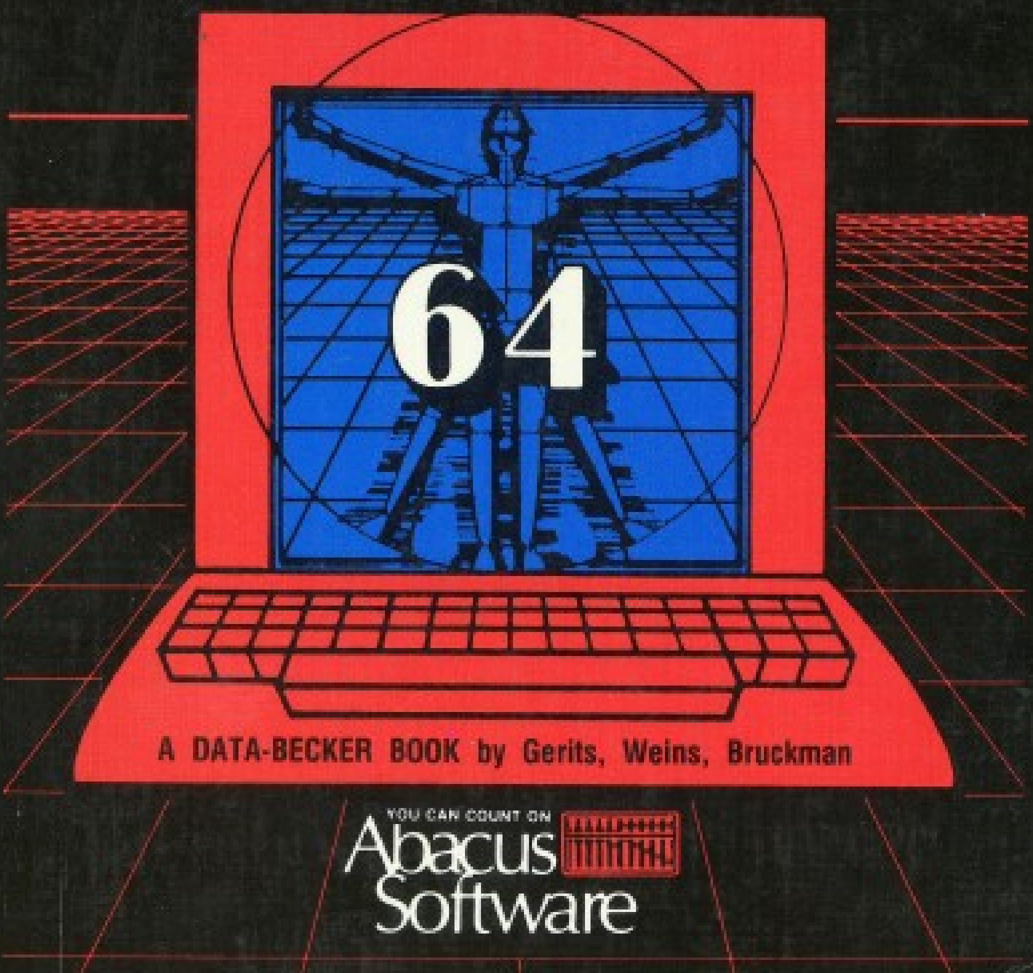
PRINTER BOOK FOR THE COMMODORE-64 AND VIC-20



A DATA-BECKER BOOK by Gerits, Weins, Bruckman

YOU CAN COUNT ON
Abacus Software

PRINTER BOOK

for the Commodore 64 (and Vic-20)

By: R. Bruckmann

K. Gerits

T. Wiens

A DATA BECKER BOOK

Published by:

You Can Count On

Abacus Software

All commands, technical instructions and programs contained in this book have been carefully worked on by the authors, l.e., written and run under careful supervision. However, mistakes in printing are not totally impossible; beyond that, ABACUS Software can neither guarantee nor be held legally responsible for the reader's not adhering to the text which follows, or for faulty instructions received. The authors will always appreciate receiving notice of subseauent mistakes.

First Printing, April 1985
Printed in U.S.A. Translated by Gene Traas
Edited by Jim D'Haem
Programs edited by Russ Taber

ISBN 0-916439-08-9

TABLE OF CONTENTS

1.0 INTRODUCTION..1

2.0 INTERFACES..3
2.1 THE C-64 SERIAL BUS.................................4
2.1.1 FUNDAMENTALS......................................4
2.1.2 ADDRESSES...6
2.1.3 DATA TRANSFER.....................................6
2.2 THE CENTRONICS INTERFACE...........................12
2.3 THE RS-232 INTERFACE...............................18

3.0 SECONDARY ADDRESSES................................25
3.1 VIC-1525/GP-100VIC/MPS-801.........................27
3.2 VIC-1526...29
3.3 THIRD PARTY PRINTER INTERFACES.....................33

4.0 CONNECTING A TYPEWRITER TO THE USER PORT...........35

5.0 PROGRAM COLLECTION FOR VIC-20 AND C-64.............41
5.1 SHORT APPLICATION PROGRAMS FOR EVERYDAY USE.........42
5.1.1 DEVICE NOT PRESENT?..............................42
5.1.2 FORMATTING NUMERICAL AND ALPHANUMERICAL DATA......45
5.1.3 FORMATTED PROGRAM LISTINGS.......................54
5.2 WORD PROCESSING....................................65
5.2.1 MINITEXT - A SIMPLE TEXT EDITOR..................65
5.2.2 WHAT ABOUT POSTERS?..............................83
5.3 HARDCOPY...93
5.3.1 TEXT HARDCOY.....................................94
5.3.2 GRAPHIC HARDCOPY................................100
5.4 GRAPHICS-WITH AND WITHOUT SINGLE-PIN CONTROL.......118
5.4.1 GRAPHICS WITHOUT SINGLE-PIN CONTROL..............119
5.4.2 GRAPHICS WITH SINGLE-PIN CONTROL.................126

5.4.3 THREE-DIMENSIONAL HIRES GRAPHICS....................138

6.0 PRINTER PRINCIPLES....................................147
6.1 THE DAISY-WHEEL PRINTER...............................149
6.2 THE PARALLEL MATRIX PRINTER...........................151
6.3 THE SERIAL MATRIX PRINTER.............................153

7.0 THE OPERATING SYSTEM OF THE MPS-801...................157
7.1 THE 8039 MICROPROCESSOR...............................158
7.2 BLOCK DIAGRAM OF THE MPS-801..........................162
7.3 MPS-801 ROM LISTING...................................165

8.0 THE VIC-1520 PRINTER-PLOTTER..........................239
8.1 USEFULNESS AND OPERATION..............................239
8.2 SECONDARY ADDRESSES ON THE VIC-1520...................248
8.3 CONSTRUCTING FIGURES OF ALL KINDS.....................269
8.3.1 SQUARE..269
8.3.2 RECTANGLE...270
8.3.3 PARALLELOGRAM.......................................276
8.3.4 TRIANGLE..278
8.3.5 ARCS..287
8.3.6 TRIANGLE 2..294
8.4 REPRESENTATIONS OF FUNCTIONS..........................297
8.5 PECULIARITIES OF THE 1520.............................309
8.6 STATISTICS ON THE 1520 PLOTTER........................316
8.7 PLOTTER UTILITY PROGRAM...............................323

1.0 INTRODUCTION

When you saw this book for the first time, you probably wondered to yourself: "How in the world can they write so much material about such a trouble-free device as a printer?"

You can see from just looking at this book, though, that it is quite a large one. You ask again, "Why?"

Let us ask you a question: Would you ever think of writing a book entitled 'The Big Hand Mixer Handbook'? Of course you would dismiss such an idea immediately at first, but it is unbelievable what you might need to know to operate a hand mixer. For example, whipping cream is something you wouldn't need this handbook for; but what do you do when you try kneading bread dough in the mixer, and the mixer jams? Of course you don't know--the Handbook would probably tell you. [Until the book is written, we'll tell you the solution: Simply set the beaters turning in the opposite direction].

You see, our book was written with this in mind. It covers almost everything, from connectors to operating systems, with plenty of material in between for all interested parties. Of interest to most people, though, will be our collection of program listings, which will make using a printer somewhat easier.

We have tried to make this book as comprehensive as possible for the owners of popular printers (and their compatibles) built for the C-64.

Chapter 8 is devoted exclusively to the VIC-1520 printer-plotter. Although the remaining chapters have a good deal which applies to you 1520 owners, we wanted to be as thorough as possible; we didn't want you to feel left out.

All that remains is for us to offer the best to you, in the hopes that you will reap rich rewards from the information to follow.

The Authors

2 0 INTERFACES

There is hardly a subject in home computing that attracts technicians as much as interfaces. These will allow the tech hobbyist to convey information to and from the computer in any number of ways.

Interfaces are necessary to printing. However many capabilities your printer might have, the odds are very good that it is not an "original" device (not ready to plug directly into your computer). To complicate matters even more, most printers within a brand name can differ greatly in connections. Perhaps you have found the printer which will fulfill your needs, yet you aren't sure whether it will connect to your computer.

The salesperson says to you,"But your computer has an X interface, and the printer has a Y connector. Everyone knows that you can't combine the two."

Is it really as clear as that?

We have attempted in this chapter to explain the different interfaces available to you. In the end you can make your own judgement regarding the compatibility of the different interfaces.

Naturally, we'll begin as so many others do when talking about the C-64; with the serial bus.

2.1 The C-64 Serial Bus

The C-64 serial bus is not the same as the standard port of the same name, agreed upon by U.S. manufacturers in 1974.

The 64's has a smaller logic than the "big" serial port, although its small number of lines used are just as complex. See for yourself.

2.1.1 Fundamentals

The serial port has six lines, as opposed to the parallel port's 24 lines. Let's look at them individually:

1 **SRQ (Service ReQuest).** This line allows the computer to convey any new data or tasks to the printer, and lets the printer demand new material if its previous assignment is completed.

This function is not used by Commodore.

2 **GND**

3 **ATN (ATteNtion).** Whenever the computer transmits a command to the printer, this line is activated. Rather than simply send a command directly, ATN is used first to let the printer know that a command will be coming. Once the command is executed, this line can be "uncoupled" from the bus to let it control another device if so desired.

4 CLK (CLocK). Since the serial port transfers data one bit at a time (rather than in bytes), the CLK line sends a time impulse with every bit sent, which checks the validity of the data lines.

5 DATA. This is the data line, which transmits the data byte with the least significant bit. Along with that, this line has other tasks to perform later (as we shall soon see).

6 RES (RESet). A low-level gauge in this line arranges all connected devices in a certain sequence, and checks to see whether a device is switched on.

The example below will serve to illustrate this line's function. Take these two BASIC lines:

```
10 OPEN 1,4
20 PRINT #1,"A"
```

All this program should do is print the letter A to a printer with a device number of 4. Line 20 triggers the bus into action. The OPEN command alone doesn't set the printer into action; this is also the case with the disk drive, if you do not use a filename when accessing programs.

The next section deals with exactly what is addressed when we run our two line of BASIC from above.

2.1.2 Addresses

Please refer to the diagrams on pages 8, 9 and 10.

1: ATN will be low for characters which follow a command.

2: This will turn the data line going to the device low within 1 millisecond. The computer signals when a byte is ready with CLK=high.

3: As long as the device is not ready to take in data, the DATA line will read low; otherwise, DATA=high.

3-4: Now the device number travels over the DATA line, and sends a bit with every high impulse from the CLK line.

5: Within that same one millisecond, the printer returns the DATA line to a low setting. This serves to turn off reception, until more data is sent by the computer.

We now have the DATA line performing three tasks, i.e.:

1. Transmitting data
2. Receiving data
3. Closing off data

2.1.3 Data transfer

6: The data transfer begins here, as long as the sender has set CLK high. This shows that a byte is ready for transfer.

7: The receiving device acknowledges its readiness by setting DATA high.

8-9: A data bit now travels over the DATA line with every impulse given by CLK.

9: Receipt of the entire byte is conveyed to the sender by DATA=low. Our letter A in line 20 has just been printed (and is all that is left of this entire procedure).

11-12: The carriage return, or CHR$(13), is transmitted in the same way.

Now the CLK line has an additional task:

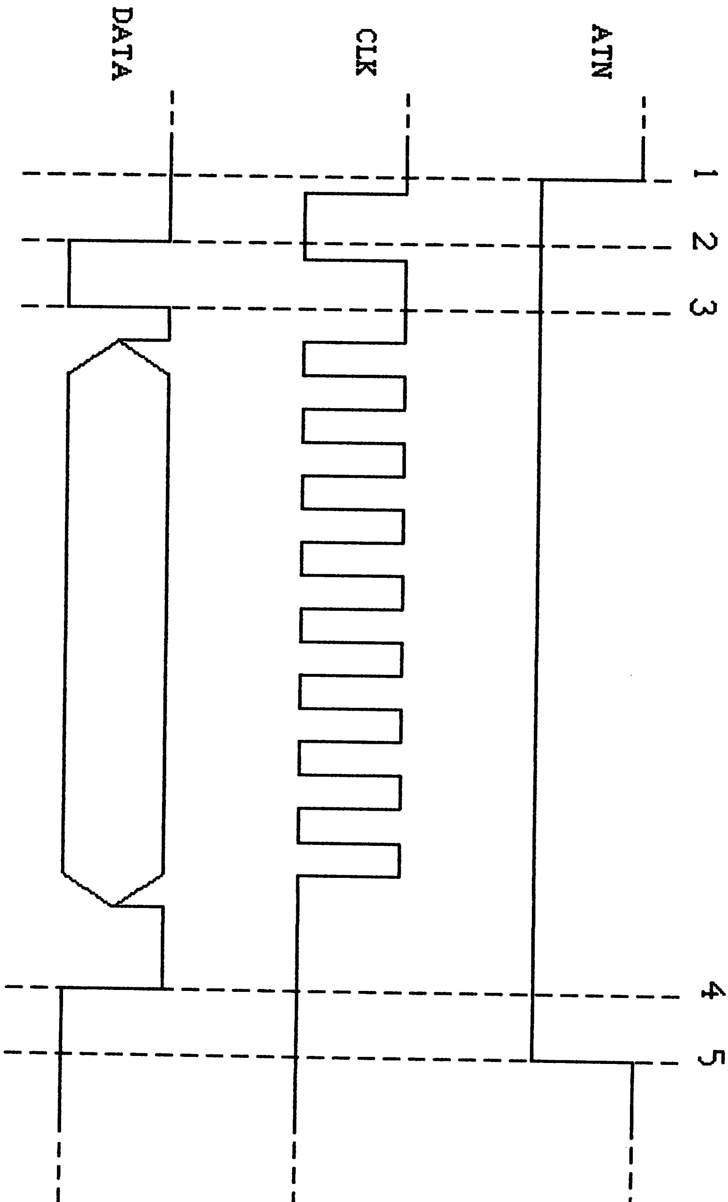
At the same time that the last byte of transfered data is registered, the receiver is told to begin its assignment. This function is entrusted to CLK.

13-14: This arrangement is similar to the last two illustrations, with the addition of the receiver setting the DATA line to high; the sender receives this change within 0.2 milliseconds after the data tranfer was initialized by CLK=low. Now the CLK remains high, which obviously means that the byte previously sent was the last byte.

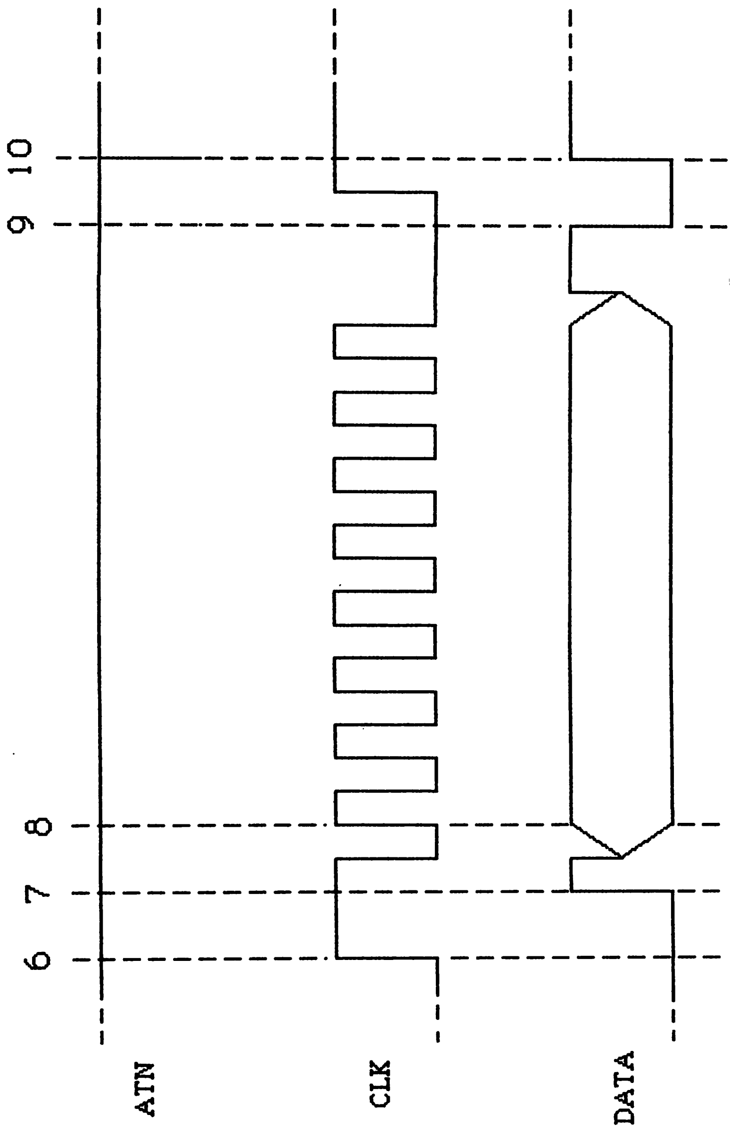
This condition will cause the receiver to quit with DATA=low.

15: As soon as DATA is again high, the sender carries over an empty byte (chr$(0)--this is point 16 in the illustration), after which the bus stands in a state of rest.

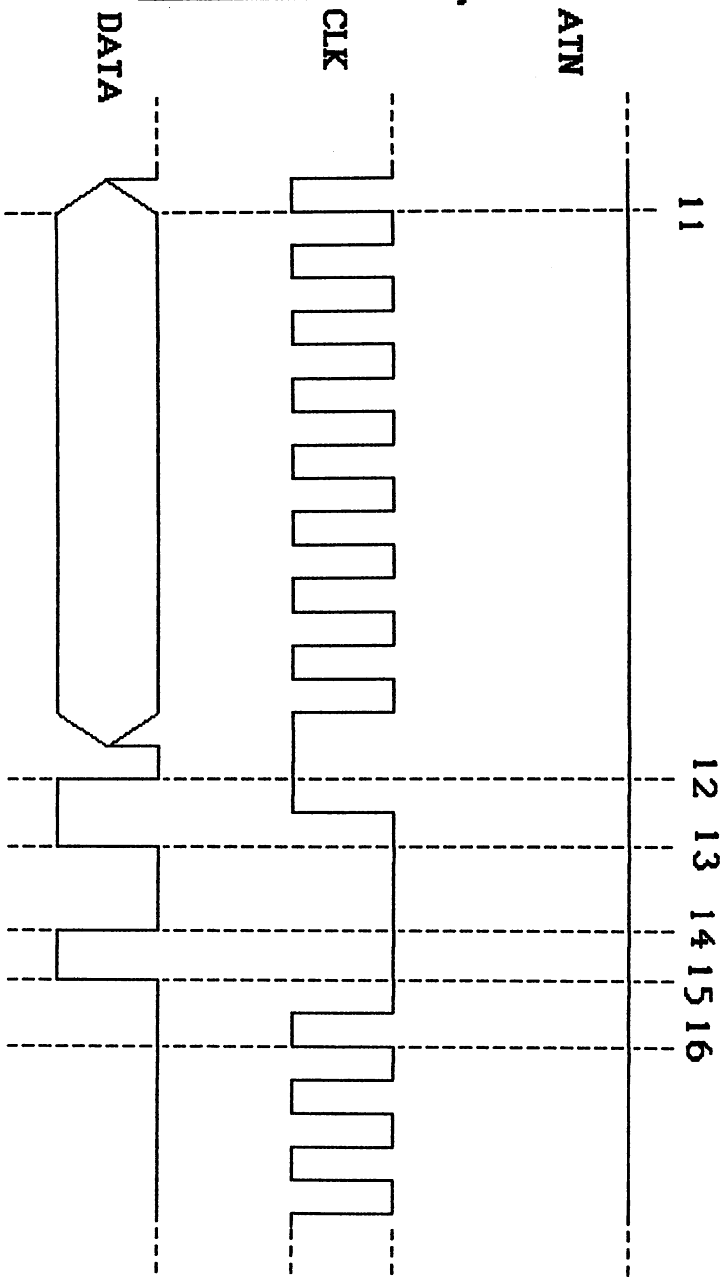
Addressing



Data Transfer



Last Data Byte



The deaddressing follows along the lines of points 1-5, whereby the data byte shows a value of 63 ($3F). This is the so-called UNLISTEN command, which means that all devices receiving data up until now are uncoupled from the bus, so that no more data will be transferred.

As you can see, this system works beautifully. The most annoying disadvantage to a serial port (as opposed to a parallel port) is the slow speed of data transfer (the parallel port does in 1 millisecond what takes the serial port 64 milliseconds), which results in lots of waiting time.

2.2 The CENTRONICS Interface

One of the most widely-used printer interfaces is the one proposed by Centronics. This method of data transfer has been accepted by most printer manufacturers, and most printers are compatible with this product.

You have a great advantage if you have a printer with this sort of interface (or an economical variant of it). The only problem remaining is connecting the printer to the C-64.

The most economical method of making this connection is presented in detail in ABACUS Software's <u>Tips and Tricks for the Commodore 64</u>. You'll need a connecting cable to rig the printer to the user port, and a little driver software. With the help of these writings on interfaces (which will help you find your way around the user port a bit), it will not be at all difficult for you to assemble such a cable.

Somewhat more expensive (but clearly more convenient) is an interface which converts the serial C-64 serial port into a readable form for the printer. This is usually a small box which simply connects between the two devices. This solution has a catch to it, however: Many units of this kind limit you to "standard" alphanumeric characters. The many graphic and control characters in the C-64's character set are not usually acknowledged by the printer using this box. Otherwise this interface is quite useful, especially when you consider that no driver software of any kind is required. From BASIC the printer can be addressed as a normal Commodore device. For any character code conversions that might be necessary, please see Section 3.3.

A good all-around solution (as the ease of use is well worth the expense) is a full graphic interface manufactured by CARDCO and others. This sort of interface (mentioned in Section 3.3) gives you great control over non-commodore printers. First, you just connect the printer up to the serial port as if it were a standard Commodore printer, which means that you can easily operate it from BASIC. Secondly, the printer will easily produce those control characters and graphics with the help of this interface. Aside from that, the device is compatible with the VIC-1525 printer, so that most software used with the latter will run the EPSON with no real trouble.

Now, however, we come to the subject at hand, i.e., the Centronics interface.

As a rule, it has a 36-pin Cinch connector, from which about half the lines actually carry a signal. If you wish to limit yourself to data transfer and nothing more, 11 of those lines would be sufficient. The others serve as signals to certain conditions within the printer, and are not available on all the different brands of connecting jack.

Let's begin with the pin layout on the Centronics case.

1 STROBE. This line is high when the printer is in a state of rest. A low impulse takes less than one microsecond to signal the printer that data is on the way.

2-9 DATA 0-7. The printer receivies bytes over these lines.

10 ACKNLG. The printer acknowledges receipt of the data byte over this line, using a low impulse of approximately 12 microseconds. The sender can then transmit the next byte over the data lines (but still no strobe to send it back!).

This line is absolutely necessary to error-free control of the data flow. The so-called BUSY line can serve the same purpose, i.e., the 11 signals mentioned above are the important ones, so most interfaces do not include ACKNLG.

11 BUSY. This line is high when the printer is not ready to receive a new data byte. This line has different ways of operating; the printer can be stopped when the paper has run out, or if it is OFFLINE, etc.

If possible, the data sender (the computer) should first send the data with STROBE when BUSY=low.

12 PE. This line gives a high signal to the printer when the paper has run out.

16 GND. This is the ground wire for all signals.

18 +5V. This can provide any external interfaces with voltage. This current is not available on all printers. The necessary information should be in your printer manual.

31 INIT. This line gives a low impulse of less than 50 microseconds, bringing the printer to the starting position, as if you had just switched the printer on.

32 ERROR. Using a low impulse, this line informs the printer whether there is no paper left, or whether the printer is OFFLINE, or if any errors appear.

The combination of the PE and ERROR signals can help you interpret any cause for the BUSY mode shutting off, if you can get these signals on your computer. These lines are rarely used.

36 SLCT IN. A low signal on this line makes it possible for the printer to function initially. It's conceivable that you could produce an artificial low signal with the help of a switch rigged to the inside of the printer.

If your printer does not stir on initialization, then you should disregard the above.

The unused lines either have no function whatsoever, or they perform tasks specific to the printer. Check your printer documentation.

It is possible to run a printer with lines 1-9, 11 and 16 alone. The remaining lines may be used by some manufacturers, but no universal agreements have been made.

The following timing diagram should illustrate the way that the interface operates.

1: When BUSY=low, a byte can be sent over the data lines. After a minimum of 0.5 microseconds, the

2: STROBE will be released, which has a duration of at least 1 microsecond.

The arrival of the character byte will change BUSY to high.

3: After STROBE is taken off, a time period of

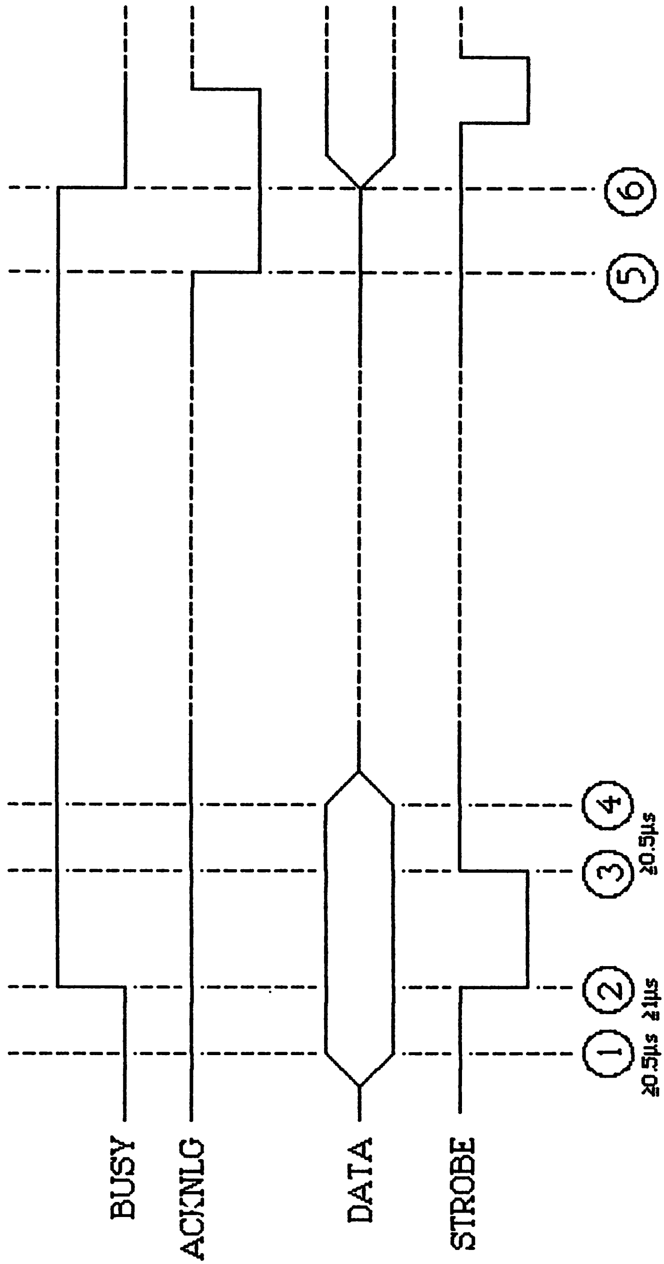
4: less than 0.5 microseconds elapses. Now the data lines will change their condition.

