of execution if desired. Any arrays generated by a program remain intact after program execution is completed. One can interrogate the contents of any array in the direct mode by means of the PRINT statement. The array remains intact until a CLD is executed or any editing to the program is done. Note that if a "RUN expr" is executed, the data space is not cleared and the array has not been deleted. In this case, redimensioning will result in an error message. For that reason, the programmer should begin execution at a line-number after the DIM statement, if he wants to use previously generated data. The array data as well as the string data, exists directly after the user space. Any editing of the original program will alter the user space and thus destroy the data. It should be noted that an array can be redimensioned without destroying other arrays or strings.

FIXED expr

This statement when executed formats the printing of all numbers both floating point and integer. The value of the expression defines how many digits to the right of the decimal point will be printed. Trailing zeroes will be filled in to complete the number. If necessary, the number will be rounded. The expression must evaluate to a number between 0 and 6. If a number greater than 6 is entered COMX 35 BASIC reverts to its normal mode (i.e. FORMAT 0) of outputing numbers. For example,

```
FIXED 2
PR 123         will result in 123.00
PR 123.567     will result in 123.57
PR 6E7         will result in .60E08
FIXED 7
PR 123.50      will result in 123.5
```

RAD

This statement sets the unit of measure for angular functions to radians. This state is the preset state of COMX 35 BASIC. The complement of this statement is DEG.

REM

When REM is placed at the start of any COMX 35 BASIC line, the entire line is listed but ignored during program execution. Its purpose is to enable the programmer to insert comments for documenting the listing.

ClibPDF - www.fastio.com

5.10 Program Data Statement

Program data statements allow the programmer to embed lists of data in the program and provide the mechanism to read the data during program execution. The program data statements include DATA, READ, and RESTORE.

DATA date list

The DATA statement contains data to be used by a READ statement. Each element in the DATA statement must be separated by a comma. Any string of characters must be enclosed in quotes. The DATA statement may appear anywhere in a COMX 35 BASIC program. The following are acceptable DATA statements.

```
   1 DATA 2*A, A$(1), "HELLO", B
1000 DATA 1,2,3,4
5000 DATA SIN(45), SIN(60), SIN(90)
```

READ var list

The READ statement is used to read data from the DATA statement and assign that data to a particular variable. Each time the READ statement requests more data, an internal COMX 35 BASIC pointer moves to the next piece of data in the DATA statement. When a DATA statement is out of data, COMX 35 BASIC will search onward for another DATA statement. If none is found, an error message will be returned indicating the lack of data. It is important to keep track of the variable names or string names in the READ statement and the corresponding data in the DATA statement. They must match in form; strings go with strings, and so on. An example of an acceptable DATA/READ statement pair is as follows.

```
 5    DIM A(4)
10    DATA 10,20,30,40
20    FOR A=1 TO 4
30    READ A(A)
40    NEXT
```

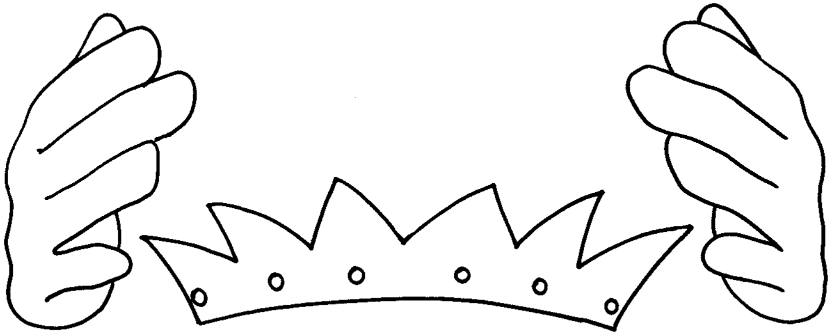
```
50    READ A$(1),B,C
60    PR A$(1); B*C
70    DATA "B*C IS EQUAL TO", 20,30
```

The result of the above example is that the array A would be loaded as follows:

```
        A(1)=10 A(2)=20 A(3)=30 A(4)=40
Line # 60 would result in
        B*C IS EQUAL TO 600
```

RESTORE

This statement resets the previously mentioned COMX 35 BASIC pointer back to the start of the first DATA statement in the program, thus allowing the multiple use of DATA in one program.



ClibPDF - www.fastio.com

5.11 I/O Statement

I/O (input/output) statements provide the interface between a program and the program user. The I/O statements include INPUT, PRINT and PR.

INPUT var list

When COMX 35 BASIC encounters an input statement, it stops and issues a question mark "?" to the screen as a prompt. It then waits for a user response. A list of variable names may appear after the INPUT statement. Each of the variable names must be separated by a comma. In response to the prompt "?", the user may answer with any expression or group of expressions separated by commas. If the INPUT statement has three variable names after it and the user responds with only two expressions, COMX 35 BASIC will issue another prompt (?) and wait for the third expression. Conversely, if the INPUT statement has only one variable name and the user responds with two or more expressions, COMX 35 BASIC will continue execution. When it encounters another INPUT statement it will pick up the last unused expression without issuing another prompt. It will continue to do so until all the unused inputted data is used up. Then, at that time, another prompt "?" will be issued. The data that is picked up is assigned to the associated variable name as it is inputted. The mode of the variable name defines the mode in which the associated input expression will be evaluated.

Note that an INPUT statement may not contain both string variable names and normal variable names at the same time. Furthermore, each INPUT statement containing a string variable name can contain one and only one name. Because of this limit, in response to an INPUT A$ statement, the programmer can answer with a string of characters with no quotation marks. The carriage return marks the end of the string. The following are acceptable INPUT statements.

```
10    INPUT A, B, C,(1)
10    INPUT A(1), A(B)
10    INPUT A$
10    INPUT B$(B)
```

The following are incorrect INPUT statements.

```
10    INPUT AB(no delimiter between variables)
10    INPUT A$, B$(multiple string names forbidden)
```

INPUT "string" var list

This INPUT statement is identical with the one above except that directly after the keyword INPUT a quote appears allowing a message to be printed prior to the question mark prompt. No delimiter or separator is used before the first quote or after the last quote.

PRINT or PR

PRINT print list

PR is used as an abbreviation for PRINT. In a listing, PR's are replaced with PRINT's. The purpose of this statement is to output to the I/O routines any expr or string function as defined earlier. Each of the items to be printed must be separated by an acceptable delimiter, either a comma or a semicolon. Each of the delimiters serves a special function. When a comma is used, the next item to be printed is placed at the next eighth column increment called "printing-zone". When a semicolon is used, no spaces are inserted and the next item is printed directly after the last. The semicolon and comma also serve to inhibit the carriage return at the end of PRINT statement. The next PRINT statement encountered in the program, therefore, will begin where the previous one left off. If a PRINT statement is used all by itself, then a carriage return/line feed is outputted. There are two functions that are usable only in a PRINT statement: TAB(expr) and CHR$(expr). A description of these two functions is given in sections 5.13 and 5.5 respectively. Examples of acceptable PR statements are as follows:

```
PRINT 5
PRINT A
PR (A+B)/C
PR A, B, 1234, C(5)
PR "THE VALUE OF A= ";A
PR A$(1)+A$(2)
```

ClibPDF - www.fastio.com

```
PR  A$(2),A,B;
PRINT  MID$(A$,1,2)
PRINT  TAB(A+B);10;TAB(30);E
PR  A,
PR
```

Note that when more than one arithmetic expression appears in a PRINT statement, each expression may be evaluated in a different mode.

5.12 Machine Language Subroutine Statement

Machine language subroutine statements allow the programmer to include machine language code within his program to take advantage of particular characteristics of the processor or for speed enhancements in real-time applications. The machine language subroutine statement is CALL.

```
CALL  (expr1)
CALL  (expr1,  expr2)
CALL  (expr1,  expr2,  expr3)
```

This statement provides the link between COMX 35 BASIC and machine language programming. It serves as a machine language subroutine call. It transfers execution to a machine language subroutine, the address of which is determined by expr1. The machine language routine should be written with the following rules in mind.

(1) . . . The program counter upon entry into the subroutine is R3.

(2) . . Transfer is made back to COMX 35 BASIC by means of a D5 (SEP R5) instruction.

(3) . . The machine language routines have free use of R8, RA, RC, RD, and RE. If any other registers are to be used, they should be saved first on the stack and restored before returning to COMX 35 BASIC.

Note: The standard call and return technique (SCRT) is not used because it destroys the upper part of register F(RF.1).

(4) . . Call and return conventions have been established by COMX 35 BASIC and no further initialization is required by the machine language subroutines.

(5) . . The stack is available for use (point to by R2) so long as it returns as it was left.

Any of the expressions (expr1, expr2, expr3) may be expressed in either integer or floating point. COMX 35 BASIC will automatically convert them to integer. The value of expr2, if used, is then passed to the machine in R8. A second piece of data may also be passed to the machine language subroutine in register RA. The value will be that of expr3. RD is initialized to timing constant for utility program.

5.13 I/O Functions

The function in this group include MEM, PEEK, and TAB.

MEM

This function enables the user to determine how much memory is left. When executed, MEM returns a decimal number representing the number of memory bytes left between the end of data (string and arrays) and the end of the normal stack. The actual number returned is reduced by 256 to allow for stack growth during program execution. For example, to receive the value on the screen the user should type

```
PR MEM
```

PEEK (expr)

This function returns the decimal equivalent of the contents of memory at an address determined by the expression. The decimal result defaults to floating point. For example:

```
10    A=PEEK (16842)
20    IF A=8 THEN PRINT "NTSC MACHINE"
30    IF A=9 THEN PRINT "PAL MACHINE"
```

Variable A now contains the "CONTENTS" stored at memory location 16842. This is a "system variable" which tells whether your COMX 35 is a PAL or NTSC machine.

ClibPDF - www.fastio.com

TAB (expr)

This function is not a function in the same sense as the previous ones in that it does not return a value of any kind. It is used and recognized only in the PRINT statement. It places the cursor at the horizontal position determined by the value of the expression. The new cursor position is referenced to column 0 and not to the previous cursor position. Printing continues from that point on.

Following are some examples.

```
PR TAB(10); 1
```

will print a 1 at column 10.

```
PR TAB(10);1; TAB(20);2
```

will print a 1 at column 10 and 2 at column 20.

```
PR TAB (A); "*"
```

will print a star (*) at column A.

5.14 Machine Language Function

The only function in this group is USR

```
USR (expr1)
USR (expr1, expr2)
USR (expr1, expr2, expr3)
```

This function acts like to CALL statement described in the previous section but with the difference that USR is a function to be used as part of an expression. When USR is encountered, a subroutine call is made to the machine language routine stored at expr1. Data may be passed to the subroutine in exactly the same way as the CALL statement.

Please note that USR and CALL use 1802 microprocessor conventions and registers, please consult an 1802 programming manual for a detailed explanation of the 1802 CPU.

ClibPDF - www.fastio.com

CHAPTER SIX

MORE PROGRAMMING EXAMPLES

CHAPTER 6 MORE PROGRAMMING EXAMPLES

This chapter contains a collection of programs. Besides the usefulness of the programs themselves, they are included as examples to illustrate the application of certain BASIC commands. For each program, besides providing the BASIC source listing, and the screen displays for a typical run, comments will be made on certain programming techniques used.

6.1 First example on CAI (Computer Aided Instruction) – demonstrating the use of the COMX 35 COLOR, SCREEN and MUSIC commands

6.1.1 The program listing

The program listing

```
10   REM THIS PROGRAM IS A TUTORIAL TO
20   REM SHOW HOW TO COMMAND COMX 35
30   REM TO CHANGE THE COLOR OF THE
40   REM CHARACTERS, AND OF THE SCREEN
50   REM AND HOW TO PRODUCE SOUND
60   REM EFFECT
70   REM
75   CPOS (0,0):CLS
80   SCREEN (3): COLOR (8): CPOS (1,7): PRINT "HI!
     I AM THE CLEVER COMX 35!"
90   CPOS (3,4): PRINT "I LIKE TO SHOW YOU"
100  CPOS (5,4): PRINT "HOW TO DO THE FOLLOWING:"
110  CPOS (7,4): PRINT "    1. HOW TO CHANGE THE
     COLORS"
120  PRINT "    OF THE CHARACTERS DISPLAYED"
140  CPOS (10,4): PRINT "    2. HOW TO CHANGE THE
     COLOR"
150  PRINT "   OF THE SCREEN."
160  CPOS (13,4): PRINT: "   3. HOW TO PRODUCE SOUND
     EFFECTS."
170  CPOS (15,2): INPUT "WHICH OF THE ABOVE WOULD
     YOU LIKE TO KNOW? 1,2 OR 3. ANSWER 0 TO END"
     K
```