5: The time can vary greatly from here on in, whereas in the previous steps, specifics could be given. This depends on what the printer has to do in the meantime, i.e., bringing up characters from the buffer, or whether the most recently handled character was at the end of a line.

6: Now new data can be sent over the data lines.

When BUSY=low, the STROBE can be released, and the entire procedure started anew.

Timing of a Centronics Interface



2.3 The RS-232C Interface

Here we'll look at yet another method of the logic of interfacing two devices. The RS-232C type of interface is preferred because of its ability to transfer data and its versatility; its physical construction and logic circuits are the reason for this.

An RS-232C plug is considered bi-directional, i.e., data can travel in both directions. In most cases, the data transfer only goes in one direction (to the printer), but when definite control lines are involved (such as on the Centronics interface), the printer sends information in the opposite direction (back to the computer).

The reason that we are presenting so much detail on this interface is that, as we've already mentioned, this interface is very widely used; another reason is that the Commodore RS-232C Cartridge supports the C-64 operating system with no problem; finally, in the sections to follow on setting parameters, we are using this interface as the standard, to help take some of the mystery out of addressing for the layman.

The problem lies not only in the proper choice of data transfer parameters (we'll explain parameters later), but in the fact that sometimes two devices will operate using this interface using a simple "one-on-one" connection--more often than not, they won't work. We'll show you how to solve this problem, and how the problem is caused, do not be discouraged if you aren't sure whether the problem is in the electronic equipment, or in the program logic.

The following explanations deal with the VIC-1011 RS232-Cartridge from Commodore only.

The connector is based on a 25-pin connector (best known as a DB-25-subminiature). You can purchase a matching connecting cable at any electronics store or computer store.

The following pin arrangement will help you in setting up the connector. It is very important that you note which data direction line goes in which direction, so that you avoid two signal outputs connected to one another. This may sound like a silly thing to say, but you'll feel much sillier when the connection has been soldered, and the connection doesn't work for that reason.

Don't worry, though, in our pin layout for the RS-232 Cartridge, you'll find that only the relevant signals will be used. You needn't use the pins left unmentioned.

PIN 2: TXD Transmitted Data (signal output). This line serially transfers the user data.

PIN 3: RXD Received Data (signal input). The connected device sends back that data has been received. As a rule, this line is not usually reserved in printers.

PIN 4: RTS Request to Send (signal output). This line is activated (>3V) when the interface is awaiting data. This is not always the case in printer operations; in 3-wire operations, this signal is always on.

PIN 5: CTS Clear to Send (signal input). The C-64 can send data ONLY when this line from the connected device is on (>3V).

This line must be connected up in printer operations since it dictates data flow; otherwise, characters can be lost during data transfer.

PIN 6 DSR Data Set Ready (signal input). This line sort of "takes attendance" for the C-64's operating system, i.e., it checks to see whether a device is connected to the interface, and whether it is switched on.

Any operation, be it sending or receiving, will only occur if this line detects a voltage over 3 volts.

PIN 7 GND Ground. Applies to all lines.

PIN 8 DCD Data Carrier Detected (signal input). This signal has roughly the same meaning as 6. You would be best off connecting it with pin 20.

PIN 20 DTR Data Terminal Ready (signal output). This line is the opposite of 6. This signal is activated with the first OPEN command containing device number 2.

The transfer parameters are determined in the computer in the OPEN command, and in the printer by the dip switches. It is of first priority to understand what these parameters are.

1. Baud rate

This determines the transfer speed in bits per second (a rough estimate) and is the most important parameter of them all---if the baud rates do not match, the system doesn't work at all.

2. Number of Databits

This parameter determines how many bits a byte will contain. Normal values when connected with a printer are 7 and 8.

3. Number of Stopbits

As a rule, this is only 1 or 2. You can try playing with this number, but watch out for compatibility problems.

4. Parity

You have a choice of no parity, even, or odd parity.
You can choose no parity, because it saves trouble with 8 data bits, and if no transfer errors are expected.

In case you don't know what parity means:

A data byte consists of 0-bits and 1-bits. To check whether a byte has been transferred ungarbled, an additional bit is added, i.e., the so-called parity bit. This should produce either an odd or even number in a series of 1-bits consistently, dependent on the parameter set.

The receiving end counts the 1-bits, and compares the number to the parity setting. If they don't match, the matter is handled as a transfer error--the printer normally ignores the bad characters that come through.

Should you have all the parameters set correctly, and the devices are properly connected, and the thing still won't run, you can try a little trick on the parity; just try no parity, then look at the table below for setting parameters from BASIC.

To simplify setting parameters from the computer end, we have assembled the most important variables and placed them in the table below; the unnecessary parameters have been left out. If you are interested in seeing those not listed, see ABACUS SOFTWARE's <u>The Anatomy of the Commodore 64</u>.

1.	Baud rate:	Variable br=
	50	1
	110	3
	150	5
	300	6
	600	7
	1200	8
	2400	10
2.	Data bits:	Variable db=
	7	32
	8	0
3.	Stop bits:	Variable sb=
	1	0
	2	128

4. Handshake:	Variable hs=
no	0
yes	1

We have yet to discuss the variable hs. After choosing your parameters, the OPEN command would look like this:

```
OPEN 1,2,0, CHR$(br+db+sb)+CHR$(hs)
```

Do the following to construct your cable:

Solder a cable wire to pin 7 of both connectors (printer-side and computer-side).

Connect (again, both ends) pins 4-5 and 6-8-20 to one another. This will first check for handshaking, and look for operational readiness at both ends.

Check your printer handbook to see which pin is for data input (usually 2 or 3). Connect that pin, and solder the other end of that lead to pin 2 at the computer side.

If all parameters are correct (and parity is disabled), the cable should be ready to go. Connect the printer and computer, turn them on, and try an OPEN and PRINT command.

Continuous printing should help us determine whether some characters are being skipped (you'll often find this at the beginning of a line). We haven't given you any method yet for shutting off the printer when the data flow stops.

Now you'll have to figure out which line of the printer signals this condition. Set variable hs (in the above OPEN) to equal 1. Open the bridge 4-5 on the computer side, and connect pin 5 (from the computer side) first to bridge 4-5, then with bridge 6-8-20. One of these two cases has to give you complete data in every line.

If hs=0, the handshake lines are not interpreted, and the computer sends data anyway. This procedure only works well with an extremely low baud rate, since the printer halts for input after each character. In order to use a slow data flow, you should use a printer without any sort of input buffer.

The only item remaining is to let you know how to switch on lower-case lettering, and call up upper-case again. First you'll need some technical background--see the explanation of secondary address 1 in section 3.3.

You see, it is not so difficult to find your way through the maze of parameters and lines, as long as you have some idea of what you're doing.

3.0 SECONDARY ADDRESSES

There is hardly a subject as important as the secondary address. This "grey area" can make life fairly difficult when it comes to software compatibility with printers.

Don't be misled by the prefix "secondary"; this doesn't mean that the address takes second priority. This indication is only used in serial bus terminology to distinguish it from the primary address (the device address itself). The secondary address gives the manner in which the device will be used.

By the end of this chapter, the beginning programmer should have a better idea of how these numbers work.

Remember, printers do not all respond the same way to the same secondary address numbers. The biggest difference lies in that, for example, one printer may use an address number of 7 as a switch to halt data, while another printer uses the same number to do the opposite. This, of course, means that software designed for the former will crash when used with the latter. There is a way around this; it is a matter of changing PRINT statements around a bit, so that problems can be ironed out.

Bear this tip in mind: Set up your printer output routine as a single subroutine, where the specific PRINT statements can be found. Then, when different printers are brought in to play, you only have to change a minimum of variables. It would be wise to do the same with the OPEN commands. As a rule, this will be all that you may have to change.

If the program logic doesn't do this, and you still have several print statements, proceed as follows:

Define the logical file number, device address, and secondary address at the beginning of the program, and fill in the numbers in the OPEN and PRINT statments with these definitions. You'll kill two birds with one stone by doing this: You will be able to easily change both the secondary address, and you can quickly alter the logical file number for printers that require an extra linefeed (use a number >127) or don't require additional linefeeds (<128).

Here is an example:

```
10  LF=1:DA=4:SA=7
:
100 OPEN LF,DA,SA
:
1000 PRINT #LF,..............
```

Now it is just a matter of changing the values in line 10 when the necessity arises, without having to look up all the OPEN and PRINT statements.

We will now give you the secondary address functions for the most popular printers. We'll call particular attention to any peculiarities in the printer functions.

3.1 VIC-1525/ GP-100VIC/ MPS-801

These printers only recognize two secondary addresses which help them determine which character set to use. Like the 64, these printers also have two basic representations.

Secondary address 0 (or no secondary address)

```
10 OPEN 1,4,0 [or  10 OPEN 1,4]
20 PRINT#1,"abcABC"
```

This will cause the printer to print lower-case letters in upper-case, and upper-case letters as graphic characters.

You can temporarily change this mode for the length of a program line by inserting the character string for "cursor-down" before the text:

```
30 PRINT#1,CHR$(17)"abcABC"
```

Upper/lower-case will print properly in this line only.

Secondary address 7

```
10 OPEN 1,4,7
20 PRINT#1,"abcABC"
```

Upper- and lower-case are printed consistently. This can be changed temporarily (cf. secondary address 0) using the character string for "cursor-up" before the text:

```
30 PRINT#1,CHR$(145)"abcABC"
```

Now the lower-case appears as upper-case, and the upper-case letters are printed as graphics.

Please remember that the changes made by CHR$(17) or CHR$(145) are in effect for the length of the current PRINT statement ONLY.

Naturally, you can mix techniques within a line, either like this:

```
10  OPEN 1,4
20  PRINT#1,"abc"chr$(17)"ABC"
```

or like this:

```
10  OPEN 1,4:OPEN 2,4,7
20 PRINT#1,"abc":PRINT#2,"ABC"
```

Both versions give the same result. These versions are only compatible on Commodore printers.

3.2 VIC-1526

Much has been written about this printer; few devices have caused so much confusion in the past. Here is a summary of its history:

The printers put on the market before 12/83 were nothing like what the manual claimed--rather, they ran like the VIC-1525s. The only regrettable difference was that the single-needle graphics supposedly set up by CHR$(8) were not possible.

However, the following measures were installed in the 1526: After opening the device, pin 16 of the IC in socket U4D is grounded. See the manual for applications.

Run a self-test to find out which is your printer. The first line of the printout should have the version number of the printer's operating system. This should be less than 7.

1526 operation features the following secondary addresses:

Secondary address 0 (or no secondary address).

Similar to address 0 on the printers in Section 3.1. Temporary changes can be made using CHR$(17) and CHR$(145).

Secondary address 1

This gives out the formatted data previously determined by the formatting command given by secondary address 2.

The choice of character set follows (given by secondary address 7 or 8). Also, line changes can be made here.

Bear in mind that you can separate one field of data from another by using CHR$(29). See your printer manual for other examples.

Secondary address 2

This address transfers the format command. In other words, it informs the printer to print out hardcopy in a certain sequence. This is useful for numbers in which the decimal places should be lined up vertically. See your manual.

Secondary address 3

Here you can determine how many lines per page to print before the printer automatically performs six linefeeds. This is practical for endless printer paper; the linefeeds skip over the perforation, leaving three lines on top of the next page, and three lines on the bottom of the previous page. To figure out how to reach this value, consider that a maximum of 66 lines fill a sheet of 11" paper. Subtract 6 (three for each border), and send the following commands:

```
10    OPEN 1,4,3
20    PRINT#1,CHR$(66)
30    CLOSE 1
```

Please note that this measure alone does not perform this function; page subdivision must first be activated using

```
10    OPEN 1,4
```

```
10   OPEN 1,4
:
500  PRINT#1,CHR$(147)CHR$(141).
```

Secondary address 4

This is nothing more than a plaything; it switches on the printer's diagnostic mode:

```
10   OPEN 1,4,4:PRINT#1:CLOSE 1
```

The printer will now list any program errors.

Secondary address 5

This address allows you to define any custom characters that will fit into an 8X8 matrix.

In order to draw any large single-point graphics, please see Section 5.3.2.

Secondary address 6

This lets you adjust the distance between two lines in increments of 1/216"; the best range lies between 1 and 127.

Please note that the distance between points is three steps. In order to produce an unbroken line, you'll need the following commands:

```
10   OPEN 1,4,6:PRINT#1,CHR$(24):CLOSE 1
```

Secondary address 7

This switches on upper/lower-case mode. Use the following to do so:

```
10    OPEN 1,4,7:PRINT#1:CLOSE 1
```

Note: No data is transmitted. This mode can be switched off using secondary address 0 or 1.

Secondary address 8

Performs the opposite of secondary address 7.

```
10    OPEN 1,4,8:PRINT#1:CLOSE 1
```

Returns the printer to upper-case/graphic mode.

Secondary address 9

```
10    OPEN 1,4,9:PRINT#1:CLOSE 1
```

Switches off the diagnostic mode initialized by secondary address 4.

Secondary address 10

```
10    OPEN 1,4,10:PRINT#1:CLOSE 1
```

Initializes the printer (just as if you'd turned the printer off and on again). This gives the option of being able to switch the printer off in mid-run, to make any program changes and additions, without the using the OFF/ON switch.

3.3 Third Party Printer Interfaces (Non-Commodore Printers)

This section has some additional material due to the diversity of hardware.

Basically, third party interfaces try to make your printer identical to the VIC-1525.

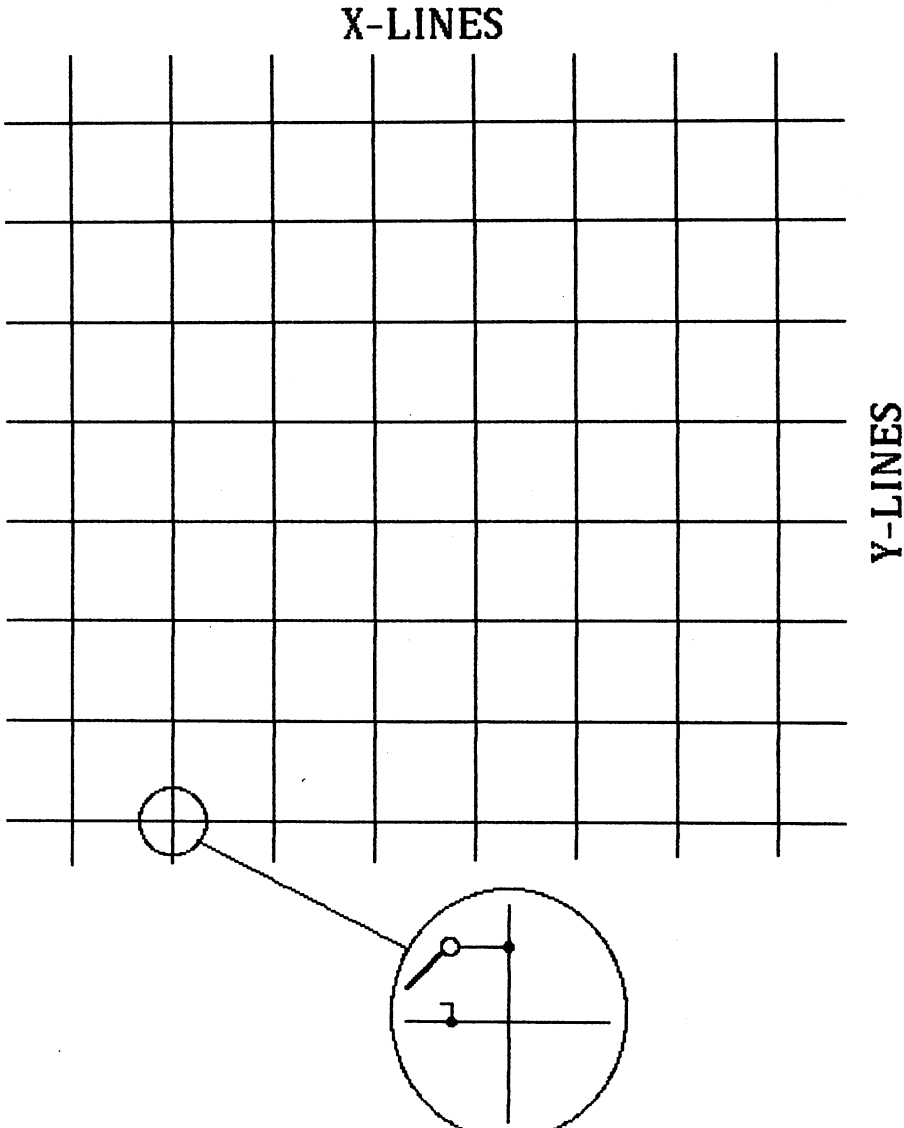
All printers on the market (with the exception of Commodore) have a character distribution known as ASCII (American Standard Code for Information Interchange). In this internationally acknowledged system, you can count on the characters and control characters being in the same order. So, for example, CHR$(65) will always equal "A", and CHR$(97) will be "a". Commodore's character codes (known as Commodore ASCII) are quite different; CHR$(65)="a" and CHR$(193)="A".

Third party printer interfaces change Commodore ASCII into true ASCII to enable Commodore computers to communciate with many other printers.

Each third party printer interface uses secondary addresses to preform a variety of different options. One such interface the "CARD?+G" includes commands for "Special listing modes", "upper/lower case", "upper case/graphics", "graphics mode" and the ability to "LOCK" the interface into the specified mode.

Check your manual for complete details about your particular interface.

Common Keyboard Matrix



4.0 CONNECTING A TYPEWRITER TO THE USER PORT

This chapter refers to connecting a recent-model electronic typewriter, just to keep the record straight.

You will find that building an interface for this purpose is really not that difficult. The hard part is figuring out how the typewriter works, and how to connect it. Despite the problems, maybe you have one of these typewriters at home, and perhaps you've had an itch toward "computerizing" it.

First you have to determine whether the wiring in the typewriter is suited for this connection; it seems that the simpler and cheaper the action (the material value of the action can hardly be worth more than $3.00), the more choosy the action can be. Now, if by reading the instructions or looking inside the machine, you have found that it has a microprocessor controlling it, then we can go ahead with the procedure. Don't worry, you won't have to reprogram the typewriter; we'll use the set-up on hand, and check the keyboard's X-Y matrix (see below), then take the connection from there.

If the typewriter fits the criteria listed above, what next? For the next step, all you'll need is an ohmmeter. Open the typewriter carefully, so that you can get at the keyboard contacts.

Before we go any further, an explanation of the keyboard matrix might be in order. This matrix is designed along the lines of the illustration on the previous page. The number of lines will vary from machine to machine, but it doesn't matter.

You can see that every point of intersection on the matrix lies even with a typewriter key. The microprocessor runs a voltage across every X-line. If no keys are pressed, of course nothing happens. If, however, a key is pressed, the voltage travels across the corresponding Y-line. This tells the processor exactly which key was pressed, because of the varying resistance between X-lines and Y-lines.

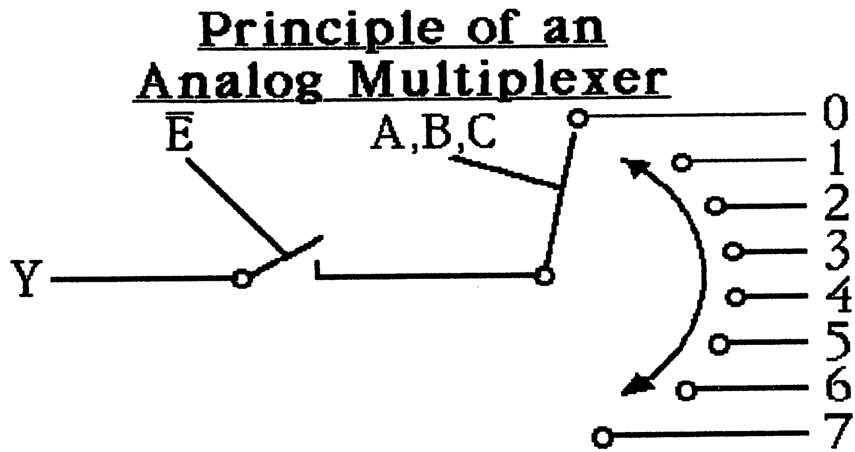
Make sure that this voltage isn't over 5 volts, since our interface cannot handle higher voltages; most processor-equipped machines will not go over this voltage, so there won't be any real problem.

Our task is now to simulate the keyboard switches using our interface. The matrix arrangement is ideal for doing this with a minimum of soldering. Imagine if you will, you can control all 50 keys using TWO wires. To do this, we'll tap the X- and Y-lines, but that isn't important at the moment.

Now for the hard part. You must figure out where the X- and Y-lines lie. As you remember from the illustration, the lines connect underneath keys. You'll have to check these connections using an ohmmeter; this means that you'll have to check out the resistance of each key, since there is no guarantee that the voltages run sequentially from key to key.

It will probably take you about two hours to do this. You should end up with an arrangement similar to the illustration on the previous page. Don't forget to make a record of the combinations; this will make things much easier when we start programming the computer to match the typewriter.

Now for our controller, which is based of two analog multiplexers. The illustration below shows its inner workings.



Look at this picture, and compare it to the complete circuit at the end of the chapter. You'll see what we mean. We can connect one of eight X-lines with one of eight Y-lines, controlled by the user port. The typewriter programming is juggled around a bit, and the key which is under the intersecting lines, strikes.

Since this arrangement of lines is not a permanent one, we use the Enable Input on the multiplexer, which will switch on only for the duration of the keypress.

The system works in such a way that three bits sent over the user port choose the X-line, three more bits choose the Y-line, and the remaining two bits serve as the analog switch for the shift key and whatever other key you wish. Now that the connector is built and rigged up, test it out using the following program:

```
10 PA=56576:PB=56577
20 DA=56578:DB=56579
30 POKE DA,PEEK (DA) OR 4:REM DATA DIRECTION REGISTER
100 INPUT "X-LINE";X
110 INPUT "Y-L1NE";Y
120 INPUT "SHIFT 0=NO, 1=YES";SH
130 CH=X+8*Y+64*SH:PRINT CH
140 GOSUB 200:GOTO 100
200 POKE PB,CH
210 POKE PA,PEEK (PA) AND 251
220 FOR DL=0 TO 100:NEXT DL
230 POKE PA,PEEK (PA) OR 4
240 RETURN
```

Write down the combinations between the printed characters and the X- Y- lines and SHIFT key; the program will give you these numbers onscreen at the same time the printing occurs.

You can now create a translation table with the help of the program below. This will make possible easy use of your new printer/typewriter.

```
10 DIM CH(255)
20 GET A$:IF A$=""THEN 20
30 PRINT A$" CORRESPONDS TO ";
40 INPUT CH
50 CH(ASC(A$))=CH
60 GOTO 20
```

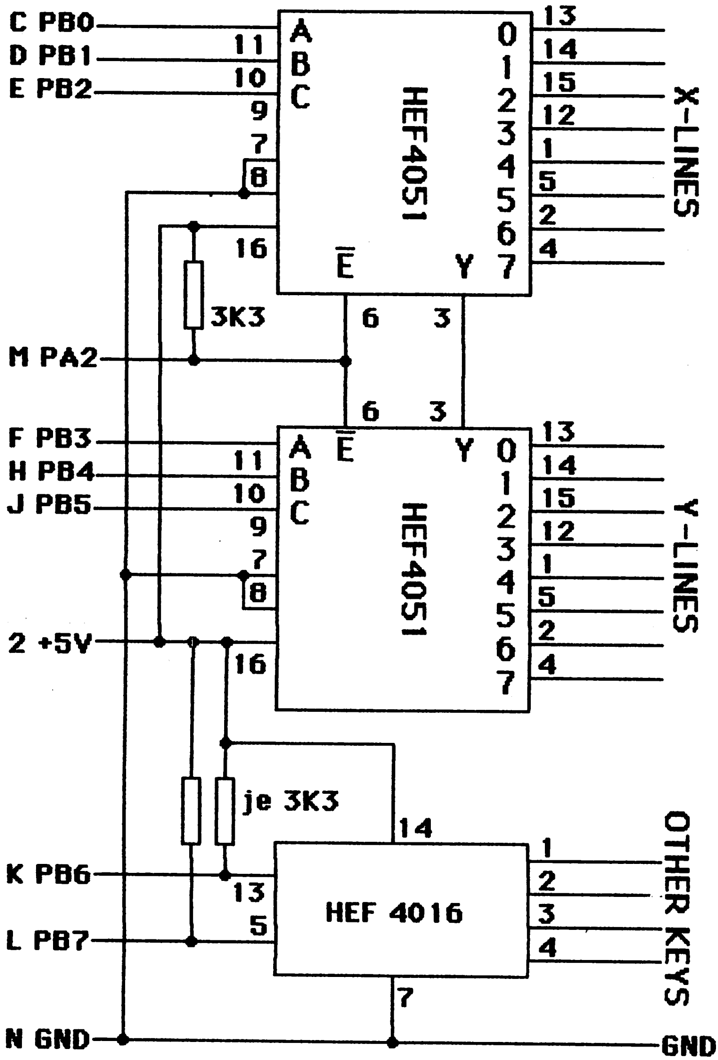
When you're done, stop the program (STOP key) and save the table on diskette or cassette for later use.

Printing out text is as simple as this: Replace lines 100-140 of the first program using the following routine:

```
100 FOR L=1 TO LEN(CH$)
110 CH=CH(ASC(MID$(CH$,L,1)))
120 GOSUB200:NEXT L:RETURN
```

Some prerequisites: Your text must be contained in CH$, and you start the routine with GOSUB 100. Lines 10-50 must be run the first time to give variables, but can them be clipped off for speed.

So, that's it. Good luck in your efforts.


C PB0
D PB1
E PB2
M PA2
F PB3
H PB4
J PB5
2 +5V
K PB6
L PB7
N GND
HEF4051
HEF4051
HEF 4016
3K3
je 3K3
X-LINES
Y-LINES
OTHER KEYS
GND

5.0 PROGRAM COLLECTION FOR VIC-20 AND C-64

In this chapter we'll present some programs which have some good ideas for you to apply to your own printer programming.

These programs cover a wide range of subjects. Unfortunately, it's impossible to produce programs covering every general subject, for every printer on the market. If we tried to do this, this book would be the size of the Manhattan Yellow Pages!

The single-point graphics program naturally doesn't offer any support to the daisy-wheel printer owner. However, we have included some small graphic programs that will give these readers some surprising results.

Tips have been included at all important points, to help you adapt these programs to your printer. If your printer wasn't mentioned in these pages, and your machine uses different control codes from the ones we've listed, you may find codes that perform the same tasks in your printer documentation.

Regardless of whether or not all these programs run on your printer at the moment, you'll still be able to apply most of these suggestions to your own programming.