ClibPDF - www.fastio.com

```
175  PRINT
177  IF K=0 THEN GOTO 710
180  GOTO K*200
200  SCREEN (1): COLOR (7): CPOS (0,0): CLS
204  CPOS (3,8): PRINT "TO CHANGE THE COLOR OF THE"
205  CPOS (4,11): PRINT "CHARACTERS, WE USE THE"
210  CPOS (5,11): PRINT "COMMAND COLOR(N) WHERE N"
215  CPOS (6,11): PRINT "DETERMINES"
220  CPOS (8,8): PRINT "  THE COLOR OF THE COMPUTER"
225  CPOS (9,9): PRINT "  OUTPUT CHARACTERS, AND"
230  CPOS(11,8): PRINT "  THE COLOR OF THE KEYBOARD"
235  CPOS (12,9):PRINT "  INPUT CHARACTERS"
240  CPOS (14,8): PRINT "PLEASE TYPE"
245  CPOS (16,8): PRINT "  COLOR (N)"
247  CPOS (18,8): PRINT "(WHERE N CAN BE 2,3,4,5,6,7,8)"
250  CPOS  (20,8): PRINT  "AFTER THE   ? SIGN.  THEN.
     PRESS"
255  CPOS  (21,11):  PRINT  "THE  CR  KEY"
260  INPUT  A$
265  B$=MID$(A$,7,1)
270  C=FVAL  (B$)
275  COLOR  (C)
280  PRINT  "TYPE  ANY  KEY,  FOLLOWED  BY"
283  PRINT  "PRESSING  CR  TO  REVERT  TO"
285  PRINT  "ORIGINAL  COLOR"
290  INPUT  A$
295  COLOR (12)
300  PRINT  "N  CAN  HAVE  VALUE  FROM  1  TO  12"
305  PRINT  "INCLUSIVE.  YOUR  MANUAL  WILL"
310  PRINT  "TELL  YOU  WHAT  COLOR"
315  PRINT  "COMBINATION  FOR  EACH  VALUE"
320  PRINT  "OF  N."
325  CPOS  (0,0):  CLS:  GOTO  80
400  SCREEN (7): COLOR (12): CPOS (0,0): CLS
404  CPOS  (6,4): PRINT "TO CHANGE THE BACKGROUND
     COLOR"
405  CPOS (8,4): PRINT "OF THE SCREEN, WE USE THE"
410  CPOS (10,4): PRINT "SCREEN(M) COMMAND."
415  CPOS (13,4): PRINT "FOR EXAMPLE' PLEASE TYPE"
420  CPOS (16,4): PRINT "  SCREEN(M)"
```

```
422  CPOS (19,4): PRINT "(WHERE M IS ANY INTEGER 1
     TO 8)"
425  CPOS (21,4): PRINT "AFTER THE ? SIGN. AND PRESS
     CR"
430  INPUT A$
435  B$=MID$(A$,8,1)
440  C=FVAL (B$)
445  SCREEN (C)
450  PRINT "PRESS ANY KEY FOLLOWED BY CR"
455  PRINT "REVERT TO THE ORIGINAL SCREEN"
460  PRINT "COLOR"
465  INPUT A$
470  SCREEN (1)
475  PRINT "M CAN HAVE VALUES OF 1 TO 8"
480  PRINT "INCLUSIVE. YOUR MANUAL WILL"
485  PRINT "TELL YOU THE SCREEN COLOR"
490  PRINT "CORRESPONDING TO EACH"
495  PRINT "VALUE OF N"
500  CPOS (0,0): CLS: GOTO 80
600  SCREEN (2): COLOR (5): CPOS (0,0): CLS
604  CPOS  (3,4): PRINT "MUSICAL NOTE IS GENERATED
     USING"
605  CPOS (5,4): PRINT "THE MUSIC (X,Y,Z) COMMAND"
610  CPOS (8,4): PRINT "  X=1 TO 7 FOR DO TO TEE"
615  CPOS (10,4): PRINT "  Y=1 TO 8 FOR OCTAVE"
620  CPOS (12,4): PRINT "  Z=0 TO 9 FOR AMPLITUDE"
625  CPOS (15,4): PRINT "TRY, FOR EXAMPLE, TYPING,"
630  CPOS (18,4): PRINT "  MSUIC (X,Y,Z)"
635  CPOS (21,4): PRINT "AFTER THE ? SIGN. THEN PRESS
     CR"
640  INPUT A$
645  X$=MID$ (A$,7,1)
650  Y$=MID$(A$,9,1)
655  Z$=MID$(A$,11,1)
660  X=FVAL (X$)
665  Y=FVAL (Y$)
670  Z=FVAL (Z$)
675  MUSIC (X,Y,Z)
680  PRINT "HIT ANY KEY FOLLOWED BY CR TO"
685  PRINT "STOP THE MUSIC NOTE"
```

ClibPDF - www.fastio.com

```
690  INPUT A$
700  MUSIC (0,0,0)
705  CPOS (0,0): CLS: GOTO 80
710  SCREEN (1): COLOR (12): END
```

6.1.2 Notes on programming tips

(1) . . Lines 10 to 60 Note the use of REM to describe the aim of the program.

(2) . . Line 75 This clears the screen for subsequent display.

(3) . . Line 80 Note the use of SCREEN and COLOR to change the colors of the display.

(4) . . Lines 80 to 160 Note the use of CPOS to position the messages to be displayed.

(5) . . Line 170 This INPUT statement outputs a message to explain to the user how to "talk" to the computer. In this case, he is asked to choose an option by answering 1,2,3 or 0.

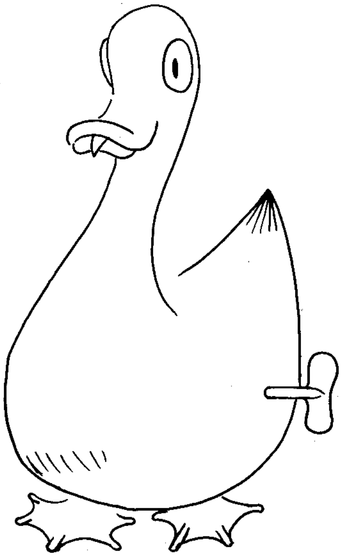
(6) . . Line 260 The computer accept the string A$ which has the form COLOR (N) from the user. Let's assume that the user enters COLOR (6).

(7) . . Lines 265 to 275 The computer has to be instructed to execute the COLOR (6) instruction as requested by the user. These three lines do just that using the MID$ and FVAL functions. Line 265 extracts the character 6 from the string COLOR (6). Hence B$ equals the ASCII code of the character 6. In line 270, FVAL evaluates the character 6 and returns its value to C. In this case C=6, and COLOR (C) does the color change as required. The same technique is repeated in lines 435 to 445, and lines 645 to 675.