5.1 Short Applications Programs for Everyday Use

You'll find three utility programs in this section which should be of interest to all printer owners. First we have set up a short routine which prevents a program break if the printer or disk drive has not been switched on. You can attach or merge this routine to any BASIC program, to prevent the annoying ?DEVICE NOT PRESENT error.

The second routine is somewhat more extensive; it is used for formatting numerical and alphnumeric data.

Finally, there is another formatting program. This one, however, is expressly for formatting program listings, arranging the lists for easy reading. Spaces are inserted between commands, FOR/NEXT loops are indented, and IF-THENs are broken down. Thus, a formerly unreadable listing is made comprehensible.

5.1.1 Device Not Present?

Commodore BASIC has no built-in provision for avoiding a DEVICE NOT PRESENT error, which can cause a program crash.

SYNTAX ERRORs are easily cured, as are DIVISION BY ZERO errors. Unfortunately, it's not that simple with printer or disk access. Below is a brief subprogram that will inform you of the device being off; just switch it on, and get on with your work. There is a neat feature to this program; for those of you who are having problems protecting your programs from listing, there is a small protection device in the program.

```
110 REM ******************************
120 REM **                          **
130 REM **        PRINTER           **
140 REM **                          **
150 REM ******************************
160 POKE768,1
170 OPEN 1,4
180 PRINT#1,"";
190 CLOSE 1
200 POKE 768,139
210 IF ST<>-128 THEN 230
215 PRINT" PRINTER IS NOT TURNED ON"
220 RETURN
230 REM *******************************
240 REM **                           **
250 REM **            DISK           **
260 REM **                           **
270 REM *******************************
280 POKE768,61
290 OPEN1,8,15,"I"
300 CLOSE1
310 POKE 768,139
320 IF ST<>-128 THEN 340
330 PRINT"DISK DRIVE NOT TURNED ON"
340 RETURN
```

Note that the disk routine isn't very different from the printer routine. Both routines can be switched on with GOSUB. You can change the messages to suit your own tastes.

Lines 150 and 280 shut off the STOP key vector. These pokes also turn off the error message output, and go directly to the RETURN commands. If you use RUN-STOP/RESTORE, or if the program breaks off, the computer will hang up.

The only way to get control back is to do a "cold start" (switch off, and back on).

5.1.2 Formatting Numerical and Alphanumeric Data

Every once in a while you have a need for a neatly formatted printout of numbers and/or characters. Yet you often end up frustrated, because Commodore BASIC doesn't have a FORMAT or PRINT USING command.

Other computers have great commands like PRINT USING and other utility commands that tend to leave the Commodore user feeling a bit left behind. So, we search the Commodore handbooks, and find that the solution just isn't there.

One fine day, however, the first 1526 printer appeared on the market, and there was great rejoicing in the Commodore community. It was said that this printer had the ability to print formatted listings easily. It was not capable of graphics, but that didn't seem to matter, with the other neat features.

The rejoicing didn't last long. One of the items not mentioned in the manual was the printer's ability to completely lock up the serial port, which can be inconvenient when you want information from the disk drive.

This fact was brought to the attention of the manufacturer, who quickly announced a replacement ROM for the 1526.

The manufacture of this ROM was much slower in coming than the announcement-- and those who live by the saying, "Good things come to those who wait" were terribly disappointed when the ROM did come out.

In order to cure the old problems, the developers of the operating system software had to utilize some other memory locations which were not available to the old IC socket. So they took the scalpel to the socket, and created some memory space. Part of that space was the area where the formatted output existed.

So, now we're back where we started. There are five routes to take to solve our problem. You can:

1) Buy a new computer with PRINT USING capability;
2) Buy a software development system such as MASTER-64;
3) Write your own routine;
4) Use the routine printed below; or
5) Forget the whole business, and do without the formatted lists.

Number 4 seems the most economical method. Before we go on with this routine, however, let's see what output control Commodore BASIC does offer us for printing.

Printers are used for reproducing characters. Printing a letter is a fairly simple form of output, as far as controlling borders and justification.

We've all seen printouts of calculations. Text and numbers must be straightened out, i.e., the text left-justified, and the numbers right-justified. One possibility exists in your computer's BASIC language: The function TAB. Unfortunately, this function is for the screen, not the printer. This should be obvious when you try this example:

```
PRINT"HELLO"TAB(10)"HOW'S IT GOING?"
```

will be arranged on the screen like this:

```
HELLO     HOW'S IT GOING?
```

but the printout will look like this:

```
HELLO          HOW'S IT GOING?
```

How is this possible? Simple; the operating system converts TAB(X) into SPC(X) when the output is to any device other than the screen.

TAB is a position counted from the left border, while SPC is counted over from the last character position. The starting position of the second text, then, depends on the length of the first text.

There is a possibility that can be used on some Commodore printers:

It's possible to get a pseudo-TAB by returning the printhead to the left border without a linefeed, then counting off the TAB and printing the second part of the line. You can accomplish this on some Commodore printers using CHR$(141). Here is a concrete example to try:

```
10 OPEN 1,4
20 PRINT#1,"EXTENSION CORD"CHR$(141)SPC(40)"$1.00"
```

The text is printed on the left edge, then the CHR$(141) sends the printhead back to column 1 without advancing the paper; next, the SPC(40) moves the printhead to column 40, and the price is printed there.

Some printers, such as the 1525 don't run this program as is; replace the CHR$(141) with the 1525's special control code {CHR$(16)}.

This control code allows you to put your printhead in one of 80 positions. The print position must be sent in ASCII codes to the printer. Here is an example:

```
10 OPEN 1,4
20 PRINT#1,"STRAWBERRY JELLY"CHR(16)"40$1.98"
```

This example tells the printer to put the price on column 40. The print position is determined by the 40 preceding the dollar sign.

You have to manually figure out the distance from the left border needed by SPC(x) to print the text in that particular place on the paper; this isn't always the best method.

Let's take the first example (the extension cord listing):

Let's suppose that the amount contained in the price string was "$11.45" instead of the original price. The amount would always start at the same left position; however, this would shift the decimal point one place to the right. The result is interesting, but it certainly doesn't help your financial advancement.

We want is some method of formatting the text so that the decimal points are lined up vertically. There is a solution in program logic, which first looks for a number within a string, checks for a decimal point, adjusts the SPC-parameter, and prints the price in the proper spot.

This calculation allows you to have several fields per line. In our program, we start with the stock number, then quantity, description of the article, unit price, and we end with the total price.

You can have this luxury by typing in the following subroutine, and adjust it to your own needs.

```
48000 REM
48001 REM****************** FORMATTING *************
48002 REM
48020 IFFZ%=0THENFZ%=1:FL%=LEN(FO$):FP$=""
48040 FI%=LEN(FI$):FB$="":FC%=1
48050 IFMID$(FO$,FZ%,1)=" "THENFZ%=FZ%+1:FB$=FB$+" ":
      GOTO48050
48060 FM$=MID$(FO$,FZ%,1)
48080 FV%=0:FH%=0:FA%=0:FB%=0:FF%=0
48100 IFMID$(FO$,FZ%,1)="."THENFZ%=FZ%+1:FF%=2:GOTO48180
48120 IFFZ%>FL%THENFA%=FA%+1:FZ%=FZ%+1:GOTO48240
48140 IFMID$(FO$,FZ%,1)=" "THEN48240
48160 FA%=FA%+1:FZ%=FZ%+1:GOTO48100
48180 IFMID$(FO$,FZ%,1)=" "ORFZ%>FL%THEN48240
48200 FB%=FB%+1:FZ%=FZ%+1
48220 GOTO48180
48240 IFMID$(FI$,FC%,1)="."THENFC%=FC%+1:FF%=FF%+4:GOTO48320
48260 FV%=FV%+1:FC%=FC%+1
48280 IFFC%>FI%THENGOTO48380
48300 GOTO 48240
48320 IFFC%>FI%THEN48380
48340 FH%=FH%+1:FC%=FC%+1
48360 GOTO48320
48380 IFFM$="A"THEN48780
48400 L2=FA%-FV%:IFL2<0THENFF%=FF%+1:GOTO48720
```

```
48420 IFL2=0THEN48460
48440 FORL1=1TOL2:FB$=FB$+" ":NEXTL1
48460 IFFV%=0THEN48520
48480 L2=FV%:FORL1=1TOL2
48500 FB$=FB$+MID$(FI$,L1,1):NEXTL1
48520 IFFF%=0THEN48920
48540 FF%=0:FB$=FB$+".":L2=FV%+2
48560 IFFB%=0THEN48920
48580 IFFB%=FH%THENL3=FB%:GOTO48620
48600 L3=-FH%*(FB%>FH%)-FB%*(FH%>FB%)-1
48620 FORL1=L2TOL2+L3
48640 FB$=FB$+MID$(FI$,L1,1):NEXTL1
48660 IFFH%>=FB%THEN48910
48680 L2=FB%-FH%:FORL1=1TOL2
48700 FB$=FB$+"0":NEXTL1:GOTO48910
48720 L2=FA%+FB%:IF(FF%AND2)THENFF%=1:L2=L2+1
48740 FORL1=1TOL2:FB$=FB$+"*"
48760 NEXTL1:GOTO48910
48780 IF(FF%AND4)THENFV%=FV%+FH%+1
48800 L2=FV%:FORL1=1TOL2
48820 IFMID$(FI$,L1,1)<>" "THENL3=L1:L1=L2
48840 NEXTL1
48860 L2=-FV%*(FA%>FV%)-FA%*(FV%>FA%)
48870 IFL2=0THENL2=FV%
48880 FORL1=L3TOL3+L2
48900 FB$=FB$+MID$(FI$,L1,1):NEXTL1
48902 IFFV%>=FA%THEN48910
48904 L3=FA%-L2:FORL1=1TOL3
48906 FB$=FB$+" ":NEXTL1
48910 IFFZ%>FL%THENFZ%=0:GOTO48960
48920 L2=FZ%:L3=FL%:FORL1=L2TOL3
48930 IFMID$(FO$,L1,1)<>" "THENFZ%=L1:L1=L3:GOTO48950
48940 FB$=FB$+" "
```

```
48950 NEXTL1
48960 FP$=FP$+FB$:FF%=(FF%AND1)
48970 RETURN
```

Since the program contains many variables that greatly influence the operation of the routine, here is a list of variables and explanations.

FO$ must contain the line format desired.
FI$ must be provided with user data.
FB$ contains the processed partial string, while
FP$ holds the complete print string.
FZ% is the pointer for the term processed in FO$.
FF% signals an overrun of the digits to the left of the decimal point (FF%<>0).
FA% contains the number of digits allowed to the left of the decimal point for each processed field in FO$.
FB% is as above, but to the right of the decimal point.
FV% is the number of whole number places in FI$.
FH% contains the number of decimal places in FI$.

Program commentary:

48020-48080	Initially sets some variables to 0.
48100-48160	Numbers left-of-decimal are processed in arrays FO$ to FA%.
48180-48220	As above, but for right-of-decimal (FB%).
48240-48300	Number of whole digits in FI$ to FV%.
48320-48360	Number of decimal places if FI$ to FH%.
48400-48440	Here the arrangement of numerical fields begins. If FI$ has fewer digits left-of-decimal-point than contained in FO$, spaces fill in the difference.

48460-48500	Valid numbers left-of-decimal follow FB$.
48560-48640	Valid decimal places follow FB$.
48660-48700	If FI$ contain fewer decimal places than are in FO$, the area is filled in with zeroes.
48720-48760	If FI$ contains more digits left-of-decimal than FO$, the entire field will be filled with "*", to allow you to immediately see the error in the output. In this case, you must carefully measure the size of the array. This condition signals the program when FF%<>0.
48780-48840	Preparations for alphanumeric fields begin here. First spaces from FI$ will be printed, separating each field for easy readability.
48860-48900	Characters from FI$ are transferred according to FO$ - FB$. Reamining characters from FI$ are cut off, while leftover characters from FO$ are filled in with spaces.
48910-48950	FB$ will be filled in with spaces, according to the distance to the next field contained in FO$.
48960-48970	FB$ added to FP$, and memory is put back into order.

Determine how many fields you want per line, and whether these fields contain strings or numbers.

Put this information into FO$. Our sample will look like this:

```
FO$="99  999  aaaaaaaaa   999.99   9999.99"
```

Numerical fields are characterized with "9", strings with "a". Alphanumeric fields will be left-justified, but the numerical fields will be right-justified, in accordance with their position in FO$.

All that remains is for you to input your data to FI$; the routine will run for as long as is required.

The line prints out at the end of the routine.

Your program addition could look like this:

```
10 OPEN1,4
20 FO$="99  999  AAAAAAAAAA   999.99   9999.99"
30 FI$=RIGHT$(STR$(PO),2):GOSUB48000:
   REM POSITION NUMBER
40 FI$=STR$(AN):GOSUB48000:REM QUANTITY
50 FI$=AR$:GOSUB48000:REM ITEM DESCRIPTION
60 FI$=STR$(EP):GOSUB48000:REM UNIT PRICE
70 FI$=STR$(AN*EP):GOSUB48000:REM TOTAL PRICE
80 PRINT#1,CHR$(16);FP$
```

Note that the routine jumps in five times, or as often as needed (as determined in FO$). The configuration in FO$ is just an example; you can redo it to fit your own needs.

One small tip: Occasionally you'll want to use a smaller string for a line. You'll have to reset FZ% to 0, and set up the new string for FO$.

5.1.3 Formatted Program Listings

Everyone knows that in order to spare every byte possible, REMs and spaces aren't included in programs. If, after several weeks, you want to go in and make some changes, deciphering the listing can be difficult. Also, when you have a normal listing at hand, finding program structure can be bothersome, especially since BASIC isn't a very structured language.

Now, our solution to the problem is not the final one; the best method involves advance planning on your part, writing your programs in an easily-read form. However, with the help of the next program, you'll be able to print out programs from diskette or cassette which will have spaces inserted between command words, and FOR-NEXT loops interlocked. Besides this, a new line will be started with the appearance of each colon.

In order to understand the logic of this program, you'll need some details on program memory and how the BASIC interpreter works in the VIC-20 and C-64.

To start, let's have a look at how a BASIC program resides in memory. If you have a machine language monitor, the following steps will help you better understand what we're talking about.

First load the monitor. Then type in the following program line in BASIC:

```
10    PRINT"HELLO"
```

Next, let's look at the memory locations where this line is placed. 64 users will find this in $0800-$080F:

```
0800 00 0E 08 0A 00 99 22 48
0808 41 4C 4C 4F 22 00 00 00
```

These addresses vary on the VIC-20; depending on the amount of memory expansion plugged in (if any), the addresses can start at $0400, $1000 or $1200. The VIC-20 owners will have to remember this change--the memory stays constant on the C-64.

Back to the memory dump above. What do these numbers mean? Location $0800 must contain a zero--this marks the beginning of the BASIC program memory. Any other value will give you a ?SYNTAX ERROR every time you type RUN.

The contents of addresses $0801 and $0802 refer to the next program line. This means that the next possible line would be located at address 080E, which is used as a cue for GOTOs, GOSUBs, and FOR/NEXT loops. Another term for this line reference is **line-link**.

The next two memory locations contain the actual line number in high-byte/ low-byte format. Our example gives $0A00, which is 10 in decimal.

Now we come to the specific command in the line--the PRINT statement. Problem is, it doesn't LOOK like a PRINT statement; all that exists is the value 99. Let's change that value to see whether it functions as PRINT, since we have the monitor running. Change the 99 to a 97, exit to BASIC, and list the BASIC program.

Now we have a very different listing: The PRINT statement is gone, and the listing now looks like this:

```
10  POKE"HELLO"
```

Statements and commands within a program are converted into single-byte code, with every command having its own individual code. A little testing can help you determine which code is which. You'll note that these codes (also known as **tokens**) have values higher than $7F.

That means that the most significant bit of an address is set by tokens. Thus, finding tokens in a disassembled listing is quite simple; most significant bits are otherwise only found outside of quotes, so commands usually precede strings. If such a character sets a most-significant bit in a program line, and if the number of quotation marks add up to zero, or an even number, then the bit is handled as a token.

In addition, the tokens exist in the computer's ROM as a set of BASIC commands. The computer uses this command set for program input, in order to check command syntax. Each line is checked to see whether such a command exists. If that is th case, the command is immediately converted into its respective token. We'll be using this table of commands later in our program.

Now we come to the end of this long preface; on with the program listing. The listing below will run as is on the C-64. In order to run it on a VIC-20 (minimum 3k expansion required!), change line 370 as it suggests there.

```
100 DIMBT$(75)
110 LM$="        ":NF$="  " :BL$=".    "
120 FORI=1TO8:HZ$=HZ$+LM$:NEXT
130 HZ$=HZ$+"SEITE "
140 F1$=CHR$(145)+LM$
150 SN=1
160 PRINTCHR$(147)
170 PRINT:PRINT:PRINT"        LIST - PROGRAM"
180 PRINT:PRINT
190 INPUT"     PROGRAM NAME";NA$
200 NP$=NA$+",P,R"+CHR$(34)
210 PRINT:PRINT"         OUTPUT TO "
220 PRINT"    "CHR$(18)"P"CHR$(146)"RINTER OR "CHR$(18)"D"
    CHR$(146)"ISK";
230 INPUT GA$
240 IFGA$=""THENGA=3:GOTO310
250 IFGA$="D"THENGA=8:NL$=NA$+",S,W"+CHR$(34):GOTO310
260 IFGA$="P"THENGA=4
270 PRINT:INPUT"     LINES PER PAGE ";ZN:
280 ZN=(ZN=0)*-66+ZN
290 PRINT:INPUT"     TITLE      ";UE$
300 HZ$=RIGHT$(HZ$,LEN(HZ$)-LEN(UE$)):UE$=UE$+HZ$
310 PRINT:INPUT"     LEFT BORDER        ";L
320 PRINT:PRINT:    FORI=0TOL:LM$=LM$+" ":NEXT
330 OPEN15,8,15
340 OPEN1,8,4,NP$:GOSUB950
350 I=0
360 FORA=41118TO41372:REM VIC20 FOR A= 49310 TO 49564
370 X=PEEK(A)
380 IFX>127THEN410
390 BT$(I)=BT$(I)+CHR$(X)
400 GOTO440
410 X=XAND127
```

```
420 BT$(I)=BT$(I)+CHR$(X):PRINTF1$;BT$(I);LM$
430 I=I+1
440 NEXT :PRINT:PRINT
450 BT$(11)=BT$(11)+"   "
460 BT$(39)=BT$(39)+" "
470 REM *********  READ PROGRAM   ***********
480 PRINTCHR$(147)
490 IFGA<8THENOPEN4,GA :GOTO510
500 OPEN4,GA,3,NA$+".LST,S,W"
510 IFGA<>3THENPRINT"OUTPUT FOLLOWS ON DEVICE#";GA
520 PRINT#4:GOSUB1040
530 GOSUB980:GOSUB980    : REM PRG-START SKIP
540 REM *********    MAIN LOOP    ***********
550 GOSUB980:GOSUB980    : REM SKIP LINK ADDRESS
560 REM
570 GOSUB980 :LB=A        : REM LINE  LOW - BYTE
580 GOSUB980 :HB=A        : REM LINE HIGH - BYTE
590 ZN$=STR$(HB*256+LB)
600 LN$=LEFT$(LM$,LEN(LM$)-LEN(ZN$))+ZN$+NF$:LF=0:PF=0:NN=0
610 REM *********   LINE LOOP ***********
620 IFLEN(LN$)<70THEN680
630 PRINT#4,LN$;:IFNOT QFTHENPRINT#4
640 IFQFTHENPRINT#4,CHR$(34)
650 LN$=LM$+NF$:IFQFTHENLN$=LN$+CHR$(34)
660 NN=0
670 GOSUB1010
680 GOSUB980              : REM GET FIRST BYTE
690 IFA=0THENPRINT#4, LN$:LN$="":QF=0:RF=0:GOSUB1010:GOTO550
700 IF NOT RF THEN 720    : REM READ THE REMAINDER
710 LN$=LN$+CHR$(A):GOTO620
720 IFA=32ANDNOTQFTHEN 620
730 IF A<>44 THEN760
740 IF A=44 AND NN=0THEN760
```

```
750 IFNF$<>"    "THENNF$=LEFT$(NF$,LEN(NF$)-4)
760 IFA=34THENQF=NOTQF
770 IFA=58ANDNOTQFTHENPRINT#4,LN$;":":LN$=LM$+NF$:NN=0:GOSUB
    1010:GOTO620
780 IFA=61ANDNOTQFTHENLF=NOTLF
790 IFA>127ANDNOTQFTHENGOSUB850:GOTO620
800 LN$=LN$+CHR$(A):IFNOTQFTHENGOSUB820
810 GOTO620
820 IFRIGHT$(LN$,4)=";  "+CHR$(34)+CHR$(34)THENLN$=LEFT$(LN$,
    LEN(LN$)-4)
830 IFRIGHT$(LN$,4)="+  "+CHR$(34)+CHR$(34)THENLN$=LEFT$(LN$,
    LEN(LN$)-4)
840 RETURN
850 IFRIGHT$(LN$,1)<>" "THENLN$=LN$+" "
860 IFA=128+39THENPRINT#4,LN$:LN$=LM$+NF$ :GOSUB1010
    : REM THEN
870 IFA=129THENNF$=NF$+BL$ : REM BL$= 4 SPACES FOR FOR NEXT
880 IFA=130THENNN=1: GOSUB930
890 IFA=143THENRF=NOTRF     : REM REM FLAG ON
900 IFA=153THENPF=NOTPF     : REM PRINT FLAG ON
910 IFA=178THENLF=NOTLF     : REM LET   FLAG ON
920 LN$=LN$+BT$(A-128)+" ":RETURN
930 IFNF$<>"    "THENNF$=LEFT$(NF$,LEN(NF$)-4):LN$=LEFT$(LN$,
    LEN(LN$)-4)
940 RETURN
950 INPUT#15,A$,B$,C$,D$
960 IFA$<>"00"THENPRINTB$:CLOSE1:CLOSE4:CLOSE15:END
970 RETURN
980 GET#1,A$:A=ASC(A$+CHR$(0))
990 IF ST <>0  THENCLOSE1:CLOSE4:CLOSE15:END
1000 RETURN
1010 ZZ=ZZ+1:IFZZ=ZN THEN1030
1020 RETURN
```

```
1030 FORI=1TO72-ZN:PRINT#4:NEXT
1040 PRINT#4,LM$;"  ";UE$;SN
1050 PRINT#4:PRINT#4:ZZ=3
1060 SN=SN+1
1070 RETURN
```

In order to understand the functions of this program, some knowledge of BASIC program design is necessary. Thus, somewhat more time will be spent than previously in program description. This commentary will be helpful when you decide to make your own modifications to the program. One conceivable change would be to modify reverse characters to print out instead as CHR$(X), so that any printer can print out a program listing without concern for special characters.

Lines 100-150 initialize some very important variables. The array BT$ will later contain the individual BASIC commands, while LM$ defines a left border.

After the screen clears, you are asked for the program name, and the output device. Although the normal output device is the printer, you have the option of using the disk drive as the output device. If the latter is chosen, the program is prepared as a listing for word processing, into which you can add detailed commentary. In this case, the filename will be the same as the previously given program name.

If you respond to the output device prompt with <RETURN>, the output will default to the screen.

Bypassing the disk drive will cause lines 270-350 to jump in. This line asks for lines per printed page and title text. The title will appear on every printed page, along with the page number. If you respond to the lines-per-page input with <RETURN>, the system will default to 66 lines per page.

Line 370 asks for a left margin: Your best choice would be between 6 and 10 -- a lower number might start the left border over the perforations on the paper. If you wish, you can also respond here with <RETURN> only, but no additional room for pinfeed will be provided.

Next, the error channel and program data are executed (line 950), where any error messages will eventually show up. The most frequently-encountered error message is FILE NOT FOUND.

Once this routine is done, the FOR-NEXT loop in lines 360-460 PEEKs into BASIC ROM for the abovementioned list of command words. This list has a small peculiarity to it: All commands are listed one after the other; in order to keep the commands separated, the eighth data bit of the last character of each command word is stored in this table. This eighth bit is recognized in line 380, and dealt with in line 410; the bit is deleted, and the command displayed on the screen.

The variable index simultaneously raises, and PEEKs for the next command in memory.

Finally, the commands IF and THEN are each made 5 characters long, i.e., they are filled in with spaces.

Now, the program gets data from the diskette. The subroutine in lines 980-1000 perform this task; with every call of the subroutine, a character is loaded from the diskette and stored in variable A for processing.

First, the routine is called twice, but the value in A is not used. The reason for this; the first couple bytes of any program file gives the load address--something which doesn't interest us at the moment.

Now the program goes to the beginning of the main loop at line 540 Here, too, the first two bytes are taken but left unused. These two values are line-link addresses (see above). The next few bytes are much more important, since they contain the line number. These are computed and formatted in lines 590 and 600. The line loop, in which all the bytes belonging to this line are handled, begins in line 840.

First, the previously assembled string (LN$) is checked for length (<70 characters per line). If this amount is overstepped, then the portion given up to that point is printed out, and the subroutine in line 1010 is called. Here the line is checked to see whether it has indeed reached the maximum number of characters, and whether a linefeed is needed.

After this test, a byte is sent to line 710, and the work proper begins. After the bytes are sent to variable A, many things happen at once; we'll give line numbers as we go through the sequence.

First, A is tested to see whether it is equal to zero. If that is the case, the end of the line has been reached, which causes the line number for the print line to increase, and for LN$ to give out the next string. If A is equal to 32, however, a space is given. When a space is found in the program text, the program goes on to the next byte. If a space is found in quotation marks, it is inserted in line string LN$ for later use. The difference will be apparent in variable QF, which we shall explain in a moment.

Next follows a test for a value of 44 (comma). This is important in context of the NEXT command which has several variables separated by commas.

Line 760 will test for the value 34 (quotation marks). If A=34, then QF will be checked with a NOT operation: -1=true, while 0=false. This quickly tests to see whether a character is in quotes or program text.

A routine follows to test for colons (=58). Again, QF is used to see if the colon is in the program text, or if it is in quotation marks. If the colon is part of a program line, LN$ declares an end-of-line, and a new printed line is begun. This breaks up lines with multiple statements, and arranges them for easy readability.

If, the character received is larger than 127 and is in the program text, it is handled as a token; but if this character is in quotes, then it is put into LN$.

The value 167 (see line 860) stands for the token THEN. If the program detects a THEN token, the previous line will be printed, to make the IF-THEN structure more visible.

Line 870 searches for the interpreter code for FOR: When this token is found, the string NF$ is extended to the character sequence in BL$. This procedure prints out FOR-NEXT loops in sequence.

Line 880 seeks the presence of a NEXT command. When this command appears, the FOR- command is added to the characters in line 930, then the set is removed. When the NEXT must involve several variables separated by columns, the NN-flag is set to 1.

Line 890 tests for the presence of a REM statement. If this is the case, the REM-flag (RF) is set, printing out the REM line without checking the remainder of that line for commands.

After executing lines 850-890, the line string LN$ adds the character string for the corresponding interpreter code. Thus, the corresponding array element for the token (128) is addressed. In addition, a space is attached to increase readability, and the program returns to the line loop.

This program looks pretty complicated at first glance. Basically, all it does is perform a LIST command, with some additional functions performed by the program.

5.2 Word Processing

Next to a database, word processing is probably the foremost reason why people buy computers. Since word processing involves printers, we have produced a short program for the purpose called 'Minitext'. Now, this program doesn't compare with a true word processing program such as TEXTOMAT, for example; but this program is adequate for typing up a letter or quick note. It is written in machine language, so that you can see how printer control works in machine code.