ClibPDF - www.fastio.com

6.2 Second example program on CAI (Computer Aided Instruction) – using COMX 35 to evaluate mathematical expressions

6.2.1 The program listing



The program listing

```
10   REM THIS PROGRAM IS A TUTORIAL TO
20   REM SHOW HOW COMX MAY BE USED
30   REM AS A CALCULATOR TO EVALUATE
40   REM COMPLEX MATHEMATICAL EXPRESSIONS
60   CPOS (0,0): CLS
70   CPOS (4,5): PRINT "I WISH TO SHOW YOU MY MATH"
80   CPOS (5,6): PRINT "   ABILITY"
90   CPOS (7,5): PRINT "LET'S BEGIN WITH ADDITION."
100  CPOS (8,5): PRINT "THE OPERATOR IS +."
130  CPOS (10,5): PRINT "PLEASE TYPE"
140  CPOS (12,5): PRINT "3+4"
150  CPOS (14,5): PRINT "AFTER THE? SIGN."
160  CPOS (17,5): PRINT "THEN, PRESS THE CR KEY"
165  CPOS (19,5): PRINT "WATCH OUT FOR THE ANSWER
     IN"
```

```
167  CPOS (20,6): PRINT "  THE NEXT LINE"
170  INPUT A$
180  PRINT FVAL (A$)
182  INPUT "HIT THE RETURN KEY AND PRESS CR TO
     CONTINUE" K$
184  CPOS (0,0): CLS
190  CPOS (8,4): PRINT "NOW, LET'S TRY SOMETHING
     MORE"
200  CPOS (9,6): PRINT "COMPLICATED"
210  CPOS (11,4): PRINT "TYPE AFTER THE? SIGN"
220  CPOS (13,4): PRINT "   12+14+61"
221  CPOS (15,4): PRINT "THEN, PRESS CR FOR THE
     ANSWER"
224  PRINT
226  PRINT
230  INPUT  A$
240  PRINT  FVAL  (A$)
242  INPUT "HIT ANY KEY AND PRESS CR TO CONTINUE"
     A$
250  CPOS (0,0): CLS
260  CPOS (8,4): PRINT "THE OPERATOR IS–."
270  CPOS (11,4): PRINT "PLEASE TYPE"
280  CPOS (13,4): PRINT "  9–5  "
290  CPOS (15,4): PRINT "AFTER THE? SIGN. PRESS CR"
300  PRINT A$
310  PRINT FVAL (A$)
315  INPUT "HIT ANY KEY AND PRESS CR TO CONTINUE"
     K$
320  CPOS (0,0): CLS: CPOS (7,8): PRINT "NOW, TRY THE
     FOLLOWING"
330  CPOS (9,8): PRINT "TYPE AFTER THE? SIGN"
340  CPOS (12,8): PRINT "  5+6–7  "
345  CPOS (15,8): PRINT "AND PRESS CR FOR THE
     ANSWER"
350  INPUT  A$
360  PRINT  FVAL  (A$)
365  INPUT "HIT ANY KEY AND PRESS CR TO CONTINUE"
     K$
370  CPOS (0,0): CLS: CPOS (5,5): PRINT "NOW, LET'S
     TRY MULTIPLICATION"
380  CPOS (7,5): PRINT "THE OPERATOR IS*"
```

```
390  CPOS (9,5): PRINT "PLEASE TYPE"
400  CPOS (12,5): PRINT "9*7"
410  CPOS (15,5): PRINT "AFTER THE? SIGN. PRESS CR"
420  INPUT A$
430  PRINT FVAL (A$)
435  INPUT "HIT ANY KEY AND PRESS CR TO CONTINUE"
     K$
440  CPOS (0,0): CLS: CPOS (5,5): PRINT "NOW LET'S TRY
     DIVISION"
450  CPOS (7,5): PRINT "THE OPERATOR IS /"
460  CPOS (9,5): PRINT "PLEASE TYPE"
465  CPOS (12,5): PRINT "  63/7  "
470  CPOS (15,5): PRINT: AFTER THE? SIGN. PRESS CR"
480  CPOS (17,5): PRINT "(63/7 MEANS 63 DIVIDED BY  7)"
490  INPUT A$
500  PRINT FVAL (A$)
520  CPOS (20,0): INPUT "HIT ANY KEY AND PRESS CR TO
     CONTINUE" K$
522  CPOS (0,0): CLS: CPOS(2,5): PRINT "NOW, LET'S TRY
     A MORE COMPLEX"
530  CPOS (4,5): PRINT "EXPRESSION. PLEASE TYPE"
540  CPOS (7,5): PRINT "  (5+6)*(9–7)  "
550  CPOS (10,5): PRINT "AFTER THE? SIGN. PRESS CR"
560  INPUT A$
570  PRINT FVAL (A$)
575  INPUT "HIT ANY KEY AND PRESS CR TO CONTINUE"
     K$
580  CPOS (0,0): CLS: CPOS(8,4): PRINT "NOW, TRY
     ANOTHER. PLEASE TYPE"
590  CPOS (11,4): PRINT "  (7+4)*(19–14)/(8–3)  "
600  CPOS (13,4): PRINT "AFTER THE? SIGN. PRESS CR"
610  INPUT A$
620  PRINT FVAL (A$)
625  INPUT "HIT ANY KEY AND PRESS CR TO CONTINUE"
     K$
630  CPOS (0,0): CLS: CPOS (5,5): PRINT "PLEASE FEEL
     FREE TO TRY ANY OF"
640  CPOS (7,5): PRINT "YOUR OWN EXPRESSION"
650  CPOS (10,5): PRINT "JUST TYPE IT AFTER THE? SIGN"
660  CPOS (12,5): PRINT "AND PRESS THE CR KEY"
670  INPUT A$
```

ClibPDF - www.fastio.com

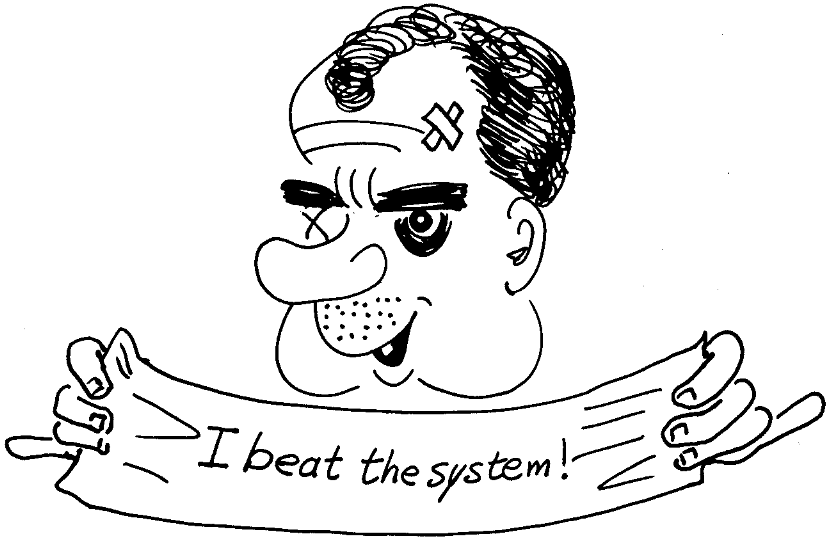
```
680  PRINT FVAL (A$)
690  INPUT "ANOTHER EXPRESSION, ANSWER Y/N"Q$
692  IF Q$="Y" THEN GOTO 630
694  IF Q$="N" THEN GOTO 720
720  CPOS (0,0): CLS: CPOS (5,5): PRINT "AFTER YOU
     FINISH THIS TUTORIAL"
730  COPS (7,5): PRINT "YOU CAN USE COMX TO COMPUTE"
740  CPOS (9,5): PRINT "ANY EXPRESSION OF YOUR OWN."
750  CPOS (12,5): PRINT "JUST TYPE PR OR PRINT.
     FOLLOWED"
760  CPOS (14,5): PRINT "BY THE EXPRESSION. THEN
     PRESS"
770  CPOS (16,5): PRINT "THE CR KEY. FOR EXAMPLE,"
780  CPOS (19,5): PRINT "PR6*5/9"
785  CPOS (23,1): INPUT "HIT ANY KEY AND PRESS CR
     TO CONTINUE" K$
790  CPOS (0,0): CLS: CPOS (5,5): PRINT "COMX CAN IN
     FACT HANDLE"
800  CPOS (7,5): PRINT "TRIGONOMETRIC, LOG, EXPONEN-
     TIAL"
810  CPOS (9,5): PRINT "AND MANY OTHER FUNCTIONS."
820  CPOS (12,5): PRINT "PLEASE CONSULT THE MANUAL"
830  CPOS (15,7): PR INT "USING THE COMX 35"
840  CPOS (20,7): PRINT "GOOD BYE! HAVE FUN"
850  END
```

6.2.2 Notes on programming tips

In line 140, the user is asked to input an arithmetic expression 3+4. Hence, string A$ is 3+4. The statement PRINT FVAL(A$) in line 180 evaluates the arithmetic expression 3+4.