The second program is a word processor of another sort: This program allows you to print larger-size characters. The characters are enlarged many times their normal size, making this program useful for making posters and banners.

5.2.1 MINITEXT - A Simple Text Editor

When a computer and printer are combined, chances are that a word processor is involved. Publishing a source listing for an efficient program such as TEXTOMAT would take over 300 pages. Although we haven't tried it, we estimate a corresponding program in the form of DATA statements would take up 40 to 50 pages. Not even the most hard-core hobbyist would enjoy typing in that much material.

So we began searching for a passable program that would require as little effort to type in as possible. We immediately ruled out writing the program in BASIC, because of Commodore's notoriously slow "garbage collection" (string functions), which get in the way of word processing.

Garbage collection removes strings no longer necessary from memory. This procedure can take anywhere from a few seconds to over 30 minutes. During this time, the computer accepts no input from the keyboard, and appears to be locked-up.

The second point against a BASIC text editor is the relative size of such a program. In order to perform the most elementary functions, the program would have to be of considerable size.

There is an alternate way that you may already have tried in printing text. This method will treat text like a normal BASIC program written in PRINT# commands. This is really quite an unsatisfactory method, since you have to give every line a line number, and type in the command "PRINT# LF". Also, you would have to be sure that each line is under a certain length: This in itself can be a nuisance to fluid writing.

On the other hand, this last method means that the screen editor is available to you, and that you have access to the commands for loading and saving the text. So, after much consideration, we chose this method, and ascertained how we could dispose of the disadvantages. The result was a short machine language routine of 508 bytes.

If you look at how text is printed in a PRINT statement, you'll notice that quotation marks must precede the text. Without these quotes, the text after the line number will not be printed.

There is yet another problem: The cursor control keys, the RUN/STOP key, CLR/HOME and the function keys produce reverse graphic characters when in quote mode, which means that editing the program line using your cursor keys is out of the question.

These problems are all solved by the first part of the machine language program. The line numbers are self-generating (similar to the AUTO command found in some BASIC extensions), as are the quotation marks, so you needn't worry about typing either. Quote mode is switched on by a memory cell in zero-page RAM. After the quotation mark appears on the screen, this memory cell is switched off, and the computer "thinks" that it is no longer in quote mode; this means that you again have the cursor keys available for editing.

We still have a problem--the line length. Most of us are from the "hunt-and-peck" school of typing, and tend to keep our eyes on the keyboard, rather than the screen. When we're trying to concentrate on writing, we seldom think of looking at the screen. Therefore, a sound signifying the end-of-line would be a good idea.

Looking again to the zero-page range mentioned above (the first 256 bytes of RAM or so), we find a memory cell which gives the number of characters in a BASIC line. Our little program regularly checks this cell to see how close we are to the end-of-line (using the interrupt, so it tests for the end number about 60 times a second). When the end is approaching, the television speaker will produce a sound. You should press the <RETURN> key at the end of the next word.

Any words extending past the end-of-line will have to be broken off and continued in the next line.

As soon as the <RETURN> key is pressed, a new line number and quotation mark immediately appear.

The second part of the program is responsible for printing out the text. Now, you could do this just by typing LIST, but who likes to write letters that have a line number at every left margin? This program section prints the line without line number or quotes, instead giving out a clean left border, so that the characters do not overstep the perforated section of the paper.

This program works in connection with the BASIC interpreter, for ease of handling. After the program is called with SYS, you have two new commands available: The !GET command to get a specific line; and the !PRINT command to print out the text. The latter can be called with the abbreviation !?.

On to the program listing. If you have a little experience working a machine language monitor, you should find it quite easy to modify this program to your own needs.

```
c000                  *=    $c000
c000       beepflg    =     2
c000       formfeed   =     6
c000       lm         =     10
c000       cr         =     $0d
c000       lo         =     $14
c000       hi         =     $15
c000       txtan      =     $2b
c000       txend      =     $2d
c000       fac        =     $62
c000       text       =     $71
c000       chrget     =     $73
c000       chrgot     =     $79
c000       prtcode    =     $99
c000       getcode    =     $al
c000       filnm      =     $b7
c000       lgfnr      =     $b8
c000       secad      =     $b9
c000       devad      =     $ba
c000       line       =     $fb
c000       incr       =     $fd
c000       pgline     =     $fe
c000       bufferl    =     $101
c000       buffer2    =     $200
c000       vector     =     $0302
c000       vectorl    =     $0308
c000       irqvec     =     $0314
c000       basloop    =     $a474
c000       oldvec     =     $a483
c000       mainloop   =     $a486
c000       goon       =     $a569
c000       weiter     =     $a576
c000       basstrt    =     $a68e
```

```
c000        execold    =    $a7e7
c000        chkcom     =    $aefd
c000        syntax     =    $af08
c000        illquant   =    $b248
c000        getpar     =    $b7eb
c000        intfloat   =    $bc49
c000        floatasc   =    $bddd
c000        sid        =    $d400
c000        clrscrn    =    $e544
c000        irqold     =    $ea31
c000        open       =    $ffc0
c000        close      =    $ffc3
c000        ckout      =    $ffc9
c000        clrch      =    $ffcc
c000        input      =    $ffcf
c000        bsout      =    $ffd2
c000        stop       =    $ffe1

*************PROGRAM BEGINS HERE*****************

c000 a9 41      lda #<testin
c002 a0 c0      ldy #>testin
c004 8d 08 03   sta vectorl
c007 8c 09 03   sty vectorl+1
c00a a9 80              lda #128       ;shift-commodore disable
c00c 8d 91 02           sta $0291
c00f a9 17              lda #$17       ;upper/lower-case on
c011 8d 18 d0           sat $d018
c014 a9 00              lda #0
c016 a8                 tay
c017 91 2b              sta ($2b),y    ;clear BASIC program
c019 c8                 iny
c01a 91 2b              sta $2d
```

```
c01c a5 2b          lda $2b         ;set vectors
c01e 18             clc
c01f 69 02          adc #$02
c021 85 2d          sta $2d
c023 a5 2c          lda $2c
c025 69 00          adc #0
c027 85 2e          sta $2e
c029 20 8e a6       jsr basstrt
c02c 20 44 e5       jsr clrscrn     ;clear screen
c02f a9 80          lda #$80
c031 8d 8a 02       sta 650         ;repeat all keys
c034 a9 64          lda #100        ;first line number
c036 85 fb          sta line
c038 a9 00          lda #0
c03a 85 fc          sta line+1
c03c a9 0a          lda #10         ;line condition
c03e 85 fd          sta incr
c040 60             rts             ;return
c041 20 73 00 testin jsr chrget
c044 c9 21          cmp #"!"
c046 f0 06          beq test2       ;if ! then test further
c048 20 79 00       jsr chrgot
c04b 4c e7 a7       jmp execold     ;otherwise, normal commands
c04e 20 73 00 test2 jsr chrget      ;get next character
c051 c9 a1          cmp #getcode    ;is it !get?
c053 f0 0a          beq get
c055 c9 99          cmp #prtcode    ;is it !print?
c057 f0 03          beq prt
c059 4c 08 af       jmp syntax      ;syntax error, or not?
c05c 4c 41 c1 prt   jmp prt1
c05f 20 73 00 get   jsr chrget      ;it was !get
c062 f0 0d          beq 10          ;further parameters?
c064 20 eb b7       jsr getpar      ;get further parameters
```

```
c067 86 fd              stx incr      ;and note them
c069 a5 14              lda lo
c06b 85 fb              sta line
c06d a5 15              lda hi
c06f 85 fc              sta line+1
c071 a9 80     10       lda #128
c073 8d 91 02           sta $0291
c076 a9 17              lda #17
c078 8d 18 d0           sta $d018
c07b a9 85              lda #>auto    ;input vector to adjust
c07d 8d 02 03           sta vector    ;line #s for AUTO
c080 a9 c0              lda #>auto    ;line number routine
c082 8d 03 03           sta vector+1
c085 20 8b c0 auto      jsr autonum
c088 4c 86 a4           jmp mainloop
c08b ad 14 03 autonum lda irqvec      ;beep on?
c08e c9 03              cmp #<beepon
c090 f0 0c              beq w0
c092 78                 sei
c093 a9 03              lda #<beepon  ;interrupt vector changed
c095 8d 14 03           sta irqvec    ;for end-of-line
c098 a9 c1              lda #<beepon  ;beep routine
c09a 8d 15 03           sta irqvec+1
c09d 58                 cli
c09e a5 fb     w0       lda line      ;get line number
c0a0 a6 fc              ldx line+1    ;and convert into
c0a2 85 63              sta fac+1     ;ASCII format
c0a4 86 62              stx fac
c0a6 a2 90              ldx #$90
c0a8 38                 sec
c0a9 20 49 bc           jsr intfloat
c0ac 20 dd bd           jsr floatasc
c0af a2 00              ldx #0
```

```
c0b1 bd 01 01 11    lda buffer1,x
c0b4 f0 09          beq 12
c0b6 9d 00 02       sta buffer2,x ;line number in input
c0b9 20 d2 ff       jsr bsout     ;buffer & on screen
c0bc e8             inx
c0bd d0 f2          bne 11
c0bf a5 fb    12    lda line      ;increase line #
c0c1 18             clc
c0c2 65 fd          adc incr
c0c4 85 fb          sta line
c0c6 90 02          bcc 13
c0c8 e6 fc          inc line+1
c0ca a9 20    13    lda #" "      ;space
c0cc 9d 00 02       sta buffer2,x ;in input buffer
c0cf 20 d2 ff       jsr bsout     ;and on screen
c0d2 e8             inx
c0d3 a9 22          lda #34       ;quotation marks
c0d5 9d 00 02       sta buffer2,x ;in input buffer
c0d8 e8             inx
c0d9 20 d2 ff       jsr bsout     ;and on screen
c0dc a9 00          lda #0
c0de 85 d4          sta $d4       ;clear qoute-flag
c0e0 20 cf ff       jsr input
c0e3 c9 0d          cmp #cr       ;only <RETURN> pressed?
c0e5 f0 03          beq 14
c0e7 4c 69 a5       jmp goon      ;if not, go on
c0ea a5 fb    14    lda line      ;otherwise, drop line #
c0ec e5 fd          sbc incr      ;to old value,
c0ee 85 fb          sta line
c0f0 b0 02          bcs 15
c0f2 c6 fc          dec line+1
c0f4 a9 83    15    lda #<oldvec  ;and set back
c0f6 a0 a4          ldy#>oldvec   ;input vector
```

```
c0f8 8d 02 03        sta vector
c0fb 8c 03 03        sty vector+1
c0fe a2 00           ldx #0          ;clear input buffer
c100 4c 76 a5        jmp weiter
c103 a5 d3    beepon lda $d3         ;cursor-column
c105 c9 40           cmp #64         ;64 characters?
c107 f0 07           beq beep        ;then beep
c109 a9 00           lda #0          ;otherwise, clear flag
c10b 85 02           sta beepflg     ;beepflag reset
c10d 4c 31 ea        jmp irqold      ;old irq routine
c110 a5 02    beep   lda beepflg     ;should it beep?
c112 d0 2a           bne end         ;if not, then onward
c114 a9 08           lda #8
c116 8d 18 d4        sta sid+24      ;volume
c119 a9 08           lda #8
c11b 8d 03 d4        sta sid+3       ;key activation
c11e a9 00           lda #0
c120 8d 00 d4        sta sid         ;lowbyte freq
c123 a9 78           lda #120
c125 8d 01 d4q       sta sid+1       ;highbyte freq
c128 a9 ff           lda #255
c12a 8d 05 d4        sta sid+5       ;attack/decay
c12d a9 f9           lda #249
c12f 8d 06 d4        sta sid+6       ;sust/release
c132 a9 41           lda #65
c134 8d 04 d4        sta sid+4       ;pulse wave on
c137 a9 40           lda #64
c139 8d 04 d4        sta sid+4       ;wave off
c13c 85 02           sta beepflg     ;once, no more
c13e 4c 31 ea end    jmp irqold      ;irq exit
```

PRINT ROUTINE

```
c141 a9 04    prtl   lda #4          ;device#
c143 85 ba           sta devad
c145 a9 73           lda #126        ;file#
c147 85 b8           sta lgfnr
c149 a9 00           lda #0          ;filename
c14b 85 b7           sta filnm

c14d a9 00           lda #0          ;secondary address
c14f 85 b9           sta secad
c151 20 c0 ff        jsr open        ;open file
c154 a6 b8           ldx lgfnr
c156 20 c9 ff        jsr ckout       ;output to printer
c159 a5 2d           lda txtan+2     ;=start of variables
c15b 38              sec
c15c e9 02           sbc #2          ;minus 2
c15e c5 2b           cmp txtan       ;basic-start?
c160 d0 0b           bne printl
c162 a5 2e           lda txtan+3
c164 e9 00           sbc #0
c166 c5 2c           cmp txtan+1
c168 d0 03           bne printl
c16a 4c fa c1        jmp endl        ;no text in memory
c16d a5 2b    printl lda txtan       ;line link,
c16f 18              clc             ;line number and
c170 69 05           adc #%          ;quotation marks mark
c172 85 71           sta text        ;position in text
c174 a5 2c           lda txtan+1     ;highbyte also noted
c176 85 72           sta text+1
c178 a9 42           lda #66         ;lines per page
c17a 85 fe           sta pgline
c17c a0 0a           ldy #1m
c17e a9 20           lda #" "        ;print left border
```

```
c180 20 d2 ff lmloopl jsr bsout
c183 88               dey
c184 d0 fa            bne lmloopl
c186 a9 11            lda #17       ;cursor-down
c188 20 d2 ff         jsr bsout     ;upper/lower case
c18b a0 00            ldy #0        ;*****loop*********
c18d b1 71    loop    lda (text),y
c18f d0 03            bne wl        ;end if BASIC line=0
c191 4c a4 cl         jmp null
c194 20 d2 ff wl      jsr bsout     ;send character
c197 20 el ff         jsr stop      ;stop printing?
c19a f0 51            beq end       ;then stop printing
c19c e6 71            inc text      ;increment text pointer
c19e d0 02            bne w2
cla0 e6 72            inc text+l
cla2 d0 e9    w2      bne loop
cla4 a9 0d    null    lda #cr
cla6 20 d2 ff         jsr bsout     ;send linefeed
cla9 c6 fe            dec pgline    ;linecounter -l
clab d0 0c            bne w5        ;66 lines printed?
clad a0 06            ldy #formfeed ;then formfeed (next p.)
claf 20 d2 ff we      jsr bsout
clb2 88               dey
clb3 d0 fa            bne we
clb5 a9 42            lda #66       ;reset linecounter to 66
clb7 85 fe            sta pgline
clb9 a0 0a    w5      ldy #lm       ;left margin for next line
clbb a9 20            lda #" "
clbd 20 d2 ff lmloop jsr bsout
clc0 88               dey
clcl d0 fa            bne lmloop
clc3 a9 11            lda #17
clc5 20 d2 ff         jsr bsout
```

```
c1c8 a0 03              ldy #3
c1ca e6 71       l6     inc text        ;skip null & line-link
c1cc d0 02              bne w3
c1ce e6 72              inc text+1
c1d0 88          w3     dey
c1d1 d0 f7              bne l6
c1d3 a5 71              lda text
c1d5 c5 2d              cmp textend     ;end of BASIC program?
c1d7 d0 06              bne w4
c1d9 a5 72              lda text+1
c1db c5 2e              cmp textend+1
c1dd f0 0e              beq end
c1df a0 03       w4     ldy #3
c1e1 e6 71       l7     inc text        ;skip line number
c1e3 d0 02              bne w6          ;and first character
c1e5 e6 72              inc text+1
c1e7 88          w6     dey
c1e8 d0 f7              bne l7
c1ea 4c 8d c1           jmp loop        ;print next line
c1ed a9 0d       end    lda #cr
c1ef 20 d2 ff           jsr bsout       ;last linefeed given
c1f2 20 cc ff           jsr clrch       ;output on screen
c1f5 a5 b8              lda lgfnr
c1f7 20 c3 ff           jsr close       ;close file
c1fa 4c 74 a4    end1   jmp basloop     ;return to basic
```

If you don't have an assembler, a version of Minitext follows, written as a BASIC loader (i.e., in DATA statements). If you type in either program, be absolutely sure that you've saved it to cassette or diskette before running; once RUN is typed in, you cannot retrieve the program!

```
100 FOR I = 49152 TO 49660
110 READ X :POKEI,X: S=S+X : NEXT
120 DATA 169, 65,160,192,141,  8,  3,140,  9,  3,169,128
130 DATA 141,145,  2,169, 23,141, 24,208,169,  0,168,145
140 DATA  43,200,145, 43,165, 43, 24,105,  2,133, 45,165
150 DATA  44,105,  0,133, 46, 32,142,166, 32, 68,229,169
160 DATA 128,141,138,  2,169,100,133,251,169,  0,133,252
170 DATA 169, 10,133,253, 96, 32,115,  0,201, 33,240,  6
180 DATA  32,121,  0, 76,231,167, 32,115,  0,201,161,240
190 DATA  10,201,153,240,  3, 76,  8,175, 76, 65,193, 32
200 DATA 115,  0,240, 13, 32,235,183,134,253,165, 20,133
210 DATA 251,165, 21,133,252,169,128,141,145,  2,169, 23
220 DATA 141, 24,208,169,133,141,  2,  3,169,192,141,  3
230 DATA   3, 32,139,192, 76,134,164,173, 20,  3,201,  3
240 DATA 240, 12,120,169,  3,141, 20,  3,169,193,141, 21
250 DATA   3, 88,165,251,166,252,133, 99,134, 98,162,144
260 DATA  56, 32, 73,188, 32,221,189,162,  0,189,  1,  1
270 DATA 240,  9,157,  0,  2, 32,210,255,232,208,242,165
280 DATA 251, 24,101,253,133,251,144,  2,230,252,169, 32
290 DATA 157,  0,  2, 32,210,255,232,169, 34,157,  0,  2
300 DATA 232, 32,210,255,169,  0,133,212, 32,207,255,201
310 DATA  13,240,  3, 76,105,165,165,251,229,253,133,251
320 DATA 176,  2,198,252,169,131,160,164,141,  2,  3,140
330 DATA   3,  3,162,  0, 76,118,165,165,211,201, 64,240
340 DATA   7,169,  0,133,  2, 76, 49,234,165,  2,208, 42
```

```
350 DATA 169,  8,141, 24,212,169,  8,141,  3,212,169,  0
360 DATA 141,  0,212,169,120,141,  1,212,169,255,141,  5
370 DATA 212,169,249,141,  6,212,169, 65,141,  4,212,169
380 DATA  64,141,  4,212,133,  2, 76, 49,234,169,  4,133
390 DATA 186,169,126,133,184,169,  0,133,183,169,  0,133
400 DATA 185, 32,192,255,166,184, 32,201,255,165, 45, 56
410 DATA 233,  2,197, 43,208, 11,165, 46,233,  0,197, 44
420 DATA 208,  3, 76,250,193,165, 43, 24,105,  5,133,113
430 DATA 165, 44,133,114,169, 66,133,254,169, 17, 32,210
440 DATA 255,160, 10,169, 32, 32,210,255,136,208,250,160
450 DATA   0,177,113,208,  3, 76,164,193, 32,210,255, 32
460 DATA 225,255,240, 81,230,113,208,  2,230,114,208,233
470 DATA 169, 13, 32,210,255,198,254,208, 12,160,  6, 32
480 DATA 210,255,136,208,250,169, 66,133,254,169, 17, 32
490 DATA 210,255,160, 10,169, 32, 32,210,255,136,208,250
500 DATA 160,  3,230,113,208,  2,230,114,136,208,247,165
510 DATA 113,197, 45,208,  6,165,114,197, 46,240, 14,160
520 DATA   3,230,113,208,  2,230,114,136,208,247, 76,141
530 DATA 193,169, 13, 32,210,255, 32,204,255,165,184, 32
540 DATA 195,255, 76,116,164
550 IF S <> 64452 THEN PRINT "ERROR IN DATA !!" : END
560 PRINT "OK"
570 SYS 12 * 4096
```

The program will run "as-is" on a C-64, and is compatible with all Commodore printers run on the serial bus.

After starting the program with a SYS 49152, different memory locations are initialized, and a NEW command is executed, which deletes the loader program; text can now be entered without further ado.

To get to input mode, type in the command !GET without a line number. The line number 100 and quotation marks will appear immediately on screen. Now you can proceed in writing your letter or article.

The >BEEP< sounds when the line passes the 64-character mark. Remember, though, that the line length includes the line number; after the >BEEP<, you should hit <RETURN> at the next possible opportunity, otherwise the printer output will be messed up. Ten spaces precede every line, bringing the printed line to a total of 70 characters. If this number is overstepped, the printer automatically performs a linefeed, and prints the offending line on paper without the left margin.

You can leave input mode by pressing <RETURN> alone after a line number appears; this returns you to normal mode on the computer. Now you can input all the commands that the computer would normally respond to, although the most important commands to Minitext would be LIST, LOAD and SAVE.

If you've left input mode, you can list and edit the text as if it were a normal BASIC program. If you retype !GET, the text editor will pick up from the line number at which you had hit <RETURN> to exit. It's obvious that you can add line numbers "by hand": Just type the line as it would appear on screen in input mode.

In addition to !GET, there are two parameters that you can control. The syntax is like this:

!GETline number, line increment

!GET 200,5 would start at line 200, and the next line would be 205, then 210, etc. This is useful if you have an already-begun text that you wish to add to: You give a line number that is higher than the end of the text previously loaded.

The !PRINT command has no additional parameters: As soon as the word is given, the program opens a channel with a logical file number of 126, and sends the text to the printer in upper/lower-case. The printout can be stopped at any time by pressing the RUN-STOP key.

It is possible to change some printer parameters manually, dependent upon your own printer's features. For example, let's take this text:

XY-Diskettes in the Family Size Box
2000 diskettes ---- only $19.95

We want to print the bottom line of text in double-width characters: Most printers switch on their double-width mode when they receive the control character CHR$(14). The problem is to convey this code to the printer. The C-64 has a CONTROL key; by using this key with a letter key, you can send ASCII values from 1 (CTRL-A) to 26 (CTRL-Z). CTRL-N = 14 in ASCII. This character can only be sent in quote mode as a part of the text. This is complicated because quote mode is switched off in input mode.

The solution is as follows: First, write the text as usual, then leave input mode by hitting <RETURN> when the next line number comes up. Now, move the cursor over the 2 in the line of text we want enhanced. Press SHIFT-INST/DEL, insert a space, and press CTRL-N. A reverse-video N appears on screen, which will control the double-width switching when the time comes to print the text. The same procedure applies to all the control characters, from CTRL-A to CTRL-Z. ASCII code 27 (CTRL-:) is graphically represented by a reverse bracket, while CTRL-; (29) displays the other reverse bracket.

Avoid CTRL-@ at all costs! This combination produces the value 0, which the program sees as the end of a BASIC line. Whatever follows that zero in that line will be skipped over in the printing stage.

If you find that you like using this text editor, you may be best off purchasing a proper word processing program. Minitext will allow you to do such things as move blocks of text, merge texts into a longer document, and add new material to old, but a great deal of effort is involved, unlike "real" word processors. On the other hand, this program is just right for short letters and such.

5.2.2 What About Posters?

For printing out letters, manuscripts or listings, the characters on the dot-matrix printer are the right size. What if we want to make larger characters for a change?

If you happen to own a 7-pin dot-matrix printer (MPS 801, VIC-1515, VIC-1525, EPSON, etc.), the following program will allow you to print characters many times normal size, with a minimum of difficulty.

The problem in printing such characters lies in the construction of the character set. The amount of code required to alter the character set can add up to plenty of kilobytes worth of data--and a frustrated user.

Then again, why go to the trouble of redesigning a character set, when we already have a perfectly good one in the computer? Both the VIC-20 and the C-64 allow their character generators to be read out. This capability is the foundation on which our project is built.

It's been said that you cannot read out the character generator from BASIC, otherwise the computer locks up. That isn't necessarily true--read on for a simple trick to get around this.

Thanks to the partial repetitions in the memory layout of the C-64, the character generator and the I/O ICs (the VIC-II chip, the CIAs and the SID) all reside in the same memory range. Whether the processor is accessing the character generator or the peripheral ICs, bit 2 in the processor holds the designation -CHAREN.

If this bit is low, the character generator is on. This bit can be influenced with a POKE to address 1.

If we POKE 1,51 (-CHAREN=low), we'll get a READY prompt, and no more obvious activity from the computer. What we've done here is deprive CIA1 of any access to the microprocessor. This IC produces a periodical interrupt (60 times per second), which can only operate if the CIA has access to the processor. If the character generator is to be read, then the interrupt has to be switched off.

This can be accomplished in machine language with the command SEI. This command isn't applicable to BASIC, so we'll need another method; we'll take away the source of the interrupt by clearing the ICR (Interrupt Control Register) in CIA1.

For the moment, we'll forego details of how this is done: However, the most important items to remember are--- POKE 56333,127 turns off the interrupt, while POKE 56333,129 turns it back on.

The situation is even more easily solved on the VIC-20. The character generator is always in the access register of the processor, so no precautions need be taken.

Now that we've cleared that up, let's consider what sort of data we'll need to reach our goal.

First, we'll want to look at how a character is constructed in the character generator.

Individual characters on screen are made up of points set in an 8 X 8 matrix, just like dot-matrix-printed characters. Unlike the printout, the screen is arranged in a set of horizontal dot sequences (printers are vertically arranged), in keeping with normal video operation.

The eight points in such a series is stored in the character generator as 8 bits (=1 byte). If a bit is set, a dot appears onscreen; if cleared (or unset), the bit remains the background color. One character is made up of 8 bytes stored successively in the character generator.

There is a peculiarity in this system which you should note: The sequence of characters is stored as screen codes, not in ASCII values. The first 8 bytes of the character generator contains the bit pattern for '@', the next 8 form 'A', and so on.

If we want to create larger characters on the printer, we can utilize single-pin mode. We want to make every bit into a black field seven points high, and six points across. Since 480 points per line are available to us in high-resolution mode, this gives us a choice of "point size" in the character matrix of 64 points high, and 48 points wide. This gives us exactly 10 characters per line.

On with the program. Since reconstructing the character ganerator takes up some time, we've divided the program up. The first program gives the printer control of the first 64 characters in the character generator. The data produced here can be saved as a sequential file on the disk drive for later use.

The second program reads the data, places it in an array, and prompts for a text to be printed out. This text will then be printed.

The first limitation to this program is that you must stay with the first 64 characters in the set; considering how little you might use the graphic characters and control characters, this is a small problem. Another disadvantage is that the data is four times as large on disk as normal. Finally, the printing process can take a very long time.

Those without disk drives can simply combine the two programs into one. Any way you look at it, you'll have to wait while the printer data is reconstructed.