More complex arithmetic expressions may be evaluated in this manner. The same technique is used throughout the program in lines 230, 240, lines 300, 310 etc.

Line 690 is an INPUT statement allowing the user to make a choice by answering Y (for yes) or N (for no). Different actions are taken in lines 692 and 694.



ClibPDF - www.fastio.com

APPENDIX

ClibPDF - www.fastio.com

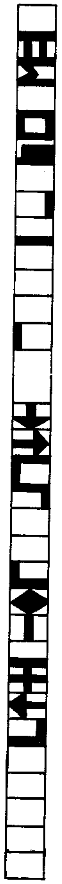
A.1 COMX 35 ERROR MESSAGE

00 Program HALTED by USER
01 Syntax error in ASC or LEN function
02 ARRAY out of RANGE or NOT DIMENSIONED
03 DIMENSION error
04 DEFINT has illegal ending
05 PARENTHESES missing in ARGUMENT
06 ARGUMENT out of range
07 MIXED MODE calculation encountered
08 DIVIDE by ZERO error, or LOG of NEGATIVE NUMBER
09 NON-EXECUTABLE function encountered
10 EXIT command must be used with FOR/NEXT or GOSUB/RETURN
11 FOR/NEXT stack overflow, or FOR/NEXT executed directly.
12 Syntax error in FOR
13 GOSUB stack overflow
14 UNACCEPTABLE character in HEXADECIMAL number
15 FLOATING POINT NUMBER too large to be converted to INTEGER or INTEGER MULTIPLY overflow
16 UNACCEPTABLE OPERATOR in CONDITIONAL statement
17 INPUT or FVAL cannot be directly executed
18 Must have VARIABLE or STRING name in READ
19 Syntax error in READ or INPUT
20 Syntax error in LEN function
21 Syntax error in ASSIGNMENT statement
22 Missing QUOTE
23 Syntax error in LIST
24 No such WORD found in LIBRARY
25 Syntax error in MID$ function
26 UNACCEPTABLE variable name found in NEXT statement
27 Either a NUMBER or a LETTER is EXPECTED
28 Missing arithmetic PARENTHESES
29 Wrong number of ARGUMENTS in POKE statement
30 UNACCEPTABLE last character in PRINT statement
31 Syntax error in DATA statement
32 No more DATA found
33 No such STRING found in INPUT statement
34 Missing EQUAL sign in ASSIGNMENT statement

ClibPDF - www.fastio.com

35 Missing PARENTHESES in STRING ARRAY
36 Too many ARGUMENTS in USR or CALL
37 Syntax error in CHR$ function
38 UNACCEPTABLE character in BINARY number
39 Line buffer OVERFLOW
40 File not opened for INPUT
41 File not opened for OUTPUT
42 UNACCEPTABLE line end or NON-EXECUTABLE statement
43 STACK OVERFLOW
44 Too many DIGITS in number
45 UNACCEPTABLE character in NUMBER fold
46 No such LINE NUMBER found
47 UNACCEPTABLE operation in IF statement
48 MEMORY OVERFLOW
49 Wrong number of arguments in MOD statement
50 Program TOO LARGE for memory
51 ARGUMENT out of RANGE
52 Wrong number of arguments
53 Wrong number of arguments
54 STRING variable not defined
55 TAPE READ error
56 TAPE WRITE error
57 FILE is not a BASIC program
58 FILE is not BASIC data
59 RESERVED
60 RESERVED
61 RESERVED
62 ROM or ROM CARD not PRESENT
63 Not enough MEMORY for RENUMBER to OPERATE
64 RENUMBER located LINE NUMBER error
65 No such SUBSCRIPT VARIABLE defined
66 STRING is over 127 characters
67 NUMBER OF ARGUMENTS in COLOR MUST BE 1
68 NUMBER OF ARGUMENTS in SCREEN MUST BE 1
69 NUMBER OF ARGUMENTS in CTONE MUST BE 1
70 NUMBER OF ARGUMENTS in VOLUME MUST BE 1
71 NUMBER OF ARGUMENTS in NOISE MUST BE 2
72 NUMBER OF ARGUMENTS in TONE or MUSIC MUST BE 3

A. 2 Key ASCII codes and the built-in characters



			GRAPHIC Output	
Decimal	Hex	Key pressed	Shape	Color
0	0			
1	1	CNTL–A		C
2	2	CNTL–B		C
3	3			
4	4	CNTL–D		C
5	5	CNTL–E		C
6	6	CNTL–F		C
7	7	CNTL–G		C
8	8	CNTL–H		C
9	9			
10	A	LINE FEED		
11	B			
12	C	CNTL–L		C
13	D	CARRIAGE RETURN		
14	E	CNTL–N		C
15	F	CNTL–O		C
16	10	CNTL–P		C
17	11	CNTL–Q		C
18	12			
19	13			
20	14	CNTL–T		C
21	15	CNTL–U		C
22	16	CNTL–V		C
23	17	CNTL–W		C
24	18	CNTL–X		C
25	19	CNTL–Y		C
26	1A	CNTL–Z		C
27	1B			
28	1C			
29	1D			
30	1E			
31	1F			

			STANDARD Output	
Decimal	Hex	Key pressed	Shape	Color
32	20	Space Bar		
33	21	!	!	C
34	22	"	"	C
35	23	#	#	C
36	24	$	$	C
37	25	%	%	C
38	26	&	&	C
39	27	'	'	C
40	28	(	(	C
41	29	)	)	C
42	2A	*	*	C
43	2B	+	+	C
44	2C	,	,	C
45	2D	—	—	C
46	2E	.	.	C
47	2F	/	/	C
48	30	ϕ	ϕ	C
49	31	1	1	C
50	32	2	2	C
51	33	3	3	C
52	34	4	4	C
53	35	5	5	C
54	36	6	6	C
55	37	7	7	C
56	38	8	8	C
57	39	9	9	C
58	3A	:	:	C
59	3B	;	;	C
60	3C	<	<	C
61	3D	=	=	C
62	3E	>	>	C
63	3F	?	?	C

Note: CNTL–A stands for CONTROL A.