First, the listing for producing the print data.

```
100 DIM A%( 63,7)
110 POKE56320+13,127      : REM ALSO APPLIES TO VIC-20
120 POKE 1,51             : REM ASLO APPLIES TO VIC-20
130 A=13*4096             : REM 8*4096 FOR VIC-20
140 FORK=0TO 63
150 PRINTK,
160 FORI=0TO7
170 A%(K,I)=PEEK(A+I)
180 NEXTI
190 A=A+8
200 NEXTK
210 POKE1,55              : REM ALSO FOR VIC-20
220 POKE56320+13,129      : REM ALSO FOR VIC-20
230 OPEN15,8,15
240 OPEN1,8,4,"NORM CHR DATA,S,W"
```

```
250 GOSUB460
260 PRINTCHR$(147);
270 FORK=0TO 63
280 PRINTCHR$(K+((K<32)*-64));
290 FORJ=0TO7
300 A=A%(K,J)
310 FORI=7TO0STEP-1
320 IF(2^I=(AAND2^I))THENGOSUB420:GOTO340
330 GOSUB440
340 NEXTI
350 PRINT#1,X$
360 X$=""
370 NEXTJ
380 NEXTK
390 CLOSE1
400 CLOSE15
410 END
420 X$=X$+CHR$(8)+CHR$(26)+CHR$(5)+CHR$(255)
430 RETURN
440 X$=X$+CHR$(8)+CHR$(26)+CHR$(5)+CHR$(128)
450 RETURN
460 INPUT#15,EN,ET$,ET,ES
470 IFEN=0THENRETURN
480 PRINTEN;ET$;ET,ES
490 CLOSE15:CLOSE1
```

Now, the program that does the printing.

```
100 DIM A$( 63,7)
110 PRINTCHR$(147)
120 OPEN15,8,15
130 OPEN4,8,4,"NORM CHR DATA,S,R"
140 GOSUB380
150 FORK=0TO 63
160 PRINTK,
170 FORI=0TO7
180 INPUT#4,A$(K,I)
190 NEXTI,K
200 CLOSE4
210 CLOSE15
220 PRINT
230 A$=""
240 INPUT"TEXT (MAX.10 CHARACTERS)";A$
250 IFA$<>""THEN280
260 PRINT#1,CHR$(15)
270 END
280 PRINTCHR$(147);A$
290 OPEN1,4
300 FORI=0TO7
310 FORL=1TOLEN(A$)
320 K=ASC(MID$(A$,L,1))
330 K=K+((K>63)*64)
340 PRINT#1,A$(K,I);
350 NEXTL:PRINT#1:NEXTI
360 CLOSE1
370 GOTO220
380 INPUT#15,EN,ET$,ES,ET
390 IFEN=0THENRETURN
400 PRINTEN;ET$;ES;ET
410 CLOSE15
420 CLOSE4
```

You aren't totally limited to 10 characters per line. If you want to print out a longer text, the program will let you turn the text around, then it would just be a matter of cutting and pasting the sheets together.

Maybe the standard character size isn't enough for all occasions. The program below offers some help for this as well.

In order to print out longer text, it can be turned 90 degrees, so that the characters are printed in a different format. Now the text length is limited only to the length of the printer paper (a 2000-sheet carton of paper totals up to 600 meters). This also means that the characters can be printed much larger than originally. During the reconstruction of print data, the printer asks for the size desired (chosen in steps between 1 and 5). Size 1 gives you the standard height (for 10 characters per line), while size 5 takes up the entire width of the paper. This produces lettering that can be seen from more than ten meters away, making these posters useful for publicity purposes, or for sports events.

If you are printing out longer texts, have patience: Your printer is no longer working in speeds of characters per second, rather, only in characters per minute. Since the printer is producing large numbers of black blocks, we suggest that you give the machine a short break every ten characters or so (in size 5); otherwise, the printhead becomes quite hot.

```
100 DIM A%( 63,7)
110 PRINTCHR$(147)
120 NA$="CHARACTER DATA"
130 INPUT "SIZE (1-5)";V
140 IFV<1ORV>5THEN110
150 PRINTCHR$(147);
160 NA$=NA$+STR$(V)+",S,W"
170 POKE56320+13,127 : REM INTERRUPT GIVEN ONLY FOR C64
180 POKE 1,51        : REM CHAREN ONLY FOR 64
190 REM CHARGENERATOR IN ARRAY A%(K,I)
200 A=13*4096        : REM A=8*4096 FOR VIC-20
210 FORK=0TO 63:PRINTK,:FORI=0TO7
220 A%(K,I)=PEEK(A+I)
230 NEXT:A=A+8:NEXT
240 POKE1,55:POKE56320+13,129 : REM INTERRUPT REMOVED
260 OPEN15,8,15
270 OPEN1,8,4,NA$
280 GOSUB440
290 PRINTCHR$(147);
300 FORK=0TO 63:PRINTCHR$(K+((K<32)*-64));
310 FORI=7TO0STEP-1
320 FORJ=7TO0STEP-1
330 A=A%(K,J)
340 IF(2^I=(AAND2^I))THENGOSUB400:GOTO360
350 GOSUB420
360 NEXTJ:PRINT#1,X$:X$=""
370 NEXTI,K
380 CLOSE1:CLOSE15
390 END
400 X$=X$+CHR$(8)+CHR$(26)+CHR$(12*V)+CHR$(255)
410 RETURN
420 X$=X$+CHR$(8)+CHR$(26)+CHR$(12*V)+CHR$(128)
430 RETURN
```

```
440 INPUT#1,EN,ET$,ES,ET
450 IFEN=0THENRETURN
460 PRINTEN;ET$;ES;ET
470 CLOSE1:CLOSE15
```

Again, here's the print program:

```
100 INPUT "SIZE (1-5)";V
110 POKE56320+13,127 : REM INTERRUPT PROHIBIT
120 POKE 1,51 : REM CHAREN TO LOW
130 REM CHARGENERATOR IN ARRAY A%(K,I)
140 DIM A%( 63,7),A$( 63,7)
150 A=13*4096
160 FORK=0TO 63
170 : PRINTK
180 : FORI=0TO7
190 : : A%(K,I)=PEEK(A+I)
200 : NEXT
210 : A=A+8
220 NEXT
230 POKE1,55:POKE56320+13,129 : REM INTERRUPT ENABLE
240 FORK=0TO 63
250 : PRINTK
260 : FORI=7TO0STEP-1
270 : : FORJ=7TO0STEP-1
280 : : : A=A%(K,J)
290 : : : IF(2^I=(AAND2^I))THENGOSUB480:GOTO310
300 : : : GOSUB500
310 : : NEXTJ
320 : : A$(K,7-I)=X$
330 : : X$=""
```

```
340 : NEXTI
350 NEXTK
360 INPUT"TEXT";A$
370 OPEN1,4
380 FORL=1TOLEN(A$)
390 : PRINTCHR$(19); MID$(A$,L,1)
400 : K=PEEK(1024)
410 : FORI=0TO7
420 : : FORJ=1TOV
430 : : PRINT#1,A$(K,I)
440 : : NEXTJ
450 : NEXTI
460 NEXTL
470 CLOSE1:GOTO360
480 X$=X$+CHR$(8)+CHR$(26)+CHR$(12*V)+CHR$(255)
490 RETURN
500 X$=X$+CHR$(8)+CHR$(26)+CHR$(12*V)+CHR$(128)
510 RETURN
```

5.3 Hardcopy

Just a few years ago, printers for hobbyists were unheard-of. A simple dot-matrix printer with minimal features cost astronomical sums of money, much more than the early user's checkbook could handle. So, if someone wanted a printout, they had to improvise.

Surplus teletypes sold very well in spite of the interfacing problems; the price was well worth the hassles ($75.00 was a good price). These users were overjoyed when, after building special interfaces and designing driver software, their first program listings rolled out (noisily, and with lots of garbage printed out as well). These teletypes were also bought in spite of the limited stock of characters, as compared to the computer's character set.

In those times, you had to admire the program listings printed in some technical journals. Some brilliant people devised a way of printing out program listings from the screen contents (screen photocopies would be a better term).

Now, of course, the situation has improved considerably. You don't need a darkroom next to the computer--the printer will produce graphics just as well as the screen does.

To make these screen dumps, the program below works nicely.

5.3.1 Text Hardcopy

The characters appearing on your screen are simply reproductions of a predetermined memory range in your computer. This memory (always in RAM) is free for reading and writing to: A character can only be printed to the screen using POKE, while PEEK allows you to see the screen code.

If, for example, the letter A is in the upper left corner of the screen, PEEKing the screen memory will give you the number 1. In other words, PRINT PEEK (address), in which (address) would be in the range from 1024 to 2023 (1024 {$0400 hex} is the upper left corner) for the C-64; the VIC-20's screen memory begins at 4096 ($1000) or 7680 ($1E00), depending on the amount of memory expansion involved.

Commodore International has a rather strange method of coding the characters displayed. The screen RAM codes for POKEing characters is similar to CHR$ in only a few cases; most of these screen codes need to be recalculated to correspond with their ASCII values.

The best possible hardcopy program should call itself, as well as produce a screen contents printout. This program has both. The program, called with a SYS command, goes on to hardcopy without fanfare. Pressing CTRL-F1 simultaneously will produce hardcopy. The assembler listing below is tailored for the C-64, while the BASIC loader which follows can be used with the C-64 and the VIC-20. VIC-20 users who wish to use the assembler listing should check a comparison table of the operating system routines for both machines.

```
033C                    *=$033C
033C         IRQ        =$314
033C         KEY        =$EA87
033C         CTRL       =$028D
033C         INTER      =$EA31
033C         IRQFLAG    =$02
033C         KEYS       =$C5
033C         FA         =$BA
033C         LF         =$B8
033C         TEMP       =$71
033C         FNLEN      =$B7
033C         AMNT       =$C6
033C         SA         =$B9
033C         OPEN       =$FFC0
033C         CKOUT      =$FFC9
033C         BSOUT      =$FFD2
033C         STOP       =$FFE1
033C         STORE      =$67
033C         CLRCH      =$FFCC
033C         CLOSE      =$FFC3
033C 78                 SEI         ;INTERRUPT OFF
033D A9 49              LDA #<NEW   ;INTER.VECTOR OFF
033F A0 03              LDY #>NEW   ;TURN DOWN HARDCOPY
0341 8D 14 03           STA IRQ
0344 8C 15 03           STY IRQ+1
0347 58                 CLI
0348 60                 RTS         ;AND BACK

; NEW INTERRUPT ROUTINE
0349 20 87 EA NEW       JSR KEY     ;READ KEYBOARD
034C AD 8D 02           LDA CTRL    ;CTRL-KEY PRESSED?
034F C9 04              CMP #4      ;THEN SET BIT 2
0351 D0 79              BNE EXIT1   ;IF NOT, THEN NO HC
```

```
0353 A3 C3                LDA KEYS
0355 C9 04                CMP #4      ;F1 PRESSED?
0357 D0 73                BN1 EXIT1   ;NO? THEN NO HC
0359 A9 00                LDA #0
035B 85 C6                STA AMNT
035D F0 04                BEQ HCOPY   ;UNCONDITIONAL JUMP
035F A9 FF     SYSONE     LDA #$FF    ;JUMP FOR SYS MARKED,
036B 85 B8                STA IRQFLAG;IF NOT OVER IRQ
0363 A9 04     HCOPY      LDA #4      ;HARDCOPY (HC)
0365 85 BA                STA FA      ;DEVICE ADDRESS
0367 A9 7E                LDA #126
0369 85 B8                STA LF      ;LOG. FILE NUMBER
036B A9 00                LDA #0
036D A0 04                LDY #4      ;VIDEO RAM @ $0400
036F 85 71                STA TEMP
0371 84 72                STY TEMP+1
0373 85 B7                STA FNLEN   ;NO FILENAME
0375 A9 00                LDA #0
0377 85 B9                STA SA      ;SECONDARY ADDRESS
0379 20 C0 FF             JSR CKOUT
037C A6 B8                LDX LF
037E 20 C9 FF             JSR CKOUT
0381 A2 19                LDX #25     ;25 SCREEN LINES
0383 A9 0D      LOOP      LDA #13     ;LINEFEED
0385 20 D2 FF             JSR BSOUT   ;OUTPUT
0388 20 E1 FF             JSR STOP    ;STOPKEY PRESSED?
038B F0 2E                BEQ EXIT    ;THEN END
038D A0 00                LDY #0
038F B1 71      LOOP2     LDA(TEMP),Y;GET CHARACTER FROM
0391 85 67                STA STORE   ;VIDEO RAM & STORE IT
0393 29 3F                AND #$3F
0395 06 67                ASL STORE
0397 24 67                BIT STORE   ;CONVERT SCREEN CODE
```

```
0399 10 02                BPL *+4      ;INTO ASCII CODE
039B 09 80                ORA #$80
039D 70 02                BVS *+4
039F 09 40                ORA #$40
03A1 20 D2 FF             JSR BSOUT    ;AND PUT OUT
03A4 C8                   INY
03A5 C0 28                CPY #40      ;40 CHARS. PRINTED?
03A7 D0 E6                BNE LOOP2
03A9 98                   TYA
03AA 18                   CLC
03AB 65 71                ADC TEMP     ;COUNTER IN VIDEORAM
03AD 85 71                STA TEMP     ;RAISED TO 40
03AF 90 02                BCC *+4
03B1 E6 72                INC TEMP+1
03B3 CA                   DEX
03B4 D0 CD                BNE LOOP     ;ALL LINES PRINTED?
03B6 A9 0D                LDA #13      ;OUTPUT NEW LINE
03B8 20 D2 FF             JSR BSOUT
03BB 20 CC FF   EXIT      JSR CLRCH    ;OUTPUT BACK TO SCRN.
03BE A9 7E                LDA #126
03C0 20 C3 FF             JSR CLOSE    ;CLOSE FILE
03C3 A5 02                LDA IRQFLAG;HC STOPPED BY CTRL-F1?
03C5 F0 05                BEQ EXIT1    ;THEN GOTO EXIT1
03C7 A9 00                LDA #$00     ;CLEAR FLAG
03C9 85 02                STA IRQFLAG
03CB 60                   RTS          ;RETURN FOR SYS CALL
03CC 4C 31 EA   EXIT1     JMP INTER    ;CALL OLD INTERRUPT
```

As you can see, this program resides in the casette buffer (828 decimal). This is the address used to activate the program with SYS 828. A logical file number of 126 is used. If you also wish to use this file number in other programs afterward, be sure that the file is closed after using the Hard Copy program. In order to prevent errors, reserve 126 as a file number for printing hardcopy, and don't use 126 for other files.

To get a hardcopy, all you need do is press CTRL and F1 simultaneously. If you need to break out of the program, you can do so with SYS 863, or press STOP.

Now for the BASIC loaders for C-64 and VIC-20. The 64 program first:

```
100 FOR I = 828 TO 974
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 120,169, 73,160,  3,141, 20,  3,140, 21,  3, 88
130 DATA  96, 32,135,234,173,141,  2,201,  4,208,121,165
140 DATA 197,201,  4,208,115,169,  0,133,198,240,  4,169
150 DATA 255,133,  2,169,  4,133,186,169,126,133,184,169
160 DATA   0,160,  4,133,113,132,114,133,183,169,  0,133
170 DATA 185, 32,192,255,166,184, 32,201,255,162, 25,169
180 DATA  13, 32,210,255, 32,225,255,240, 46,160,  0,177
190 DATA 113,133,103, 41, 63,  6,103, 36,103, 16,  2,  9
200 DATA 128,112,  2,  9, 64, 32,210,255,200,192, 40,208
210 DATA 230,152, 24,101,113,133,113,144,  2,230,114,202
220 DATA 208,205,169, 13, 32,210,255, 32,204,255,169,126
230 DATA  32,195,255,165,  2,240,  5,169,  0,133,  2, 96
240 DATA  76, 49,234
250 IF S <> 17800 THEN PRINT "ERROR IN DATA!!" : END
260 PRINT "OK":SYS 828
```

In the list that follows for the VIC-20, the highbyte of the starting address of video RAM has been underlined. The present value is for an unexpanded VIC; if you have at least 8K of memory expansion plugged in, change this value from 30 to 16.

```
100  FOR I = 828 TO 974
110  READ X : POKE I,X : S=S+X : NEXT
120  DATA 120,169, 73,160,  3,141, 20,  3,140, 21,  3, 88
130  DATA  96, 32, 30,235,173,141,  2,201,  4,208,121,165
140  DATA 197,201,  4,208,115,169,  0,133,198,240,  4,169
150  DATA 255,133,  2,169,  4,133,186,169,126,133,184,169
160  DATA   0,160, 30,133,113,132,114,133,183,169,  0,133
170  DATA 185, 32,192,255,166,184, 32,201,255,162, 23,169
180  DATA  13, 32,210,255, 32,225,255,240, 46,160,  0,177
190  DATA 113,133,103, 41, 63,  6,103, 36,103, 16,  2,  9
200  DATA 128,112,  2,  9, 64, 32,210,255,200,192, 22,208
210  DATA 230,152, 24,101,113,133,113,144,  2,230,114,202
220  DATA 208,205,169, 13, 32,210,255, 32,204,255,169,126
230  DATA  32,195,255,165,  2,240,  5,169,  0,133,  2, 96
240  DATA  76,191,234
250  IF S <> 17844 THEN PRINT "ERROR IN DATA!!" : END
260  PRINT "OK"
270  SYS 828
```

5.3.2 Graphic Hardcopy

In order to print out of text-screens, we should see whether or not such graphics can be produced.

Good graphic extensions such as VIDEO BASIC 64 already contain special routines and commands to send the screen contents to the printer. However, when graphics are made in BASIC with PEEK and POKE, there are few utility programs suited to this task.

Below are two graphic hardcopy routines for the C-64, which you will find very convenient. The first routine operates a 7-pin printer. The second routine is very special, as it makes possible a screen printout on a 1526 printer--if you'll remember the chapter a few pages back, the 1526 is incapable of high-res graphics. Both routines lie in an address range that allows you to use them with the graphics in the book THE GRAPHICS BOOK FOR THE COMMODORE 64 by Abacus Software.

We'll follow the past procedure of giving you the assembler listings first, followed by the BASIC loader program.

Now we come to the graphic hardcopy program for the 7-pin printer.

```
C200                    *=   $C200
C200            A        =   $AC
C200            B        =   A+1
C200            USE      =   $FD
C200            XK       =   $14
C200            FLG      =   $97
C200            OFFX     =   $63
C200            GRMEM    =   $2000
C200 20 F1 B7           JSR $B7F1
C203 86 67              STX $67
C205 20 0F F3           JSR $F30F ;LOOK FOR LOG. FILE #
C208 20 1F F3           JSR $F31F ;SET FILE PARAMETERS
C20B A6 67              LDX $67
C20D 20 C9 FF           JSR $FFC9 ;OPEN CHANNEL
C210 A9 FF              LDA #$FF
C212 85 61              STA $61   ;MASK
C214 A9 07              LDA #7
C216 85 FD              STA USE   ;STACK SIZE
C218 A9 1C              LDA #28
C21A 85 97              STA FLAG  ;LINE COUNTER
C21C A9 00              LDA #0
C21E 8D 0D C3           STA BTWN  ;YK-MARKER
C221 A9 28    SEV1      LDA #40
C223 8D 0F C3           STA FLG2  ;STACK COUNTER
C226 A2 04              LDX #4
C228 BD B1 C2 SEV11     LDA SEVTAB,X ;CENTERING
C22B 20 D2 FF           JSR $FFD2
C22E CA                 DEX
C22F 10 F7              BPL SEV11
C231 A9 00              LDA #0
C233 85 63              STA $63
C235 85 64              STA $64      ;XK=0
C237 AD 0D C3 SEV2      LDA BTWN
```

```
C23A 85 65              STA $65       ;YK
C23C A9 00              LDA #0
C23E 85 FE              STA USE+1
C240 A5 63     SEV3     LDA $63       ;XK-L
C242 A6 64              LDX $64       ;XK-H
C244 A4 65              LDY $65       ;YK
C246 20 B6 C2           JSR HPOSN     ;CALCULAtE POSITION
C249 A0 00              LDY #0
C24B B1 AC              LDA (A),Y
C24D A6 FE              LDX USE+1
C24F 9D 11 C3           STA BUFFER,X ;IN BUFFER
C252 E6 65              INC $65       ;YK
C254 E8                 INX
C255 86 FE              STX USE+1
C257 E4 FD              CPX USE
C259 D0 E5              BNE SEV3
C25B A9 00              LDA #0
C25D A0 07              LDY #7
C25F A6 FD     SEV4     LDX USE
C261 1E 11 C3  SEV5     ASL BUFFER,X
C264 2A                 ROL A
C265 CA                 DEX
C266 10 F9              BPL SEV5
C268 25 61              AND $61     ;LINE=#$0F FR. LAST BYTE
C26A 09 80              ORA #$80      ;HIGH-BYTE
C26C 20 D2 FF           JSR $FFD2     ;SEND
C26F 88                 DEY
C270 10 ED              BPL SEV4
C272 A5 63              LDA $63       ;XK-L
C274 18                 CLC
C275 69 08              ADC #8
C277 85 63              STA $63       ;XK+8
C279 90 02              BCC SEV6
```

```
C27B E6 64              INC $64        ;XK-H
C27D CE 0F C3 SEV6      DEC FLG2
C280 D0 05              BNE SEV2
C282 A9 0D              LDA #$D
C284 20 D2 FF           JSR $FFD2
C287 AD 0D C3           LDA BTWN
C28A 18                 CLC
C28B 69 07              ADC #7
C28D 8D 0D C3           STA BTWN       ;MARKER+7
C290 C6 97              DEC FLG
C292 F0 03              BEQ SEV81
C294 4C 21 C2 SEV8      JMP SEV1
C297 A9 04    SEV81     LDA #4
C299 C5 FD              CMP USE
C29B F0 0C              BEQ SEV7
C29D 85 FD              STA USE        ;LAST LINE
C29F A9 01              LDA #1
C2A1 85 97              STA FLG
C2A3 A9 0F              LDA #$F
C2A5 85 61              STA $61        ;MASK
C2A7 D0 EB              BNE SEV8
C2A9 A9 0F    SEV7      LDA #15        ;NORMAL MODE
C2AB 20 D2 FF           JSR $FFD2
C2AE 4C CC FF           JMP $FFCC      ;CLOSE CHANNEL
C2B1 50 00 10 SEVTAB    .BYT 80,0,16,27,8
C2B6 85 14    HPOSN     STA XK
C2B8 86 15              STX XK+1
C2BA 98                 TYA
C2BB 4A                 LSR A
C2BC 4A                 LSR A
C2BD 4A                 LSR A
C2BE AA                 TAX
C2BF BD EF C2           LDA MULH,X
```

```
C2C2 85 AD              STA B
C2C4 8A                 TXA
C2C5 29 03              AND #3
C2C7 AA                 TAX
C2C8 BD 09 C3           LDA MULL,X
C2CB 85 AC              STA A
C2CD 98                 TYA
C2CE 29 07              AND #7
C2D0 18                 CLC
C2D1 65 AC              ADC A
C2D3 85 AC              STA A
C2D5 A5 14              LDA XK
C2D7 29 F8              AND #$F8
C2D9 85 63              STA OFFX
C2DB A9 20              LDA #>GRMEM  ;RUN GRAPHIC PAGE
C2DD 05 AD              ORA B
C2DF 85 AD              STA B
C2E1 18                 CLC
C2E2 A5 AC              LDA A
C2E4 65 63              ADC OFFX
C2E6 85 AC              STA A
C2E8 A5 AD              LDA B
C2EA 65 15              ADC XK+1
C2EC 85 AD              STA B
C2EE 60                 RTS
C2EF 00 01 02 MULH    .BYT0,1,2,3,5,6,7,8,10,11,12,13,15,16
C2FD 11 12 14         .BYT17,18,20,21,22,23,25,26,27,28,30,31
C309 00 40 80 MULL    .BYT0,$40,$80,$C0
C30D 00 00      BTWN  .WOR 0000
C30F 00 00      FLG2  .WOR 0000
C311 00       BUFFER  .BYT 00
```

Next, the listing for 1526 users.

```
C200                   *=   $C200
C200           ZF       =   $68
C200           Z3       =   $69
C200           A        =   $AC
C200           B        =   A+1
C200           GRMEM    =   $2000
C200 20 F1 B7           JSR $B7F1
C203 86 67              STX $67
C205 20 0F F3           JSR $F30F ;LOOK FOR LOG. FILE #
C208 20 1F F3           JSR $F31F ;SET FILE PARAMETERS
C20B A9 05 EIGHT        LDA $5
C20D 20 2F C3           JSR OPENCH;OPEN SA=5
C210 A9 06              LDA #6
C212 20 2F C3           JSR OPENCH;OPEN SA=6
C215 A2 16              LDX #$16
C217 20 C9 FF           JSR $FFC9
C21A A9 14              LDA #20   ;LINE POS. FOR GRAPHIC
C21C 20 D2 FF           JSR $FFD2
C21F A9 0D              LDA #13   ;END CHARACTER
C221 20 D2 FF           JSR $FFD2
C224 20 CC FF           JSR $FFCC ;CLEAR CHANNEL
C227 A0 19              LDY #25   ;NUMBER OF LINES
C229 84 64              STY $64   ;SET-UP
C22B A9 02    HA10      LDA #2    ;DOUBLE HEIGHT
C22D 85 68              STA ZF    ;REPEAT NUMBER
C22F A9 00    HA10A     LDA #0    ;NUMBER OF GUIDING BLANKS
C231 85 65              STA $65   ;SET
C233 A9 28              LDA #40   ;NUMBER OF COLUMNS
C235 85 63              STA $63   ;SET
C237 A9 19              LDA #25
C239 38                 SEC
```

```
C23A E5 64              SBC $63
C23C AA                 TAX
C23D BD 38 C3           LDA MULH,X
C240 09 20              ORA #>GRMEM
C242 85 AD              STA B
C244 8A                 TXA
C245 29 03              AND #3
C247 AA                 TAX
C248 BD 52 C3           LDA MULL,X
C24B 85 AC              STA A
C24D A2 00     HA11     LDX #0          ;GET GRAPHIC DATA
C24F A0 08              LDY #8
C251 A1 AC     HA1L2    LDA (A,X)
C253 99 56 C3           STA BUFFER,X
C256 E6 AC              INC A
C258 D0 02              BNE HA1L2
C25A E6 AD              INC B
C25C 88        HA1L01   DEY
C25D D0 F2              BNE HA1L2
C25F A0 08     HA1L4    LDY #8
C261 A2 08     HA1L3    LDX #8
C263 1E 56 C3  HA1L3    ASL BUFFER,X
C266 2A                 ROL A
C267 CA                 DEX
C268 D0 F9              BNE HA1L3
C26A 20 0E C3           JSR BSPLIT
C26D 99 5E C3           STA BUFFER+8,Y
C270 88                 DEY
C271 D0 EE              BNE HA1L4
C273 A0 08     HA1L02   LDY #8          ;PRINT GRAPHICS
C275 20 A6 C2           JSR PRDFCH
C278 A0 04              LDY #4
C27A 20 A6 C2           JSR PRDFCH
```

```
C27D C6 63              DEC $63
C27F D0 CC              BNE HA11
C281 C6 68              DEC ZF
C283 D0 AA              BNE HA10A
C285 C6 64     HA1L50   DEC $64       ;DECREMENT LINE #
C287 D0 A2              BNE HA10      ;STILL NOT READY
C289 A2 16              LDX #$16
C28B 20 C9 FF           JSR $FFC9     ;SA=6
C28E A9 24              LDA #36
C290 20 D2 FF           JSR $FFD2     ;LINE DISTANCE-TEXT
C293 A9 0D              LDA #13
C295 20 D2 FF           JSR $FFD2     ;END CHARACTER
C298 20 C3 FF           JSR $FFC3
C29B A9 16              LDA #$16      ;SA=6
C29D 20 C3 FF           JSR $FFC3
C2A0 A9 19              LDA #25
C2A2 20 C3 FF           JSR $FFC3;CLOSE SPECIAL CHAR.CHANNEL
C2A5 60                 RTS           ;READY
C2A6 A9 00     PRDFCH   LDA #0
C2A8 85 66              STA $66
C2AA A2 15              LDX #$15
C2AC 20 C9 FF           JSR $FFC9     ;CHAR DEFINITION CHANNEL
C2AF 98                 TYA
C2B0 48                 PHA
C2B1 A2 04              LDX #4
C2B3 B9 5E C3 HA62      LDA BUFFER+8,Y
C2B6 20 D2 FF           JSR $FFD2
C2B9 20 D2 FF           JSR $FFD2
C2BC 05 66              ORA $66
C2BE 85 66              STA $66
C2C1 CA                 DEX
C2C2 D0 EF              BNE HA62
C2C4 20 CC FF           JSR $FFCC     ;CLOSE CHANNEL
```

```
C2C7 A5 66            LDA $66
C2C9 F0 31            BEQ HA61
C2CB A6 67            LDX $67
C2CD 20 C9 FF         JSR $FFC9      ;DATA CHANNEL
C2D0 A5 65            LDA $65        ;NUMBER OF BLAMKS
C2D2 F0 09            BEQ HA63       ;NONE
C2D4 AA               TAX
C2D5 A9 20            LDA #$20
C2D7 20 D2 FF HA64    JSR $FFD2
C2DA CA               DEX
C2DB D0 FA            BNE HA64
C2DD A9 FE    HA63    LDA #254       ;SPECIAL CHARACTERS
C2DF 20 D2 FF         JSR $FFD2
C2E2 68       HA61A   PLA
C2E3 C9 04            CMP #4
C2E5 D0 0A            BNE HA75A
C2E7 A5 63            LDA $63
C2E9 C9 01            CMP #1
C2EB D0 04            BNE HA75A
C2ED A9 0D            LDA #13
C2EF D0 02            BNA HA75
C2F1 A9 8D    HA75A   LDA #141
C2F3 20 D2 FF HA75    JSR $FFD2
C2F6 20 CC FF         JSR $FFCC      ;CLOSE CHANNEL
C2F9 E6 65            INC $65
C2FB 60               RTS
C2FC A5 63    HA61    LDA $63
C2FE C9 01            CMP #1
C300 D0 08            BNE HA61B
C302 A6 67            LDX $67
C304 20 C9 FF         JSR $FFC9
C307 4C E2 C2         JMP HA61A
C30A 68       HA61B   PLA
```

```
C30B E6 65              INC $65
C30D 60                 RTS
C30E A6 68     BSLPIT   LDX ZF
C310 E0 01              CPX #1        ;PRINT SECOND LINE?
C312 F0 04              BEQ HA90      ;YES
C314 4A                 LSR A
C315 4A                 LSR A
C316 4A                 LSR A
C317 4A                 LSR A         ;FORM TABLE INDEX FROM
C318 29 0F     HA90     AND #$0F      ;TOP TO BOTTOM
C31A AA                 TAX
C31B BD 1F C3           LDA HATAB1,X ;GET OTHER SPLIT VALUES
C31E 60                 RTS
C31F 00        HATAB1   .BYT %00000000
C320 03                 .BYT %00000011
C321 0C                 .BYT %00001100
C322 0F                 .BYT %00001111
C323 30                 .BYT %00110000
C324 33                 .BYT %00110011
C325 3C                 .BYT %00111100
C326 3F                 .BYT %00111111
C327 C0                 .BYT %11000000
C328 C3                 .BYT %11000011
C329 CC                 .BYT %11001100
C32A CF                 .BYT %11001111
C32B F0                 .BYT %11110000
C32C F3                 .BYT %11110011
C32D FC                 .BYT %11111100
C32E FF                 .BYT %11111111
C32F 85 B9     OPENCH   STA $B9
C331 09 10              ORA #$10
C333 85 B8              STA $B8
C335 4C C0 FF           JMP $FFC0
```

```
C338 00 01 02 MULH   .BYT0,1,2,3,5,6,7,8,10,11,12,13,15,16
C346 11 12 14        .BYT17,18,20,21,22,23,25,26,27,28,30,31
C352 00 40 80 MULL   .BYT 0,$40,$80,$C0
C356 00       BUFFER .BYT 00
```