ClibPDF - www.fastio.com

Decimal	Hex	Key pressed	Shape	Output Color
64	40	■	■	C
65	41	A	A	C
66	42	B	B	C
67	43	C	C	C
68	44	D	D	C
69	45	E	E	C
70	46	F	F	C
71	47	G	G	C
72	48	H	H	C
73	49	I	I	C
74	4A	J	J	C
75	4B	K	K	C
76	4C	L	L	C
77	4D	M	M	C
78	4E	N	N	C
79	4F	O	O	C
80	50	P	P	C
81	51	Q	Q	C
82	52	R	R	C
83	53	S	S	C
84	54	T	T	C
85	55	U	U	C
86	56	V	V	C
87	57	W	W	C
88	58	X	X	C
89	59	Y	Y	C
90	5A	Z	Z	C
91	5B	[	[	C
92	5C	\	\	C
93	5D	]	]	C
94	5E	^	↑	C
95	5F	□	▁	C

Decimal	Hex	Key pressed	Shape	Output Color
96	60		ϕ	G
97	61	SHIFT–A	1	G
98	62	SHIFT–B	2	G
99	63	SHIFT–C	3	G
100	64	SHIFT–D	4	G
101	65	SHIFT–E	5	G
102	66	SHIFT–F	6	G
103	67	SHIFT–G	7	G
104	68	SHIFT–H	8	G
105	69	SHIFT–I	9	G
106	6A	SHIFT–J	ϕ	G
107	6B	SHIFT–K	■	G
108	6C	SHIFT–L	■	B
109	6D	SHIFT–M	■	Blk
110	6E	SHIFT–N	♟	C
111	6F	SHIFT–O	◣	C
112	70	SHIFT–P	☰	C
113	71	SHIFT–Q	◢	C
114	72	SHIFT–R	♪	C
115	73	SHIFT–S	▲	C
116	74	SHIFT–T	C	C
117	75	SHIFT–U	O	C
118	76	SHIFT–V	M	C
119	77	SHIFT–W	X	C
120	78	SHIFT–X	▁	G
121	79	SHIFT–Y	C	C
122	7A	SHIFT–Z	O	G
123	7B		M	B
124	7C		□	
125	7D		═	G
126	7E		©	G
127	7F		◆	B

SPECIAL CHARACTERS (HEX 80 – HEX 8F)

Decimal	Hex	Used as output	Read from keyboard
128	80	Cursor Up (Vertical tab)	
129	81	Cursor right	
130	82	Cursor down (Line feed)	
131	83	Cursor left	
132	84	Carriage return	
133	85	Clear to end of screen	
134	86		Delete Key
135	87		Control–S
136	88		Joystick up
137	89		Joystick right
138	8A		Joystick down
139	8B		Joystick left
140	8C		Control–R
141	8D		Control–C
142	8E		
143	8F		

ClibPDF - www.fastio.com

GRAPHIC

Decimal	Hex	Key pressed	Shape	Output Color
144	90		▛	W
145	91		▐	W
146	92		□	
147	93		□	
148	94		▙	W
149	95		⬢	W
150	96		┃	W
151	97		▁	W
152	98		╪	W
153	99		←	W
154	9A		▟	W
155	9B		□	
156	9C		□	
157	9D		□	
158	9E		□	
159	9F		□	

STANDARD

Decimal	Hex	Key pressed	Shape	Output Color
160	Aϕ			
161	A1		!	W
162	A2		"	W
163	A3		#	W
164	A4		$	W
165	A5		%	W
166	A6		&	W
167	A7		'	W
168	A8		(	W
169	A9		)	W
170	AA		*	W
171	AB		+	W
172	AC		,	W
173	AD		—	W
174	AE		.	W
175	AF		/	W
176	Bϕ		ϕ	W
177	B1		1	W
178	B2		2	W
179	B3		3	W
180	B4		4	W
181	B5		5	W
182	B6		6	W
183	B7		7	W
184	B8		8	W
185	B9		9	W
186	BA		:	W
187	BB		;	W
188	BC		<	W
189	BD		=	W
190	BE		>	W
191	BF		?	W

Decimal	Hex	Key pressed	Shape	Output Color
192	Cϕ		▀	W
193	C1		A	W
194	C2		B	W
195	C3		C	W
196	C4		D	W
197	C5		E	W
198	C6		F	W
199	C7		G	W
200	C8		H	W
201	C9		I	W
202	CA		J	W
203	CB		K	W
204	CC		L	W
205	CD		M	W
206	CE		N	W
207	CF		O	W
208	Dϕ		P	W
209	D1		Q	W
210	D2		R	W
211	D3		S	W
212	D4		T	W
213	D5		U	W
214	D6		V	W
215	D7		W	W
216	D8		X	W
217	D9		Y	W
218	DA		Z	W
219	DB		E	W
220	DC		╲	W
221	DD		⊐	W
222	DE		↑	W
223	DF		▁	W

Decimal	Hex	Key pressed	Shape	Output Color
224	Eϕ		ϕ	Y
225	E1		1	Y
226	E2		2	Y
227	E3		3	Y
228	E4		4	Y
229	E5		5	Y
230	E6		6	Y
231	E7		7	Y
232	E8		8	Y
233	E9		9	Y
234	EA		ϕ	Y
235	EB		■	Y
236	EC		■	M
237	ED		■	R
238	EE		▲	W
239	EF		◀	W
240	Fϕ		≡	W
241	F1		⊐	W
242	F2		♪	W
243	F3		⬆	W
244	F4		C	W
245	F5		O	W
246	F6		M	W
247	F7		X	W
248	F8		▁	Y
249	F9		C	W
250	FA		O	Y
251	FB		M	M
252	FC		X	R
253	FD		▬	Y
254	FE		©	W
255	FF		◆	M

ClibPDF - www.fastio.com

Special Keyboard Command

Control – C	Cancel input line
Control – I	Cursor up
Control – J	Cursor left
Control – K	Cursor right
Control – M	Cursor down
Control – R	Repeat last line (until 'return' key)
Control – S	Stop edit

Color Abbreviations

Blk	– Black
B	– Blue
G	– Green
C	– Cyan
R	– Red
Y	– Yellow
M	– Magenta
W	– White

A.3 Important cassette recording guidelines

1. . . Use high quality tapes.

2. . . Use shortest tapes possible.

3. . . Use the supplied cables, which are specially shielded.

4. . . Keep the heads and the pinch rollers clean.

5. . . Keep the heads aligned for tape interchangability.

6. . . Avoid recording too close to the beginning of tapes.

7. . . Make sure the cassette is properly seated in the recorder.

8. . . If you have trouble with a cassette tape, try another one. You can have a bad spot on the tape or a warped cassette.

9. . . Try to find the best volume setting for loading programs.

10. . . A dirty recorder mechanism can cause tape problems.

11. . . Make sure the recorder connection plugs make good contact.

12. . . Rewind the tape before removing them from the recorder.

13. . . Store tapes in original dust-proof containers.

14. . . Avoid exposing the tapes to heat or magnetic fields.

15. . . Before recording, wind the tape to one end.

16. . . Some cassette recorders will give problems once in a while (They don't like certain cassettes, etc.). If one unit gives you problems most of the time, try another one.

17. . . Make sure that the MIC plug is connected before recording.

18. . . You may have to record with the EAR plug out for some tape recorders, check the recorder manual.

ClibPDF - www.fastio.com

19. . . Always use the AC adaptor with the recorder for best results, unless the batteries are fully charged.

20. . . When a tone control is available, adjust it to the highest possible setting.

21. . . Demagnetize the head and pinch rollers periodically if possible. Magnetized header tends to slowly erase tape contents.

22 . . . Listen to your program by unplugging the ear cord. An uneven header tone indicates cassette/head problems.

23. . . For the best results, demagnetise your cassette recorders periodically.

A.4 INDEX OF COMX 35 STATEMENTS AND FUNCTIONS

ClibPDF - www.fastio.com

A.5 NOTES ON HARDWARE

A.5.1. POWER UP SELF–DIAGNOSTIC SEQUENCE

On initial power-up or after system reset (press SPACE BAR and RT simultaneously), your COMX 35 computer will generate several beeps and display a test pattern on the TV screen. This is a self-test sequence and it contains a lot of information about your COMX 35 computer. Our service personnel can ·deduce the source of problems should any servicing be required.

This test pattern is also useful in adjusting your TV receiver. Colors will change at a rate of about once every 7 seconds and you should adjust the TUNING, BRIGHTNESS, CONTRAST, COLOR and HUE control of your TV set for the best color picture. After a good TV picture is obtained, you can press any key, except the space bar, the COMX 35 will display its software version number and is now ready for your commands.

A.5.2. PAL AND NTSC VERSION

The COMX 35 computer has 2 versions which are specially designed for PAL and NTSC television system. There are several differences between these two versions and users are advised to note the following points:

1. . . Each character in PAL version is made up of 9 horizontal lines while characters in NTSC version consist of 8 horizontal lines. This makes some graphic characters look differently since the last (9th) horizontal line is not displayed in the NTSC version of COMX 35.

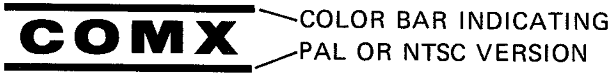
This will also affect the SHAPE command since the 17th and 18th digits are not used to define the shape of symbols or characters, in the NTSC system.

2. . . The internal timer is decremented at a rate of 50 times per second in the PAL version. It is decremented at a rate of 60 times per second in the NTSC version. Users should use different values in the TIME(X) command if accurate timing is needed.

ClibPDF - www.fastio.com

A.5.3. DIFFERENTIATION OF PAL AND NTSC VERSION

The user can tell whether his COMX 35 is a PAL version or a NTSC version machine by just looking at the test pattern after power-up or system reset. The two horizontal lines will show a different color sequence in the PAL or the NTSC version.



	PAL version					NTSC version				
	C	O	M	X	BAR	C	O	M	X	BAR
Picture 1	C	Y	M	R	G	C	Y	M	R	Y
Picture 2	Y	C	M	B	G	Y	C	M	B	C
Picture 3	M	C	Y	G	B	M	C	Y	G	C

C=CYAN	Y=YELLOW	M=MAGENTA
R=RED	G=GREEN	B=BLUE

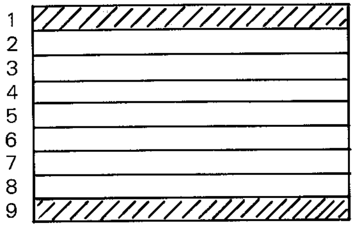
If a program is to be run on both PAL and NTSC machines with identical results, the program must first check the machine version by looking at the system parameter located at address 41CA (HEXIDECIMAL). If the content is 9, it is a PAL machine. If the content is 8, it is an NTSC machine.

Users can access this parameter with the PEEK command and do the appropriate actions as illustrated by the following example.

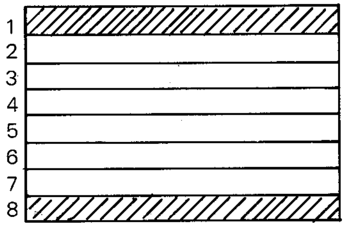
```
10    V = PEEK (■ 41CA) :  REM find out the version and
                           put into variable V.

20    IF V = 8 THEN TIME (120)  :  REM 2 seconds delay
                                   in NTSC system.

30    IF V = 9 THEN TIME (100)  :  REM    2    seconds
                                   delay in PAL system.
```

```
40    IF V = 8 THEN SHAPE (20,'FF000000000000FF')

50    IF V = 9 THEN SHAPE (20,'FF00000000000000FF')
```

This program example first finds out the version and puts it into variable V. In lines 20 and 30, there are different time delay constants for PAL and NTSC machines so that both machines can have an accurate 2 seconds delay. In lines 40 and 50, the program will create a graphic symbol with the top and bottom lines. This will look identical on TV screens although they are made up of 8 or 9 horizontal lines respectively.



PAL version



NTSC version

A.5.4. ADJUSTMENT OF COMX 35 COMPUTER

Every COMX 35 computer has been factory-adjusted to give the best results. If any adjustment has been disturbed, re-adjustment may be made with a small screw-driver (those used by watch-repairman) through 3 small holes located at the bottom side of the computer.

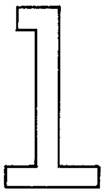
These three holes are marked MIX, SYNC and VIDEO respectively. Their functions and adjustment procedures are listed below.

(I) MIX

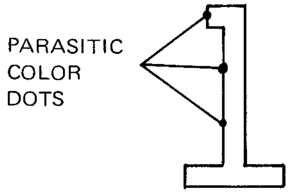
This is used to adjust the frequency of the color subcarrier signal so as to minimize the interference between the luminance and chrominance signals.

ClibPDF - www.fastio.com

If this control is disturbed, moving or stationary color dots will appear on the edges of characters displayed.



NORMAL DISPLAY



'MIX' MIS-ADJUSTED

PROCEDURE FOR ADJUSTMENT OF "MIX" CONTROL

(a)... Type in the program below and vertical bars will be displayed.

(b)... Carefully insert a small screw-driver into the hole marked "MIX".

(c)... Observe the edges of the number "1" and the boundaries between color bars; turn the screw-driver slowly and carefully.

(d)... Adjust until the parasitic color dots disappear.
Note: TV display may suddenly become BLACK and WHITE in some machines; rotate the screw-driver in the opposite direction and color will re-appear.

```
NEW
10      SHAPE (#40, "FFFFFFFFFFFFFFFF")
20      PRINT "11111111111111111111111111111111
        111111111111111111";
30      FOR L1=1 TO 22
40      PRINT CHR$ (107,108,64,237,235,236,192);
50      FOR I=1 TO 8
60      READ A
70      FOR D=1 TO 4:PRINT CHR$(A); : NEXT
80      NEXT
90      PRINT "       ";
```

ClibPDF - www.fastio.com

```
100     RESTORE
110     NEXT
120     REM
130     GOTO 120
140     DATA 32,107,108,64,237,235,236,192:END

READY
:RUN
```