Both programs look at the graphic memory ($2000 hex, 8192 decimal); most graphics are found in this area, and the graphic aid programs from THE GRAPHICS BOOK FOR THE COMMODORE 64 uses this range. If the graphics you wish to print out are in another area, all you have to do is change the variable GRMEM into the address required.

Graphic memory is arranged in such a way that the byte at address $2000 is responsible for the eight successive grapihc points in the upper left corner. If a bit is set in this byte, a point appears in this spot, whereas if a bit is low, the point remains invisible (background color). The next byte ($2001) is responsible for row of points below $2000. These horizontal bit representations go up to and including $2007, with the next address ($2008) picking up in the next column, thus:

```
$2000 . . . . . . . . $2008 . . . . . . . .
$2001 . . . . . . . . $2009 . . . . . . . .
$2002 . . . . . . . . $200A . . . . . . . .
$2003 . . . . . . . . $200B . . . . . . . .
$2004 . . . . . . . . $200C . . . . . . . .     etc.
$2005 . . . . . . . . $200D . . . . . . . .
$2006 . . . . . . . . $200E . . . . . . . .
$2007 . . . . . . . . $200F . . . . . . . .
```

Eight bytes of 8 bits comprise a 64-bit block.

Since the printer can only print point sequences vertically, a hardcopy program must isolate and recompose equally significant bits within a byte.

This problem is increased if we are considering an 8-byte character printed by a 7-pin printer: Somehow the eighth bit must be taken up by the next line sequence. A further difficulty lies in the fact that the 25 screen lines (=200 vertical points) is not easily divisible by 7. Once the graphic is printed, we still have 4 points' worth unaccounted-for.

Naturally, these problems don't crop up on an 8-pin printer. Graphics cover all 200 lines with no problem. . .

There is a little-known fact about the 1526: It is capable of printing any character in an 8 X 8 format. The 1526 manual gives a boring example, which prints the Commodore logo 10 times. That wouldn't do any of us any good for everyday usage.

To add to the difficulties, only one custom-designed character is allowed per line.

Here's help!

The 1526 acknowledges a special character which signifies the end-of-line, but gives no linefeed [CHR$(141)], which comes from the values 13+128. CHR$(13) is the value that ends a line with a linefeed, while CHR$(141) is a shifted CHR$(13), which transfers the highest set bit.

This character allows you to define custom characters anew, and print them out on the same line; you will still need some sort of counter to help adjust the printhead bit back to the original print position (whether there is a linefeed or not). To get to the proper place, a corresponding number of spaces must be put in.

Granted, this sort of output is time-consuming, since it's not simply a matter of printing the data. Also, each line routine must figure out a different number of "return spaces". However, since this routine is written in machine language, it will move considerably faster than a counterpart written in BASIC. All the procedures mentioned above are efficiently handled in the program, with an extra feature added to the 1526 version: The graphics are enlarged by a factor of 2. In other words, if you calculate an 8 X 8 matrix with this in mind, you'll get a total of 640 points (8 points * 80 characters) per line. Should you want an exact copy instead, divide the page in half.

Now, however, we come to the BASIC loaders. First we have the program which drives the 7-pin printer.

```
100 FOR I = 49664 TO 49933
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA  32,241,183,134,103, 32, 15,243, 32, 31,243,166
130 DATA 103, 32,201,255,169,255,133, 97,169,  7,133,253
140 DATA 169, 28,133,151,169,  0,141, 13,195,169, 40,141
150 DATA  15,195,162,  4,189,177,194, 32,210,255,202, 16
160 DATA 247,169,  0,133, 99,133,100,173, 13,195,133,101
170 DATA 169,  0,133,254,165, 99,166,100,164,101, 32,182
180 DATA 194,160,  0,177,172,166,254,157, 17,195,230,101
```

```
190 DATA 232,134,254,228,253,208,229,169,  0,160,  7,166
200 DATA 253, 30, 17,195, 42,202, 16,249, 37, 97,  9,128
210 DATA  32,210,255,136, 16,237,165, 99, 24,105,  8,133
220 DATA  99,144,  2,230,100,206, 15,195,208,181,169, 13
230 DATA  32,210,255,173, 13,195, 24,105,  7,141, 13,195
240 DATA 198,151,240,  3, 76, 33,194,169,  4,197,253,240
250 DATA  12,133,253,169,  1,133,151,169, 15,133, 97,208
260 DATA 235,169, 15, 32,210,255, 76,204,255, 80,  0, 16
270 DATA  27,  8,133, 20,134, 21,152, 74, 74, 74,170,189
280 DATA 239,194,133,173,138, 41,  3,170,189,  9,195,133
290 DATA 172,152, 41,  7, 24,101,172,133,172,165, 20, 41
300 DATA 248,133, 99,169, 32,  5,173,133,173, 24,165,172
310 DATA 101, 99,133,172,165,173,101, 21,133,173, 96,  0
320 DATA   1,  2,  3,  5,  6,  7,  8, 10, 11, 12, 13, 15
330 DATA  16, 17, 18, 20, 21, 22, 23, 25, 26, 27, 28, 30
340 DATA  31,  0, 64,128,192,203
350 IF S <> 31355 THEN PRINT "ERROR IN DATA!!" : END
360 PRINT "OK"
```

This is a lot of data to type in, but with the checksum routine at the bottom of each program, errors can be found before they happen. let's talk about the checksum for a moment: This doesn't correct the errors, but rather, it just locates them generally. In other words, we could calculate the checksum by hand, but the C-64 can accomplish the task much faster. A short routine sums the DATA statements. If all is well, the conversion will be made to a machine language program. If not, the program stops, allowing you to go in and fix any mistakes -- changing the number of the checksum doesn't solve the problem, by the way. So, type away (remember to SAVE before RUNning!); if there are mistakes, you'll hear about them.

Now the listing for the 1526 printer.

```
100 FOR I = 49664 TO 50006
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA  32,241,183,134,103, 32, 15,243, 32, 31,243,169
130 DATA   5, 32, 47,195,169,  6, 32, 47,195,162, 22, 32
140 DATA 201,255,169, 20, 32,210,255,169, 13, 32,210,255
150 DATA  32,204,255,160, 25,132,100,169,  2,133,104,169
160 DATA   0,133,101,169, 40,133, 99,169, 25, 56,229,100
170 DATA 170,189, 56,195,  9, 32,133,173,138, 41,  3,170
180 DATA 189, 82,195,133,172,162,  0,160,  8,161,172,153
190 DATA  86,195,230,172,208,  2,230,173,136,208,242,160
200 DATA   8,162,  8, 30, 86,195, 42,202,208,249, 32, 14
210 DATA 195,153, 94,195,136,208,238,160,  8, 32,166,194
220 DATA 160,  4, 32,166,194,198, 99,208,204,198,104,208
230 DATA 170,198,100,208,162,162, 22, 32,201,255,169, 36
240 DATA  32,210,255,169, 13, 32,210,255, 32,204,255,169
250 DATA  22, 32,195,255,169, 25, 32,195,255, 96,169,  0
260 DATA 133,102,162, 21, 32,201,255,152, 72,162,  4,185
270 DATA  94,195, 32,210,255, 32,210,255,  5,102,133,102
280 DATA 136,202,208,239, 32,204,255,165,102,240, 49,166
290 DATA 103, 32,201,255,165,101,240,  9,170,169, 32, 32
300 DATA 210,255,202,208,250,169,254, 32,210,255,104,201
310 DATA   4,208, 10,165, 99,201,  1,208,  4,169, 13,208
320 DATA   2,169,141, 32,210,255, 32,204,255,230,101, 96
330 DATA 165, 99,201,  1,208,  8,166,103, 32,201,255, 76
340 DATA 226,194,104,230,101, 96,166,104,224,  1,240,  4
350 DATA  74, 74, 74, 74, 41, 15,170,189, 31,195, 96,  0
360 DATA   3, 12, 15, 48, 51, 60, 63,192,195,204,207,240
370 DATA 243,252,255,133,185,  9, 16,133,184, 76,192,255
380 DATA   0,  1,  2,  3,  5,  6,  7,  8, 10, 11, 12, 13
390 DATA  15, 16, 17, 18, 20, 21, 22, 23, 25, 26, 27, 28
```

```
400 DATA  30, 31,  0, 64,128,192,  0
410 IF S <> 41908 THEN PRINT "ERROR IN DATA!!" : END
420 PRINT "OK"
```

In conclusion, here are some points about operating the program on the 1526 printer.

After LOADing and starting the program designed for your printer, you can start most other programs. If you want a hardcopy of the screen, start your routine with;

```
SYS 12*4096+2*256,LF.
```

That last parameter (LF) should be the logical file number previously used in the OPEN command (placed under the device address 4).

A similar change within the program can look like this:

```
10
20
30 any graphic program
40 GOSUB 1000:REM HARDCOPY
50 graphic prg
:
:
:
999 END
1000 OPEN1,4
1010 SYS49664,1
1020 CLOSE 1
1030 RETURN
```

If you have typed in the 1526 version (assuming, of course, that you own that printer), and the Hardcopy function still doesn't work, then it's possible that you have one with the exchange ROM installed. This exchange ROM, as mentioned before, generates a slightly different set of operations, making the machine behave like a 1525. This means that all the control characters and secondary addresses acknowledged by the original ROM will be ignored altogether.

On the other hand, printing out graphics is impossible in 1525 mode.

How can we figure out which operating system we have?

There are two ways to do this. For one, you can open the printer, and look at the operating system EPROM (a 28-pin IC with a sticker on it), after removing the metal cover from the EPROM. If the IC bears the inscription "1526 /C", then you definitely have an original ROM, and the above program should work. If the IC reads "325341-06 H", you have a "new" operating system.

If you are the proud owner of the latter, check to see whether the IC U4D (a 6532) has all its pins set in the socket. The U4D indication is on the guide plate of the IC. If pin #16 (counting from upper left) isn't in the socket, rather, it's connected to the guide plate with a piece of wire, then the printer is set in 1525 operating mode. This means, of course, that the 1526 version of the program won't work.

Before you get the tools, there's another way to determine all this without opening up the printer at all.

First, run the printer through a self-test. Turn on the machine while holding down the paper feed key; the first line should print out the abbreviation "REV.", followed by a number. If 1.0 appears, you have an old operating system; if it's 7.0, you have the new version. Now, to determine whether the printer is in 1526 or 1525 mode, send the printer a secondary address of 7 with a printed text. If the printer reacts and prints out text, the 1525 mode is operating (see the U4D connections mentioned above). If the printer does nothing, then you have 1526 mode operating, and the Hardcopy program will indeed work.

Manufacturers being what they are, there are many other operating systems on the market: Here, however, we've limited ourselves to the two most frequently-encountered versions.

With that, we reach the end of the Hardcopy chapter. The next section deals with a theme we haven't touched upon yet--graphics.

5.4 Graphics---With and Without Single-Pin Control

Graphics encompass some of the most interesting (yet some of the most difficult) aspects of programming. One of the best examples is high-resolution graphic programming on the C-64. This allows you to perform near-miracles on the 64 screen. The printer can be programmed in much the same way: A neatly-printed picture can be a great reward for several hours of programming.

We'll spend this chapter viewing methods of high-res printing, step by step. If your printer isn't capable of this particular process, read on a little further--we have a program that will give you amazing pictures with what's available.

First, we'll look at how text and "normal" graphics are made. Then, we'll cover two-dimensional and three-dimensional art. So, let's turn to high-res graphics, the method used to create all of the above.

The simplest method is covered in the program below: This program will work with most printers without any alterations (daisy-wheel owners take note). You may need to make some minor changes to compensate for the idiosyncracies of your printer, such as control characters, etc., but the program should work "as is".

5.4.1 Graphics---Without Single-Pin Control

As mentioned above, this is the simplest type of "high-res" printing. The program will print a sine curve using asterisks, while the paper will feed as usual.

```
10 PI = 3.14159 : OPEN 4,4
20 FOR A = 0 TO 2*PI STEP.25
30 B = SIN(A)
40 C = B*39+40
50 FOR I = 0 TO C:PRINT#4,CHR$(32);:NEXT
60 PRINT#4,"*"
70 NEXT A
80 CLOSE4
```

When you run this example, you may be disappointed in the result. The sine curve is but a vague semblance of what we consider to be a sine curve; but if you experiment a bit with the parameters, you should be able to generate a half-decent-looking sine wave. One way to improve the appearance of the printout is to play with the step-size in line 20.

The effect can be diminshed on some printers, however, by decreasing the distance between lines. One change in distance per line can be made on 7-pin printers by changing line 60 to the following:

```
60 PRINT#4,"*";CHR$(8):PRINT#4,CHR$(15);
```

Usually these printers recognize only two distances: 6 lines per inch in text mode; and 9 lines per inch in graphic mode. In the above example, we've made a switch to graphic mode at the end of the line, so that the printer gives a linefeed in graphic mode, and returns to text mode to print the next asterisk.

The 1526 printer allows you to adjust the line distance in increments of 144 steps per inch. The printer is alerted to this by giving a secondary address of 6, then the specific value. If you wish to lessen the line spacong, you send the printer a value less than 36. For instance, to print lines almost on top of one another, use this command sequence:

```
5 OPEN 6,4,6: PRINT#6,CHR$(24):CLOSE6
```

It's a sad fact that control characters change from manufacturer to manufacturer -- with these come changes in line spacing. If you have one of the printers not mentioned here, check your manual; it should have something to say about these characters.

Another improvement is available on printers that can shift print positions in increments of a single dot, rather than character-by-character. Naturally, you would only find this feature on dot-matrix printers, although a few daisy-wheel printers have proportional-shift capability.

Most 7-pin printers (e.g., Commodore 1515, 1525, and other brands) have this option, although it is seldom used. This allows a printer that normally covers 80 characters per line --in a 7*6 matrix-- to produce a resolution of up to 6*80 (480) positions per line.

A small problem accompanies this feature, though: When dealing with CHR$(X), the printer will only acknowledge X when it is within the range of 0-255, and this could limit our specifying print positions. To circumvent this, the positions must be given in two-byte form. Such a command would be given as CHR$(HIGHBYTE);CHR$(LOWBYTE), following the preparatory codes CHR$(27);CHR$(16). For example, let's print an asterisk at position 200. After OPENing the file, we would send the following:

```
PRINT#1,CHR$(27);CHR$(16);CHR$(0);CHR$(200);"*"
```

To do the same at position 300, the line would look like this:

```
PRINT#1,CHR$(27);CHR$(16);CHR$(1);CHR$(44);"*"
```

In this last combination, the third number becomes 1, and the fourth number is figured from 256 (300-256=44). Now you have 480 print positions to choose from.

The next program is an extension of a function plotter, to which we have added a scaling system. This will allow you to keep track of the range in which the results lie, among other things.

```
100 REM ***********************************
110 REM ***                             ***
120 REM ***     FUNCTION PLOTTER        ***
130 REM ***                             ***
140 REM ***      WITH SCALING           ***
150 REM ***                             ***
160 REM ***      FOR 1515,1525 ETC.     ***
```

```
170 REM ***                                ***
180 REM ***********************************
190 PI=3.1415926
200 GOSUB450  : REM SET BORDER
210 OPEN4,4
220 FORI=1TO80
230 PRINT#4,CHR$(249);
240 NEXT
250 PRINT#4,CHR$(8)
260 FORA=0TO2*PISTEP.1
270 GOSUB530
280 C=B*MY+240
290 HB=INT(C/256)
300 LB=C AND255
310 IFC>=235ANDC<=245THEN390
320 IFC<=240THEN360
330 PRINT#4,CHR$(15)CHR$(27)CHR$(16)CHR$(0)CHR$(240)CHR$
    (245);
340 PRINT#4,CHR$(27)CHR$(16)CHR$(HB)CHR$(LB)"*"CHR$(8)
350 GOTO400
360 PRINT#4,CHR$(15)CHR$(27)CHR$(16)CHR$(HB)CHR$(LB)"*";
370 PRINT#4,CHR$(27)CHR$(16)CHR$(0)CHR$(240)CHR$(245)CHR$(8)
380 GOTO400
390 PRINT#4,CHR$(15)CHR$(27)CHR$(16)CHR$(HB)CHR$(LB)"*"CHR$
    (8)
400 NEXTA
410 PRINT#4,CHR$(15)
420 CLOSE4
430 END
440 REM SCALING
450 FORA=0TO2*PISTEP.1
460 GOSUB530
470 IFB>MATHENMA=B
```

```
480 IFB<MITHENMI=B
490 NEXT
500 IFABS(MI)>MATHENMY=233/ABS(MI):RETURN
510 MY=233/MA:RETURN
520 REM FUNCTION SET IN LINE 530
530 B=SIN(A)
540 RETURN
```

The zero-line character is somewhat extravagant. The point position must dictate whether the point will appear to the right or the left of the zero-line, or whether the point will overlap the zero-line. Thus, the asterisks and the zero-line will appear in their proper sequence. However, if the asterisks cover the zero-line, the zero-line will be covered over. Before printing begins, all function values are figured out. The results will be needed when measurements are determined. This can take a considerable amount of time in BASIC, so have some patience.

If this type of printout doesn't satisfy you, see the chapter on high-res graphics for a Function Plotter which gives amazingly clear results.

To conclude this chapter, we offer a little tip for graphics involving low resolution. Many printers have what is called "compressed mode", which allows you to print 132 characters per line (rather than the normal 80). As mentioned above, the closer the points, the better the graphics. Using this, we can produce still better graphics. To work this idea into the preceding example, you would first have to change the corresponding value by multiplying by 132/80 (=1.65). This tip also applies to the programs to follow, which involve three-dimensional graphics.

You may think, before trying the first program in the next chapter, three-dimensional graphics made of asterisks don't work too well. You may be in for a small surprise.

First, though, the program.

```
100 DIMA$(80)
110 OPEN4,4
120 FORX=1TO72
130 : H=0:L=120
140 : FORI=1TO80:A$(I)=" ":NEXT
150 : FORY=1TO48
160 : : XT=(X-Y)*2
170 : : IFXT<1THEN290
180 : : IFXT>48THEN280
190 : : C=SQR((XT-24)*(XT-24)+(Y-24)*(Y-24))+1
200 : : W=XT+EXP(-.01*C*C)*45+3
210 : : IFW>HTHEN240
220 : : IFW<LTHEN260
230 : : GOTO280
240 : : H=W
250 : : IFW>LTHEN270
260 : : L=W
270 : : A$(W)="*"
280 : NEXTY
290 : FORI=1TO80:PRINT#4,A$(I);:NEXT
300 NEXTX
```

Although this program is quite short, the logic of it is far from simple. We're not going through the entire routine line-by-line; instead, we'll offer you some ideas for changes and development. Any suggestions, of course, can also be applied to any programs you've seen up until now.

Producing a longer line (e.g., 132 characters per line) involves multiplication by 1.65 (see above). This especially applies to the DIM statement at the beginning of the program. There are two exceptions; the 2 in line 160, and the 0.01 in line 200.

You can also try the above mentioned measures to decrease distance between lines.

To set up another upper surface, you can change the function in line 200. Give this a try:

```
200 W=XT-EXP(-.01*C*C)*45=25
```

These tricks and tips naturally won't give you the same results as a high-res, single-pin printout. In fact, for all the reductions we've suggested, you would be better off working on single-pin control, which is the subject of our next section.

5.4.2 Graphics -- With Single-Pin Control

In the last chapter, we tried to produce graphics on printers that had no special abilities in that area. This chapter is for the printers with single-pin capability.

If we drastically raise the number of controllable points, the graphic resolution increases proportionately. You've probably seen many high-resolution printouts (a fine example is the graphic depicting Winston Churchill).

How do we program such graphics? After reading this chapter, you, too, will be able to do such artwork, right up to three-dimensional art (assuming that your printer allows for single-pin control).

The examples listed here are intended for the VIC-1515, 1525, and MPS 801 printers, as well as Epson machines. Owners of other brands (e.g. ITOH) shouldn't throw in the towel just yet; conversions should be relatively easy with the help of your printer manual, and our explanations.

First, let's look at the fundamentals of single-pin control. Most printer manuals cover this subject only nominally, often leaving the user with more questions than answers. Yet, this feature is what persuades most folks to buy such printers.

In order to operate a printer in graphic mode, you'll need to know a few basics on how a computer works. The VIC-20 and C-64 are "8-bit" machines. As you've probably heard elsewhere, the bit is the smallest information unit available. A bit can be set in one of two ways; off or on.

This indicates false and true (or no and yes, if you prefer). Technicians prefer using the terms High and Low, which corresponds to measuring a certain bit line with a voltmeter. One other designation for bit condition is 1 for high, and 0 for low.

The latter (1/0) has the advantage that the value of the bit can be computed using the numbers. This is exactly what must be done when designing graphic programs.

So, now we have a bit with only two conditions possible. You might well ask, why only two positions? Put simply, with a bit, you can only have "Yes" or "No"--there is no "Maybe" amongst bits. We can have a total of four conbinations between two bits:

BIT 1	BIT 2
NO	NO
NO	YES
YES	NO
YES	YES

Adding another bit would give us 8 possible combinations, a fourth would give us 16, a fifth bit, 32. We won't show you all of them here--try them on your own.

This fast increase in bits is a geometrical one. You may know the old math puzzle: First day of the month, someone gives you a penny, with a promise to double the amount given you each day. Day 2 brings two cents, day 3, four cents, day 4, eight cents, and so on, until by the end of the month, you have around $10,000,000 (or $20,000,000, if you prefer a 31-day month).

By taking a good look at these bits, you can determine exactly which combinations are in effect. All you need to do is multiply 2 by itself the number of times equal to the number of bits. For example, two bits gives you 2*2, four bits equal 2*2*2*2, etc.

Now, you probably know that such expressions can be written a bit more concisely. That is, the above two examples can be expressed using exponentiation, which uses a symbol called a caret [^], represented by an up-arrow in Commodore BASIC. So, 2*2 = 2^2, 2*2*2*2 = 2^4, and so on.

Since your computer is an 8-bit machine, we can set or unset any combination of 2^8, or 256, bits, which gives us a value of 0 to 255. Yes, the zero counts as a value as well; we get this value, of course, when none of the eight bits are set.

We'll now number our 8 bits, just as we did earlier with our oiginal two bits. In this case, we use numbers from 0 to 7 to represent our bits. Granted, this number system may seem more difficult to grasp than, say, 1 to 8, but there is a very important reason for this unusual system. Let's take another look at our chart from the last page or so, and change the bits accordingly:

BIT 1	BIT 0	
NO	NO	= 0
NO	YES	= 1
YES	NO	= 2
YES	YES	= 3

It's important to remember that every set bit has a specific value. So, Bit 0 has a value of 1 if set, and Bit 2 has a set value of 2. If both bits are on, then we get 3.

Next, we'll extend this table to include a bit with the value of 4; please note that the bit values increase exponentially. Such a bit would amount to 2^2, or 4. Increasing the number of bits to a total of 8, the largest would equal 2^7 = 128 (remember, 0-7 bits). The following table shows the result:

```
BIT  7    6    5    4    3    2    1    0
     :    :    :    :    :    :    :    :
     :    :    :    :    :    :    :    :::  2^0  =    1
     :    :    :    :    :    :    ::::::::  2^1  =    2
     :    :    :    :    :    :::::::::::::  2^2  =    4
     :    :    :    :    ::::::::::::::::::  2^3  =    8
     :    :    :    :::::::::::::::::::::::  2^4  =   16
     :    :    ::::::::::::::::::::::::::::  2^5  =   32
     :    :::::::::::::::::::::::::::::::::  2^6  =   64
     ::::::::::::::::::::::::::::::::::::::  2^7  =  128

                                             -----------
                                                     255
```

What does all this have to do with single-pin control? Well, using the decimal values in a certain way, we can turn any bit on or off, and also print out any sequence of bits that we'd like. The only thing that we must caution you about, is that this point sequence only applies to the VIC-1515 printers and the like. Epsons and some other printers use the opposite sequence: The lowermost needle has a value of 1, while the uppermost equals 128.

If these point values are clear to you, we'll try out some graphics. Rather than write a program, let's start with a printer that exists only in theory: This printer has a lot of features, and all we need do is call them out. Just for the sake of argument, let's say we want to draw a circle on the printer, and that the uppermost point sequence equals 1. Our circle would look something like this:

```
   ***          1
  *   *         2
 *     *        4
 *     *        8
 *     *       16
  *   *        32
   ***         64
```

In order to simplify the task, the point values will be stored in specific lines. Now, we'll have to add up the values of every line. To keep things symmetrical, calculate the values for the first three lines, and repeat accordinly.

The first line comes to 4+8+16=28. The second line contains 2+32=34, and the third line gives the result 1+64=65. Repeat the 65 twice more, then use 34 again, and finally, 28.

After these values have been figured out, then what? Well, our next task is to switch the printer in graphic mode, which, in the case of our imaginary printer, is done with the value 128, within this sequence:

```
PRINT#1,CHR$(128)
```

Now the numbers we have from above can be sent to the printer by way of PRINT commands. Storing these figures in DATA statements is a convenient method of holding them for transmission.

Let's look at the logic for such a program:

```
10    OPEN 4,4
20    PRINT#4,CHR$(128)    :REM SHOULD ACTIVATE GRAPHIC MODE
30    FORI=1TO7            :REM AMOUNT OF DATA TO BE SENT
40    READ A               :REM GET INDIVIDUAL BYTE
50    PRINT#4,CHR$(A);
60    NEXT I
70    PRINT#4,CHR$(128)    :REM SWITCH OFF GRAPHIC MODE
80    CLOSE4
90    DATA 28,34,65,65,65,34,28
```

The program above can only work if our imaginary printer can control 7 pins in graphic mode. If it happens to be an 8-pin printer, then the number 128 is useless for switching graphic mode on and off. So, we must take another route, which will still involve the above program, and still use 128 for graphic mode. However, we will also send the number of graphic bytes involved, which means that line 20 will now look something like this:

```
20    PRINT#4,CHR$(128);CHR$(7);
```

This will give you some idea of how modifications are performed for different printers.

But, let us turn to the graphic modes of real printers. First, consider the 7-pin printer: Graphic control is turned over to us by two control characters -- code 8 switches on graphics, while 15 turns graphics off. This graphic switching poses a modest problem, though; the models mentioned above use graphic codes from 0 to 127. So, code 15 is interpreted as a graphic byte, AND is printed out in its bit pattern. Most printer manufacturers have noted this problem, and let us circumvent it by adding 128 to the value. Therefore, the graphic bytes lie in the range of 128-255, and we can switch it back to text mode using code 15.

Code 8 performs still another printer function: It increases the number of printed lines from 6 lines per inch to 9 lines per inch. This step puts us in a position to print smooth graphics (unbroken lines). If you use an Epson on your VIC or 64, with or without a special interface, you have a very powerful form of single-pin control at your fingertips; this particular printer can have up to 9 pins at hand, but if you're trying to print 7-pin graphics, this can be a problem. Each graphic byte will send out and print 8 bits, while the last pin will be ignored.

One comment on this last item: The Epson FX-80 recognizes 9-pin graphic mode, which must always be transmitted using two graphic bytes. The highest model in the Epson line (the LQ 1500) is equipped with 24 needles which are controlled by a special graphic mode. This means that you can produce graphics which approach photographic quality.

The values which apply to the Epson machines are exactly reversed for the Commodore printers,i.e., the lowest needle is 1, and the highest is 128. The numbers of the data bytes must also be reworked, this doesn't pose much of a problem.

Let's return to the first short program; a function plotter similar to the plotter in the previous chapter. This version of the program has a tremendous advantage over the other -- the result can be seen quickly, and the program moves fairly quickly. There are two function plotters here: One for the Commodore printers, and one for Epson models. Both will be documented in the next few pages, so that you can easily adapt them for your own printer (e.g., Seikosha or Itoh).

The program below contains no fancy routines, no "cute" user prompts. We've deliberately made this program simple, so that the routines are easily readable for you.

```
10 REM
20 REM FUNCTION PLOTTER
30 REM
40 REM IN SINGLE-POINT CONTROL
50 REM
60 REM FOR COMMODORE  7-PIN PRINTERS
70 REM
80 REM
90 PI = 3.14159
100 DEFFNA(X)=SIN(X)
110 INPUT"HORIZONTAL RESOLUTION";X%
120 IFX%<1ORX%>479THEN110
130 INPUT"VERTICAL RESOLUTION";Y%
140 IFY%<1THEN130
150 INPUT"RANGE OF DEFINITION";A,B
160 A=A*PI:B=B*PI:IFA>=BTHEN150
170 INPUT"RANGE OF VALUES";C,D
180 IFC>=DTHEN170
190 DIMY%(X%)
200 VF=Y%/(D-C)
210 HF=(B-A)/X%
220 H=1
230 FORI=6TO0STEP-1
240 Y$(I)=CHR$(H+128)
```

```
250 H=H+H
260 NEXT
270 FORI=1TOX%
280 Y=FNA(A+HF*I)
290 Y%(I)=(Y-C)*VF+.5
300 NEXT
310 OPEN1,4
320 PRINT#1,CHR$(8);
330 A$=CHR$(128)
340 FORH=Y%-6TO0STEP-7
350 REM
360 FORI=1TOX%
370 Y=Y%(I)-H
380 IFY<7ANDY>=0THENPRINT#1,Y$(Y);:NEXT:GOTO410
390 PRINT#1,A$;
400 NEXT
410 PRINT#1
420 NEXT
430 PRINT#1,CHR$(15):CLOSE1
READY.
```

The lines below are the ones you'll have to change to run the program on an Epson printer. This version is for an Epson running an interface from the serial port, so OPEN commands have been changed accordingly.

```
20 REM  FUNCTION PLOTTER
30 REM
60 REM   FOR EPSON MX/RX/FX
120 IFX%<1 OR X%>1152 THEN 110
230 FORI=0TO7
310 OPEN1,4,4
320 PRINT#1,CHR$(27)"{SHA}"CHR$(8);
330 A$=CHR$(0)
340 FORH=Y%-7TO0STEP-8
350 PRINT#1,CHR$(27)"{SHL}"CHR$(X%AND255)CHR$(X%/256);
380 IFY<8ANDY>=0THENPRINT#1,Y$(Y);:NEXT:GOTO410
430 CLOSE1

READY.
```

Line 100 defines the function of the graphic output. You can define this function as you see fit--experiment.

Next, you are asked for the most important parameters. The horizontal resolution refers to the width of the graphic on paper. As a quick test, you can input a smaller value (e.g., 100). This will give you a narrow graphic (which takes less time to print out). The maximum is 479, the number of possible graphic points per line.

There is no limit to the vertical resolution, but if you give a value much higher than the vertical resolution, your graphic could come out quite distorted.

The definition range VALUE decides to what extent the vertical direction should travel. If, for example, a printout does not show the edges of, say, a sine curve graphic, the definition range should be larger, so that all of the picture will fit onto the paper.

After inputting these values, variables HF and VF are CALCULATED, which are necessary for determining the locations of the plotted points. Next, the possible bit patterns are calculated and stored in array d$(I) (in lines 230-260). The highest bit is set for 7-pin printers (+128), since as mentioned above, the graphic data for these printers lies in the range from 128 to 255. Here is where the big differences exist between such machines and the Epson models. For one thing, the pins are numbered from bottom to top on an Epson, and there are 8 pins instead of 7. So, the Commodore version of the program has a sixfold loop in lines 230-260, while the Epson version executes a sevenfold loop. Another distinction between the two programs: The sequence of pins counts in the loop from 6 to 0 for Commodore printers, and from 0 to 7 for the other machine.

If you are adapting this program for, say, an Itoh machine, keep in mind that the needle sequence is similar to Commodore printers (highest=1, lowest=128). The loop also runs in reverse, so should execute from 7 to 0. The functions for all values will be run through, and stored in Y% (lines 270-300).

Now for the printout, beginning with the opening of the printer channel. The OPEN command will have to be changed to concur with your individual model of printer.

The printout occurs in two loops. First loop is the line loop, which executes the amount of vertical resolution in Y% -6 in steps of 7 (or 7 pins -6, again dependent on printer model). Line 340 has the value which you'll have to change for an 8-pin model. The second loop runs from 1 to the number given for horizontal resolution. The variable I is used in this loop in conjunction with the corresponding value for the X-axis. The line loop draws its value from this function value. If the result lies in the range from 0 to 6 (0-7 for Epson), then this number is used to set up the proper point distance between lines. These point patterns are predefined in their proper sequence in array Y$. In all other cases, only a value of 128 will be sent to the printer, which produces a blank point sequence. (NOTE: in the case of an Epson, CHR$(0) is sent.

So, hopefully you've got an understanding of your first attempt at single-pin graphics; we know it's not easy to follow, but you can't take these graphics any farther without knowing what's going on. You'll find a knowledge of the basics especially important when working on three-dimensional figures, which involves some very difficult program logic to yield a good likeness.

This chapter will help you in adapting the following programs for other printers, since not everyone owns a Commodore printer.

5.4.3 Three-Dimensional HIRES Graphics

One true challenge to a programmer is preparing three-dimensional objects using a printer. Now that the basics of output are taken care of, we can set to work learning how to do this sort of graphic. Be forewarned, though; along with the programming, some knowledge of mathematics is necessary.

The obvious problem in 3-D printing lies in the fact that the paper is only two-dimensional. The third dimension, depth, can only be represented by shades of black and grey. So, convincing three-dimensional graphics are possible only by using single-pin control.

We've developed two programs which will produce neat pictures on your printer. Firstly, we have a program which will illustrate a ball for you. The result looks remarkably like it's hand-drawn if viewed from a distance.

The second program is a sort of three-dimensional function plotter. With this program, you can make surfaces of a flexible nature, such as waves; this means that you can make designs such as, say, a hat in perspective.

On to the ball program: Although primarily designed for drawing a sphere, this can be changed for other figures as needed.

```
100 REM ****************************
110 REM **                        **
120 REM **     3-D  BALL          **
130 REM **                        **
140 REM ****************************
150 PRINTCHR$(147)
160 PRINT"BALL"
170 INPUT "DIAMETER IN POINTS (480 MAX.)";D
180 HD=D/2:FF=D/10
190 INPUT "PRINTER OR DISK";D$
200 IFD$="P"THENOPEN1,4:PRINT#1:PRINT#1,CHR$(8);
210 IFD$="D"THENOPEN1,8,4,"O:BALL DATA,S,W"
220 SK=1/10000:R=1
230 FORI=0TO6
240 MK(I)=R
250 R=R+R
260 NEXT
270 R=0
280 DIMWE%(D)
290 FORZ=-1TO1STEP1/HD
300 ZZ=ZZ+1
310 PRINTZZ;
320 R=R+1
330 MY=SQR(1-Z*Z)-SK
340 Y=-MY
350 X=SQR(ABS(1-Y*Y-Z*Z))+SK
360 A=SQR(ABS((1+Y*Y/X/X)*(1+Z*Z/X/X)))
370 S=1/A/FF
380 IFS<0THENS=0
390 Y=Y+S
400 IFY>MYTHENWE%(HD+MY*HD)=WE%(HD+MY*HD)ORMK(R-1):GOTO430
410 WE%(HD+Y*HD)=WE%(HD+Y*HD)ORMK(R-1)
420 GOTO350
430 IFR>6THENGOSUB490
440 NEXTZ
450 IFR>0THENGOSUB490
460 PRINT#1,CHR$(15)
470 CLOSE1
480 END
490 FORQ=1TOD
500 PRINT#1,CHR$(WE%(Q)+128);
510 WE%(Q)=0
520 NEXT:PRINT#1:R=0
530 RETURN

READY.
```

Once the program is started, you will be asked to input the diameter of the ball to be printed; your maximum diameter will be 480 (the number of points per line). Remember that the smaller the diameter chosen, the faster the picture will be completed. If you do want a sphere with the maximum allowable diameter, then you'd better not be needing your computer for the next few hours. The best thing to do is leave your system running overnight -- the next morning you'll have your printout. Of course, this idea is not everyone's cup of tea, either, since printers tend to make a great deal of noise. An alternative to this is to save the data on diskette (which takes considerably less time than printing), and print it out at a later date. This has the advantage of allowing you to print out a number of identical spheres if you wish, without regenerating the program every time.

You'll need the following program to print out your data.

```
100 OPEN4,4
110 PRINT#4,CHR$(8)
120 OPEN1,8,4,"BALL DATA,S,R"
130 GET#1,A$:IFST=64THEN150
140 PRINT#4,A$;:GOTO130
150 PRINT#4,CHR$(15)
160 CLOSE4
170 CLOSE1

READY.
```

Let's look again at the main program, though. After you input the diameter and the output device, the necessary values are computed from the diameter, such as the radius, the location of the middlepoint, and the variable FF (fill factor), which will be vital in determining how many points will be printed and where. You should try experimenting with this last variable; see how large a value is needed for a perceptible change.

Next, the printer and disk drive channels are opened, and the variables SK (sehr klein, or very small in English) and R are initialized. SK is an infinitesimal number which prevents any DIVISION BY ZERO-ERROR in line 360. The variable R, meanwhile, refers to the pin sequence, and will be needed at the beginning of the program to store the proper bit patterns in the array MK.

The proper calculations are performed in the FOR-NEXT loop in lines 290-440. Within that set of lines, 350 and 360 are held responsible for the shape of the drawing, which check variable R for needle sequence. If all points are figured for all 7 pins, then the calculations are sent to the output device, and the next row of points are worked out.

There is one variable of particular importance to adaptation to other printers; the variable HD. It controls the number in the point sequence in the FOR-NEXT loop.

Now, in the same way that some printers will take 100 X 100 points and produce a rectangle instead of a square (which is what you'd logically expect), these same printers will probably give you an egg instead of a sphere. Using a proper proportion of vertical or horizontal resolution, you can adjust the egg to stand on its side, or its end, or make the sphere come out just right. All this is controlled by the variable HD: The smaller HD is, the "flatter" the sphere, while a higher value gives an egg standing on end.

Other modifications can be made using the control codes for switching on graphics. In order to calculate graphic data for an 8-pin printer, the necessary information must be put into variable D (diameter). This varible stores the number of points in horizontal resolution.

Once the novelty of drawing spheres has worn off, you'll love the next program, which reproduces any surface.

The program below is the final one in our little set for printer owners.

This program lets you make printouts of the most complicated figures. As a matter of fact, it will allow you to design a series of waves of different length and height. The possibilities of this program are endless.

As with all BASIC programs requiring a good deal of repeated calculations, this program is quite slow. All of the programs in this chapter, fortunately, are easily compiled (try BASIC 64 from Abacus Software); compiling will greatly cut execution time.

Under normal conditions, an entire surface can take up to two hours, so you might want to save your data to diskette.

This program, like the preceding ones, operates with all 7-pin printers. Adaptations to other machines should pose no great problem by now.

```
100 PI=3.14159
110 DIML%(480),NS(6),HM(4),XP(4),YP(4),WX(4),WY(4)
120 X=1
130 FORI=0TO6
140 NS(I)=X
150 X=X+X
160 NEXT
170 PRINTCHR$(147)
180 PRINT:PRINT"        3D-PLOTTER"
190 FORI=0TO480
200 L%(I)=128
210 NEXT
220 I=0
230 PRINT:PRINT"   BEST VALUE OR RANGE"
240 PRINT"    WILL BE ENCLOSED IN PARENTHESES"
250 PRINT:PRINT"        DO YOU WANT TO OUTPUT TO"
260 PRINT:PRINT"  THE (D)ISK DRIVE OR THE (P)RINTER?"
270 INPUT"              ";A$
280 IFA$="D"THENOPEN1,8,4,"3D-DATA,S,W" :GOTO310
290 IFA$="P"THENOPEN 1,4:GOTO310
300 GOTO 270
310 PRINT#1:PRINT#1,CHR$(8);
320 PRINT:PRINT"    PLEASE GIVE PARAMETERS"
330 PRINT:PRINT
340 INPUT"VERTICAL DOTS";A$:LN%=VAL(A$)
350 MF%=(LN%*2/3):IFLN%<1ORLN%>360THEN340
360 PRINT:INPUT"NUMBER OF EDGES (1-4)";A$:NP%=VAL(A$)
370 IF NP%<1ORNP%>4THEN360
380 PRINT:INPUT"LEFT BORDER        (150)";A$:LM%=VAL(A$)
390 PRINTCHR$(147):GOTO420
400 FOR I=1TONP%
410 PRINT CHR$(19)
420 PRINT:PRINT"   POINT NO."I:PRINT"   X POSITION (0 TO"M
F%")";
430 IFFF=1THEN INPUT A$:XP(I)=VAL(A$)
440 IFFF=0THENPRINT
450 PRINT:PRINT"   POINT NO."I:PRINT"   Y POSITION (0 TO"M
F%")";
460 IFFF=1THEN INPUT A$:YP(I)=VAL(A$):M%=MF%/7
470 IFFF=0THENPRINT
480 PRINT:PRINT"   POINT NO."I:PRINT"   X WAVELENGTH    ("
M%")";
490 IFFF=1THEN INPUT A$:WX(I)=2*PI/VAL(A$)
500 IFFF=0THENPRINT
510 PRINT:PRINT"   POINT NO."I:PRINT"   Y WAVELENGTH    ("
M%")";
```

```
520 IFFF=1THEN INPUT A$:WY(I)=2*PI/VAL(A$):M%=LM%+XP(I)-20
530 IFFF=0THENPRINT
540 PRINT:PRINT"   POINT NO."I:PRINT"   HEIGHT     (20 TO"
M%")";
550 IFFF=1THEN INPUTA$:HM(I)=VAL(A$)
560 IFFF=0THENFF=1:GOTO400
570 NEXTI:PRINT
580 GN=1/10000:K=0:MM%=480
590 FORI=1TOLN%*1.25:PRINTI;
600 IL=2000:IH=-IL
610 FORJ=1TOMF%
620 IFJ>ITHENJ=MF%:GOTO740
630 IX=(I-J)*2:IFIX>MF%THEN740
640 X=IX:Y=J:Z=IX+LM%
650 FORN=1TONP%
660 X1=(X-XP(N))*WX(N)+GN
670 Y1=(Y-YP(N))*WY(N)+GN
680 W=SQR(X1*X1+Y1*Y1)+GN:Z=Z-SIN( W)/W*HM(N)
690 NEXTN
700 IZ=Z
710 IFIZ<0ORIZ>MM%GOTO740
720 IFIZ>IHTHENIH=IZ:L%(IZ)=L%(IZ)ORNS(K)
730 IFIZ<ILTHENIL=IZ:L%(IZ)=L%(IZ)ORNS(K)
740 NEXTJ
750 K=K+1
760 IFK>6THENGOSUB820
770 NEXTI
780 IFK>0THEN  GOSUB820
790 GOSUB820
800 PRINT#1,CHR$(15):CLOSE1
810 END
820 FORJ=1TOMM%
830 PRINT#1,CHR$(L%(J));:L%(J)=128
840 NEXTJ:K=0:PRINT:PRINT#1
850 RETURN

READY.
```

You have the option of choosing output -- either straight to the printer, or first to the disk drive. Should you store your data on diskette, you can print it out later using the short loader program which we gave you with the ball program. One reminder, though: Remember to put in the proper file name, or else you'll get a FILE NOT FOUND message.

The program will ask you for some parameters. We suggest that for testing purposes, you choose the most convenient values or ranges.

First, you'll be asked for the number of vertical print points. A smaller value (50 - 80 points) will do for a demonstration. Once you see that the program works well, then you can increase the number.

The next input asks for the number of edges. An edge in this case is the middle point of a wavy line, similar to when you throw a stone into a pond (producing ripples).

After this input, a mask appears on the screen, into which the parameters for positioning should be given (i.e., wavelength).

Maybe you've been wondering why the wavelength is given in X and Y registers. Well, this lets you create oval waveforms, rather than just circular systems.

Once all input has been given, the major calculations take place. The factor 1.25 in line 590 can vary from printer to printer, as can the proportion of horizontal and vertical print positions. In the most extreme case, the wave can end up as an oval, despite equal numbers for X and Y. The best you can do under these circumstances is experiment with smaller and larger numbers until you find the most workable combination.

Lines 620-670 calculate the points, while 690-720 decide on the visibility of the point (whether to print it or not). This corresponds with the bit pattern proper from array NS. The printout routine in lines 810-830 is almost identical to the sphere printout routine.

So, we come to the end of this chapter. With these programs and a little imagination you can come up with practically any graphic you wish. If you keep getting the infamous SYNTAX ERROR in your first attempts, have courage: Programming gets easier with experience.

6.0 PRINTER PRINCIPLES

Looking at an old typewriter and a high-tech printer will tell you that printing has come a long way in the last few years. After the introduction of electronic data processing equipment, which offered calculations at lightning speed, it was discovered that these same calculations didn't make it to paper quite so quickly. It's hard to believe (or admit) that paper is still the most important data medium. Look at it this way: When was the last time you made handwritten notes on a diskette?

What happens when remote printing is required? The most popular answer lies in the good old teletype. Many computer freaks say that these days, the teletypes are easily affordable, but with print speeds in upwards of six characters per second (whoosh), those old machines are definitely not prize winners.

Mainframe computers have many new, fast printing procedures. Since the firms that own such machines have lots of space to work with, it's possible to find printers the size of closets in existence. Don't let the size fool you -- these printers can produce 600 lines per minute (!). These printers usually reside in a central spot in the room, so the employees can easily get their printouts.

The developments which benefit the home computer user today began with the mainframe machines: Eventually, a single computer had a number of screens and terminals at individual work stations. This meant that with a terminal at every writing desk, only one printer was needed for the entire company (or floor), although this is not very efficient. It was with this thought that small yet efficient printers came into being.

In the meantime, computers began appearing more and more in the home, while strides in technology succeeded in reducing costs to an alarming degree.

The printers found in most homes can be divided into three general types, corresponding with their methods of printing and operation. We have listed these three types below.

6.1 The Daisy-Wheel Printer

The operating principles behind this device were, of course, taken from the universally-known ball-type typewriter.

A vital prerequisite to working with standard printer paper ("endless" sheets, connected with perforations) is the assurance that the paper doesn't move horizontally. Most ball-head typewriters are so complicated that the ball remains stationary, while the roller is on a carriage, moving the paper across the printhead. In a few exceptional cases, the printball is moveable, but horizontal movement of the ball is a very expensive proposition.

As if this weren't enough, the type balls themselves are quite expensive, and having a number of typefaces available can run into a great deal of money.

Therefore, many thought about solving these difficulties. Let's see, we need a simple and inexpensive counterpart to the ball-head, which can be easily moved across the paper, to prevent the paper itself FROM MOVING....

The daisy-wheel was born. The principle is ingeniously simple: You take a disk with a circumference as large as needed to fit the necessary characters, numbers and special characters; fit it with its center connected to a small motor; drive the motor until it reaches the desired character; and press the character against the paper. Some toy typewriters work on this principle, only you have to turn the wheel by hand.

Naturally, the entire disk isn't pushed against the paper -- only the segment containing the character we want. In order to accomplish this, the disk is cut into tongues radiating from the center. At the highest point of the print wheel, you'll see a tiny electrically propelled hammer, which forces the characters into the ribbon, then the paper.

The character is turned into place by a step-motor attached to the axis of the daisy-wheel.

Let's digress for a moment, to the subject of stepping-motors: A motor runs the moment that current is turned on. When the current is shut off, a normal motor will continue to run for a few revolutions before coming to a standstill. This sort of motor just isn't suited for powering a daisy-wheel, since we want the motor to stop at the point that we specify. A stepping-motor, on the other hand, starts and stops at predictable points.

This entire assembly is mounted on a sled of sorts, which is likewise moved by a stepping-motor, so that the character is printed as the exact place on the horizontal axis.

The print quality of such a machine is first-class. Changing typefaces (fonts) is a simple matter of changing the daisy-wheel.

Daisy-wheel movement takes some time; therefore, don't expect incredible amounts of speed from this type of printer (figure on 15-80 characters per second). Price is more-or-less in proportion to the speed of the printer. The price can be as low as $350, and as high as -- well, use your imagination.

6.2 The Parallel Matrix Printer

In striving for high-speed printing at a low price, new paths had to be cleared. The typing principle, while very pretty, was unsuitable for moving quickly. So, another system altogether had to be developed.

The basic requirement for this new system was this: Print as many characters as possible, with as little movement as possible. Additionally, the characters should be made up of a minimum of interchangeable segments, rather than using a specific letter-for-letter setup.

Since the printhead moved anyway, the head could theoretically be rigged up vertically to produce these character segments, with the sections printed as the head moved across the paper.

This layout reduced the segment to a set of vertical points, which meant that anything from a dash on down could be printed out.

A point can be drawn by forcing a pin against a typewriter ribbon and paper. These pins have little problems with inertia (they are small and light), and can therefore move swiftly.

The advantage to this principle lies right before you: You aren't limited to one fixed character set. Rather, you can call up virtually any type of character you wish, just as you were working with single-pin high-resolution graphics in the last chapter.

The character set is built into the operating system of the printer. You can see how it is designed by peeking ahead to Section 7.3. The character set simply takes the characters sent from the computer, compares them to the printer's values, and drives the needles onto paper accordingly. The end result is the printed character.

The number of print needles varies -- they average 7 to 9 on a printhead. The more needles, the finer the printing.

The print speed can vary from 100-400 characters per second. This speed drops to 20-40 characters per second in "letter-quality" mode, in which the printhead makes two passes in printing a line, shifting by a tiny fraction on the second pass. Thus, the letters are a bit thicker, and more distinct. With some models, you have to look very closely to see the difference between daisy-wheel print and "letter-quality" dot-matrix printing.

Prices can go from $150.00 to $700.00.

This type of machine is best suited for "all-purpose" tasks, with an occasional piece of correspondence.

6.3 The Serial Matrix Printer

This term was unknown until quite recently. It applies to those who own a computer in the under-$150.00 bracket (such as the VIC-20 and, at time of writing, the C-64). Experience shows that most peripheral devices (including printers) tend to cost a good deal more than the computer itself. Resourceful technicians have succeeded in reducing the print mechanism to the barest minimum, thus driving the price of a printer equal to or below the price of the computer.