(II) SYNC

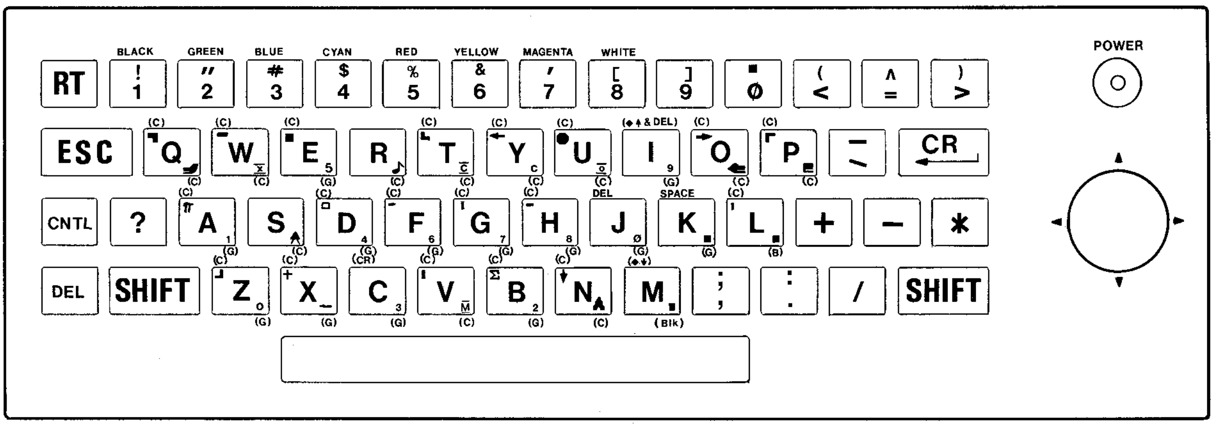
This is used to adjust the amplitude of synchronizing pube. Adjustment of this control is not necessary, unless it has been mis-adjusted by user. Re-adjustment should be done by qualified service personnel.

(III) VIDEO

This is used to adjust the amplitude of the video signal feeding into the RF modulator. Low amplitude will give a dim display while high amplitude will make yellow characters appear as white on TV screen.

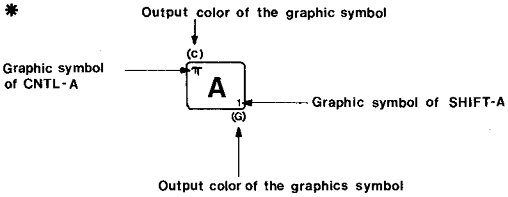
Adjustment of this control is not necessary unless it has been mis-adjusted by user. Re-adjustment should be done by qualified service personnel.

BUILT-IN CHARACTERS, A QUICK REFERENCE GUIDE



Note :

*


Output color of the graphic symbol
Graphic symbol of CNTL-A
(C)
A
1
(G)
Graphic symbol of SHIFT-A
Output color of the graphics symbol

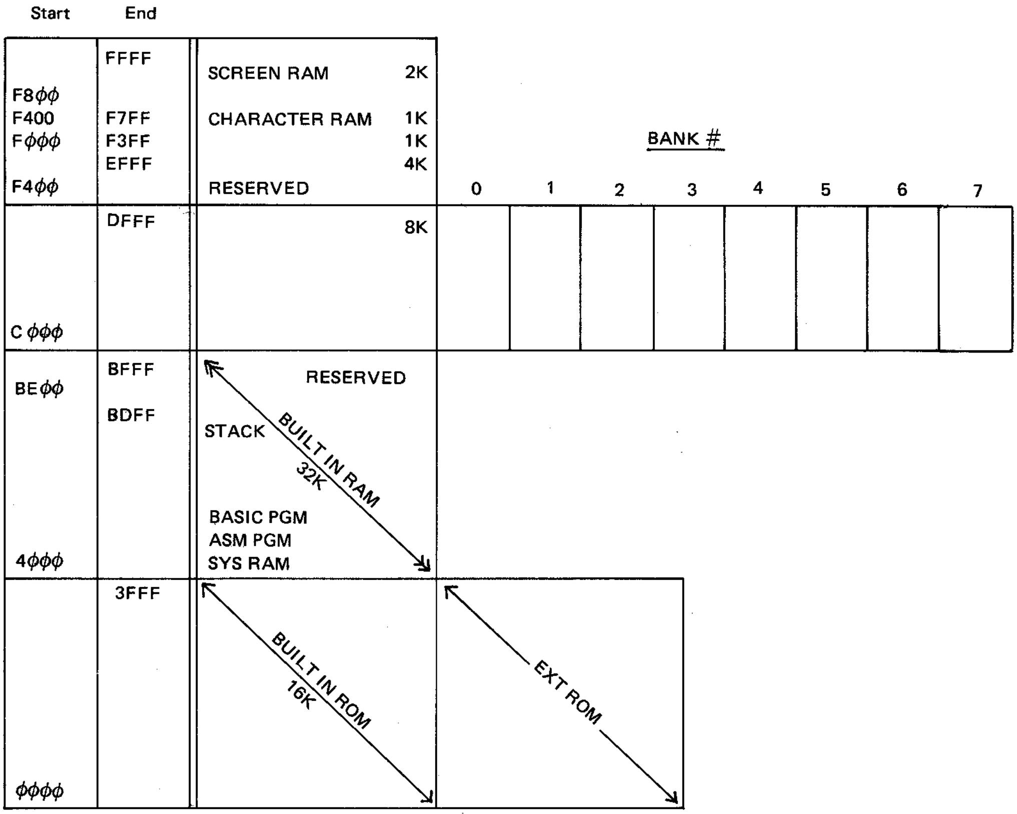
Key :

Letter	Color
C	Cyan
G	Green
B	Blue
Blk	Black

◆ :——Cursor control

** Colors above the numeric keys indicate the corresponding screen colors when SCREEN (X) is executed.

A.6 MEMORY MAP



ClibPDF - www.fastio.com

Memory Map of COMX 35

Memory Location	Description
0000 – 3FFF (hexadecimal)	Build-in system ROM Contains (1) BASIC INTERPRETER (2) TV and keyboard operating system (3) Cassette operating system
4000 – 43FF	System parameter storage area.
4400 – BDFF	BASIC 'PROGRAM' and 'DATA' storage area. Top of system stack located at BDFF.
BE00 – BFFF	Reserved for disk operating system
C000 – DFFF	8 banks of 8K bytes —Used for system expansion —Program must select the desired bank before making read/write operation to this area.
E000 – EFFF	Reserved —Used for inter-bank communication —For application that cannot be fitted into one 8K byte bank.
F000 – F3FF	Reserved
F400 – F7FF	Character definition RAM area. —This area holds the dot pattern of the 128 characters. —The character can be changed using the SHAPE command in BASIC. —Each character uses 9 bytes in PAL system. —Each character uses 8 bytes in NTSC system. —This area can only be accessed by turning on the "character memory access" mode of the CDP 1869 IC.
F800 – FFFF	Screen page memory —Each byte holds the character code of one display character position on the screen. —Before scrolling, F800 represents the left-top character position on the screen. F801 is the one next (to the right) to it and so on. FBBF represents the bottom-right position on the screen. —Hardware scrolling is accomplished by changing the home address register inside the CDP 1869 IC. —This area is write-only and can only be accessed during the non-display period. The program may wait for non-display period and directly write to this area. e.g. B1 $ . . . check EF1 and wait for non-display STR R7 . . . write to this area.

ClibPDF - www.fastio.com