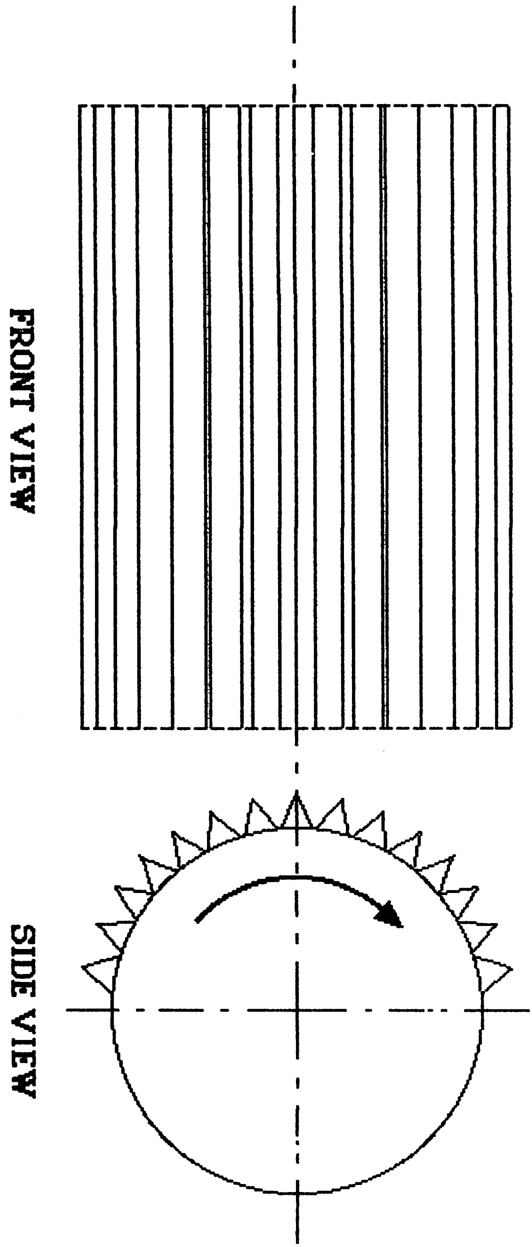
How is this possible? Simple; the principles used in Chapter 6.2 have been cut down, by decreasing the number of necessary segments in the print head. In fact, the number of segments has been cut down to ONE needle in the head.

This single needle poses a problem, though -- only single-line printing is possible, unless we have some way of moving the paper and/or head vertically as well as horizontally. You couldn't possibly convince anyone that this method is cheaper than most.

Neither head nor paper move. ?????????????????????

The answer to this problem is incredibly simple. Describing this principle verbally just won't work; please look at the illustration on the next page for a cross-section of the system.

Diagram of Serial Dot Matrix Printer



Up until now, the two methods of paper transport have been friction-feed (rubber roller) and tractor-feed (pins fitting into perforations on the paper borders). Here, however, we have a "spiked roller" to work with. This spiked roller is coupled closely with the head-movement mechanism. The head, of course, has its own carriage. The print hammer is no longer designed as a needle, but rather as a small block. Thus, when a point is left on the paper, both hammer and roller move in their respective directions.

While the printhead moves from left to right, the roller moves from bottom to top. It's clear what occurs: The hammer has to repeatedly hit the paper to make one column, as the roller turns minutely with each strike of the hammer. If the head moves slightly to the side, the line will be at an angle.

The operating system of this printer knows when the hammer should be moved, and when the roller should move. It coordinates movements using a "timedisk", in which a slot is moved past a tiny lightbox. A light impulse fires the hammer, and shifts the roller by one point-thickness.

The speed of this printer isn't exactly thrilling, since the print hammer has to move much more often than parallel matrix printheads, to accomplish the same job. Figure on 50-80 characters per second.

This type of printer is an excellent buy for those who might only do occasional printing.

{This page left blank intentionally}

7.0 THE OPERATING SYSTEM OF THE MPS 801

Much has been written about operating systems of every conceivable kind, although to our knowledge, printer operating systems have yet to be documented. It is our opinion that this book would hardly be a well-rounded piece of writing if something were not included for both advanced hobbyists and laity alike. Perhaps "operating system" is a pompous-sounding term to use for a printer; but as you'll soon see, the simpler a printer's design, the more powerful its operating system.

The MPS 801 is a prime example of minimum expense in design. In fact, it's a printer that operates on the principle discussed in Chapter 6.3 (i.e., the MPS 801 is a serial matrix printer). We chose the MPS 801 because it's relatively new on the market, and because it will give you a good idea as to how the serial printer works.

It wasn't easy to gather information on this operating system: To begin with, we had to write a disassembler for the 8039 microprocessor. Next came the problems with the different command set (we're accustomed to 6502 or Z80 machine language, not 8039).

Almost as troublesome was the commentary, for which we'll offer apologies now. Now we understand this processor. The 8039 has some similarities to the 6510 microprocessor, but the idiosyncracies of the former caused us a lot of trouble while learning about it.

Before going on to the m/l listing, let's explore some hardware, and the block diagram soon to come.

7.1 The 8039 Microprocessor

The 8039 belongs to the class of microprocessor known as "single-chip",i.e., it is self-contained -- it has virtually all the information it needs for running the hardware, without benefit of supplementary chips, or I/O ports. All that it does rely on is external program memory to tell it what to print; the chip itself runs the motors and carriage.

Here are a few specifications:

* 128 bytes RAM
* Counter/Timer
* 16 I/O lines
* 2 separate test inputs

Mind you, this is all inside this one chip.

Now, for all its compactness, there are some problems. One is that only 2k of program memory can be addressed at a time. A further 2k can be switched on, but you still only have 4k maximum. Another limitation is that the stack only has eight levels, so subprogramming is limited to a small area, but most of the stack is seldom used anyway.

Now for some commands which you've probably never heard of.

Jump Commands:

jt0 or jnt0:	jump if line T0 is high or low
jt1 or jnt1:	as above, but for line T1
jni :	jump if line INT=low
jf0 :	jump when flag0 is set
jf1 :	as above, but for flag1

The flags will be used as markers in the program. They will clear with clr f0 and clr f1 respectively, and will set with cpl f0 and cpl f1.

All jump commands, with the exception of jmp and call, can only jump within a page of memory. Therefore, their operands are only one byte in length. For example:

```
0ABF jnz 00
```

The jump (which occurs when the contents of the accumulator are unequal to 0) will not be to 0000 as expected, but to 0A00. In other words, our example is limited in jump size to page 0A.

On to jmp and call. These can move within a 2k range. If they do happen to leave the 2k range, the memory must be switched around, thus:

```
0000 sel mb1
0001 jmp 000
```

This jump doesn't go to 0000, but to 0800, as the memory has been shifted around first. The lower 2k range can be brought back with the command sel mb0, and moved back to the upper set again with sel mb1. Our example requires this change, otherwise the example would not be executable.

Not every one of these commands will appear in the ROM listing, but those that do are commented upon.

Transfer Commands: Transferring refers to moving contents to and from the accumulator. Immediate operations (preceded with #) are obviously transferred immediately, as opposed to waiting for a signal or specific address.

Memory Layout: As you can see from the memory map on the next page, memory is divided into two register banks, the stack, and remaining memory. The memory can be indirectly addressed from all positions. Direct access is possible within certain ranges.

Example:

```
0000 mov r1,#03
0002 sel rb1
0003 mov r1,#04
```

The first command loads memory cell 1; the third command, however, loads memory cell 25, since register bank 1 has been switched by the second command. It can be changed back using sel rb0. You've probably noticed the absence of comparison commands. The 8039 doesn't carry those commands in its set. Comparisons can be done only indirectly. For example, you have a value in the accumulator, and you want to know if the value is equal to 3. Here's what to do:

```
0123 add a,#FD
0125 jz  40
```

If the accumulator did indeed contain a 3, the jump will be executed. Granted, this is a roundabout way of doing this, but it works. We'll leave it at that for now. You will find examples and explanations of I/O programming in the next chapter.

8039 Memory-Map

ADR.		
0 | | | | | 7	R0 | | | | | R7	REGISTER BANK 0
8 | | 23		STACK
24 | | | | | 31	R0 | | | | | R7	REGISTER BANK 1
32 | | | | | 127		FREE MEMORY

In the MPS-801, memory locations 34 through 123 are used as a input line buffer.

7.2 Block Diagram of the MPS 801

Practically all of the 801's workings are displayed in the block illustration. Our overview features program memory only, so we have omitted the drivers for the motor system, etc. We want to spend some time here just explaining some signal names.

The left-hand signals belong to the serial bus, which you might want to review in Chapter 2.1. The right-hand signals control the internal workings of the printer.

TEST This signal tests for the switch at the rear of the printer. Line = low if this TESTs out.

4/5 This is also a switch signal, and is low when the switch on device address 5 is on; otherwise, it is high.

TIMEWHEEL This is the signal discussed in Chapter 6.3.

START POSITION Puts the print carriage to the left, interrupted by a light box. If this be the case, this signal = high.

LF-KEY When the linefeed key is pressed, this line = low.

HEADMOTOR ON This output is low when the transport mechanism for the printhead is moving. It simultaneously turns the roller and timing device.

CARRIAGE RETURN ON A low reading on this line releases a coupling which separates the printhead from the driver, spring tension sends the printhead to the start position.

LINEFEED ON This is the main power supply for the linefeed transport step-motor. In actuality, the motor only moves when a signal is sent over lines LF1, LF2, LF3 AND LF4.

PRINTHAMMER ON As the name indicates, a low reading fires the print hammer.

Programming the Port Lines:

Signal output is directly controlled as a rule. A low reading can be produced using a logical AND, a high with a logical OR, e.g.:

```
0011 anl p1,#FE
0013 orl p1,#01
```

This command sequence releases the printhammer briefly.

The condition of the port lines can be determined by examining the accumulator and testing the bits examined.

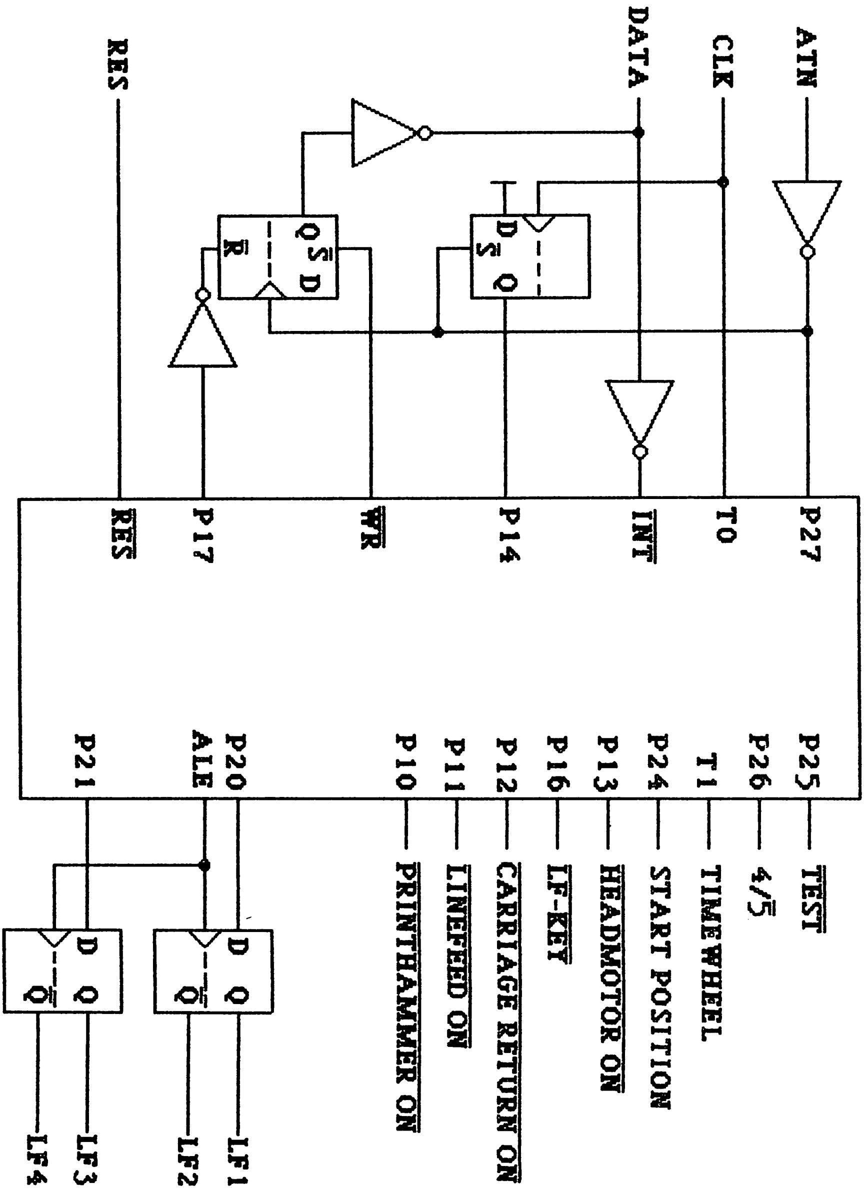
```
0025 mov a,p2
0026 jb6 47
```

This sequence will jump to 0047, if the address switch reads 4.

The lines T0 and T1 can be found in the abovementioned material on jump commands jt0, jt1, jnt0 and jnt1.

So, on to the operating system itself. If any commands are still unclear to you, please consult a command list available at most electronics stores.

Block Diagram of the MPS 801



7.3 MPS-801 ROM-Listing

```
0000 0410 jmp  010
0002 00   nop
0003 00   nop
0004 00   nop
0005 00   nop
0006 00   nop
*********************************Error routine
0007 65   stop tcnt
0008 35   dis  tcnti
0009 15   dis  i
000A 89FF orl  p1,#FF    (255)
000C 8AFC orl  p2,#FC    (252)
000E 040E jmp  00E                 Deadlock
0010 89FF orl  p1,#FF    (255)
0012 8aFC orl  p2,#FC    (252)
0014 15   dis  i
0015 E5   sel  mb0
0016 C5   sel  rb0
0017 65   stop tcnt
0018 35   dis  tcnti
0019 99FD anl  p1,#FD    (253)
001B F5   sel  mb1
001C D437 call 637                 Delay
001E E5   sel  mb0
001F 89FF orl  p1,#FF    (255)
*********************************
0021 27   clr  a                   Register bank 0
0022 B802 mov  r0,#02    (2)       and stack
0024 B916 mov  r1,#16    (22)      both
0026 A5   clr  f1                  cleared
0027 A0   mov  @r0,a
```

```
0028 18   inc  r0
0029 E927 djnz r1,27
**********************************
002B 0A   in   a,p2                     ATN?
002C F22B jb7  2B                       if so, wait
002E 997F anl  p1,#7F     (127)         for DATA
0030 BC01 mov  r4,#01     (1)
0032 5400 call 200                      test head mvt. & timewheel
0034 B811 mov  r0,#11     (17)
0036 F0   mov  a,@r0
0037 53FC anl  a,#FC      (252)
0039 A0   mov  @r0,a
003A 5402 call 202                      move printhead
003C 99FB anl  p1,#FB     (251)         to starting
003E B4CD call 5CD                      position
0040 B810 mov  r0,#10     (16)
0042 B018 mov  @r0,#18    (24)
**********************************
0044 27   clr  a                        clear
0045 B816 mov  r0,#16     (22)
0047 A0   mov  @r0,a                    register bank 1
0048 18   inc  r0
0049 A0   mov  @r0,a                    and
004A B81A mov  r0,#1A     (26)
004C B966 mov  r1,#66     (102)         memory
004E A0   mov  @r0,a
004F 18   inc  r0
0050 E94E djnz r1,4E
**********************************
0052 85   clr  f0
0053 AA   mov  r2,a                     Initialize
0054 AB   mov  r3,a
0055 AD   mov  r5,a
```

```
0056 AE   mov  r6,a                  register bank 0
0057 B821 mov  r0,#21    (33)
0059 B002 mov  @r0,#02   (2)
005B 18   inc  r0
005C FC   mov  a,r4
005D 37   cpl  a
005E B008 mov  @r0,#08   (8)
0060 1268 jb0  68
0062 B00F mov  @r0,#0F   (15)
0064 F268 jb7  68
0066 B00E mov  @r0,#0E   (14)
0068 18   inc  r0
0069 B918 mov  r1,#18    (24)
006B B100 mov  @r1,#00   (0)
006D 2379 mov  a,#79     (121)
006F 5C   anl  a,r4
0070 AC   mov  r4,a
0071 F8   mov  a,r0
0072 AF   mov  r7,a
0073 E400 jmp  700                   END OF RESET
*********************************Data byte in R2 will cause
                                 a jump to LISTEN
0075 FE   mov  a,r6
0076 C67A jz   7A
0078 4427 jmp  227
007A FC   mov  a,r4
007B 128E jb0  8E
007D FA   mov  a,r2
007E F282 jb7  82                    Data byte >127? yes>
0080 2400 jmp  100
0082 B821 mov  r0,#21    (33)
0084 B000 mov  @r0,#00   (0)
0086 BB01 mov  r3,#01    (1)
```

```
0088 B918 mov   r1,#18     (24)
008A B100 mov   @r1,#00    (0)
008C )4B5 jmp   0B5
*********************************
008E FA   mov   a,r2
008F 5360 anl   a,#60      (96)     Printable character?
0091 9695 jnz   95                  yes>
0093 2400 jmp   100
0095 FA   mov   a,r2
0096 03BF add   a,#BF      (191)    >64?
0098 F6A4 jc    A4                  yes>
009A 031F add   a,#1F      (31)     Quotation marks?
009C 96A2 jnz   A2                  nope>
009E FC   mov   a,r4                Switch
009F D302 xrl   a,#02      (2)      quote
00A1 AC   mov   r4,a                flag
00A2 0482 jmp   082
00A4 FC   mov   a,r4
00A5 72AE jb3   AE
00A7 FA   mov   a,r2
00A8 F5   sel   mb1
00A9 9400 call  C00
00AB E5   sel   mb0
00AC 0482 jmp   082
00AE FA   mov   a,r2
00AF F5   sel   mb1
00B0 34D8 call  9D8
00B2 E5   sel   mb0
00B3 0482 jmp   082
********************************* Byte buffer
00B5 FF   mov   a,r7
00B6 A8   mov   r0,a
00B7 FA   mov   a,r2
```

```
00B8 A0   mov  @r0,a
00B9 1F   inc  r7                   Buffer pointer + 1
00BA FE   mov  a,r6
00BB 96CC jnz  CC
00BD FB   mov  a,r3
00BE F2CE jb7  CE
00C0 2384 mov  a,#84     (132)
00C2 6F   add  a,r7                 Pointer >123?
00C3 F6D4 jc   D4                   yes>
00C5 D5   sel  rb1
00C6 F8   mov  a,r0
00C7 C5   sel  rb0
00C8 03FB add  a,#FB     (251)
00CA F6D4 jc   D4
00CC E400 jmp  700                  return to waitloop
00CE B820 mov  r0,#20    (32)
00D0 FB   mov  a,r3
00D1 A0   mov  @r0,a
00D2 04E9 jmp  0E9
*********************************Buffer full
00D4 D5   sel  rb1
00D5 F8   mov  a,r0
00D6 C5   sel  rb0
00D7 E7   rl   a
00D8 E7   rl   a
00D9 E7   rl   a
00DA E7   rl   a
00DB 5370 anl  a,#70     (112)
00DD A8   mov  r0,a
00DE FC   mov  a,r4
00DF 538F anl  a,#8F     (143)
00E1 48B8 jntl B8
00E3 20   xch  a,@r0
```

```
00E4 A0   mov  @r0,a
00E5 FF   mov  a,r7                 Linefeed
00E6 A8   mov  r0,a                 attached
00E7 B00A mov  @r0,#0A   (10)       to buffer
00E9 2378 mov  a,#78     (120)
00EB 5C   anl  a,r4
00EC 4301 orl  a,#01     (1)
00EE AC   mov  r4,a
00EF BF22 mov  r7,#22    (34)       reset buffer pointer
00F1 44AD jmp  2AD                  print buffer contents

00F3 00   00   00   00   00   00   00   00   00   00   00

**********************************Interpret control chars.
0100 FC   mov  a,r4
0101 37   cpl  a                    In quote mode?
0102 3207 jbl  07                   no----->
0104 F5   sel  mbl
0105 04C3 jmp  8C3
0107 23F2 mov  a,#F2     (242)      Double-width on?
0109 6A   add  a,r2
010A c638 jz   38                   yes----->
010C 07   dec  a                    Double-width off?
010D C638 jz   38                   yes----->
010F 07   dec  a                    Print position determined?
0110 C669 jz   69                   yes----->
0112 07   dec  a                    Text mode on?
0113 C673 jz   73                   yes----->
0115 07   dec  a                    Reverse on?
0116 C682 jz   73                   yes----->
0118 23F8 mov  a,#F8     (248)
011A 6A   add  a,r2                 Bit-Image mode?
011B C65F jz   5F                   yes----->
```

```
011D 23E5 mov  a,#E5      (229)
011F 6A   add  a,r2               Print start position(27)?
0120 C69C jz   9C                 yes----->
0122 17   dec  a
0123 C6A1 jz   A1
0125 FA   mov  a,r2
0126 036F add  a,#6F      (111)   Graphic mode?
0128 C6A8 jz   A8                 yes----->
012A 07   dec  a                  Reverse off?
012B C6B2 jz   B2                 yes----->
012D 23F6 mov  a,#F6      (246)
012F 6A   add  a,r2               Linefeed?
0130 C6BE jz   BE                 yes----->
0132 03FD add  a,#FD      (253)   CR?
0134 C6BE jz   BE                 yes----->
0136 E400 jmp  700                return to wait loop
*********************************Control Chars. 14/15
0138 B810 mov  r0,#10     (16)
013A B018 mov  @r0,#18    (24)
013C 2301 mov  a,#01      (1)
013E 4C   orl  a,r4
013F AC   mov  r4,a
0140 B921 mov  r1,#21     (33)
0142 F1   mov  a,@r1
0143 325A jb1  5A
0145 124C jb0  4C
0147 D5   sel  rb1
0148 18   inc  r0
0149 C5   sel  rb1
014A 245B jmp  15B
014C B4F0 call 5F0                The last three bytes
014E B4F0 call 5F0                of the buffer are
0150 B4F0 call 5F0                zeroed out
```

```
0152 F1   mov  a,@r1
0153 F257 jb7  57
0155 245B jmp  15B
0157 D5   sel  rb1
0158 C8   dec  r0
0159 C5   sel  rb0
015A CF   dec  r7
015B B102 mov  @r1,#02    (2)
015D 04B5 jmp  0B5
*********************************Bit-Image mode on
015F B810 mov  r0,#10     (16)
0161 B010 mov  @r0,#10    (16)
0163 FC   mov  a,r4
0164 537C anl  a,#7C      (124)
0166 AC   mov  r4,a
0167 2440 jmp  140
*********************************Control Char. 16
0169 BD02 mov  r5,#02     (2)
016B BB01 mov  r3,#01     (1)
016D D5   sel  rb1
016E 18   inc  r0
016F C5   sel  rb0
0170 1E   inc  r6
0171 04B5 jmp  0B5
*********************************Text mode On
0173 FC   mov  a,r4
0174 1278 jb0  78
0176 E400 jmp  700
0178 B814 mov  r0,#14     (20)
017A 2301 mov  a,#01      (1)
017C 40   orl  a,@r0
017D 53FD anl  a,#FD      (253)
017F A0   mov  @r0,a
```

```
0180 E400 jmp  700
*********************************Reverse on
0182 FC   mov  a,r4
0183 1287 jb0  87
0185 2476 jmp  176
0187 5276 jb2  76
0189 5378 anl  a,#78     (120)
018B 4305 orl  a,#05     (5)
018D AC   mov  r4,a
018E B921 mov  r1,#21    (33)
0190 F1   mov  a,@r1
0191 CF   dec  r7
0192 5298 jb2  98
0194 D5   sel  rb1
0195 18   inc  r0
0196 C5   sel  rb0
0197 1F   inc  r7
0198 B104 mov  @r1,#04   (4)
019A 04B5 jmp  0B5
*********************************Print start 27
019C BD01 mov  r5,#01    (1)
019E 1E   inc  r6
019F E400 jmp  700
*********************************Control Char. 28
01A1 FC   mov  a,r4
01A2 129F jb0  9F
01A4 BD04 mov  r5,#04    (4)
01A6 246D jmp  16D
*********************************Graphic mode on
01A8 B814 mov  r0,#14    (20)
01AA 2302 mov  a,#02     (2)
01AC 40   orl  a,@r0
01AD 53FE anl  a,#FE     (254)
```

```
01AF A0   mov  @r0,a
01B0 E400 jmp  700
*********************************Reverse off
01B2 FC   mov  a,r4
01B3 52B7 jb2  B7
01B5 E400 jmp  700
01B7 5378 anl  a,#78     (120)
01B9 4301 orl  a,#01     (1)
01BB AC   mov  r4,a
01BC 248E jmp  18E
*********************************Linefeed or CR
01BE FC   mov  a,r4
01BF 53FD anl  a,#FD     (253)
01C1 AC   mov  r4,a
01C2 BA0A mov  r2,#0A    (10)
01C4 FB   mov  a,r3
01C5 C6CB jz   CB
01C7 BB81 mov  r3,#81    (129)
01C9 04B5 jmp  0B5
01CB BB80 mov  r3,#80    (128)
01CD 04B5 jmp  0B5

01CF 00
01D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0200 99F3 anl  p1,#f3    (243)   C/R and headmotor on
0202 99FB anl  p1,#FB    (251)   C/R on
0204 B4CD call 5CD               Printhead brought to start
0206 FC   mov  a,r4                    position
0207 538F anl  a,#8F     (143)
0209 AC   mov  r4,a
```

```
020A D46D call 66D                   Test head movement and
020C F9   mov  a,rl                       timewheel
020D 03E7 add  a,#E7     (231)
020F F618 jc   18
0211 B001 mov  @r0,#01   (1)
0213 030C add  a,#0C     (12)
0215 F618 jc   18
0217 10   inc  @r0
0218 B810 mov  r0,#10    (16)
021A 27   clr  a
021B 62   mov  t,a
021C 55   strt t
021D 1621 jtf  21
021F 441D jmp  21D
0221 E81D djnz r0,1D
0223 65   stop tcnt
0224 8908 orl  pl,#08    (8)     Headmotor off
0226 83   ret
*********************************
0227 FD   mov  a,r5
0228 1247 jb0  47
022A 3282 jbl  82
022C FE   mov  a,r6
022D 07   dec  a
022E 9632 jnz  32
0230 246D jmp  16D
0232 FA   mov  a,r2
0233 F243 jb7  43
0235 B4F0 call 5F0
0237 B4F0 call 5F0
0239 D5   sel  rb0
023A C8   dec  r0
023B C5   sel  rb0
```

```
023C B4C9 call 5C9
023E B917 mov  r1,#17    (23)
0240 A1   mov  @r1,a
0241 E400 jmp  700
0243 B4C9 call 5C9
0245 0482 jmp  082
0247 FE   mov  a,r6
0248 07   dec  a
0249 C672 jz   72
024B 07   dec  a
024C C679 jz   79
024E B917 mov  r1,#17    (23)
0250 F1   mov  a,@r1
0251 A9   mov  r1,a
0252 C65C jz   5C
0254 07   dec  a
0255 9637 jnz  37
0257 FA   mov  a,r2
0258 0320 add  a,#20     (32)
025A F637 jc   37
025C B821 mov  r0,#21    (33)
025E F0   mov  a,@r0
025F 12A7 jb0  A7
0261 B081 mov  @r0,#81   (129)
0263 3267 jb1  67
0265 B001 mov  @r0,#01   (1)
0267 FF   mov  a,r7
0268 29   xch  a,r1
0269 A1   mov  @r1,a
026A 1F   inc  r7
026B B4C9 call 5C9
026D B917 mov  r1,#17    (23)
026F A1   mov  @r1,a
```

```
0270 04B5 jmp  0B5
0272 23F0 mov  a,#F0      (240)
0274 6A   add  a,r2
0275 963C jnz  3C
0277 246D jmp  16D
0279 FA   mov  a,r2
027A 5301 anl  a,#01      (1)
027C B917 mov  r1,#17     (23)
027E A1   mov  @r1,a
027F 1E   inc  r6
0280 E400 jmp  700
0282 FE   mov  a,r6
0283 07   dec  a
0284 9689 jnz  89
0286 FA   mov  a,r2
0287 447C jmp  27C
0289 B917 mov  r1,#17     (23)
028B F1   mov  a,@r1
028C 530F anl  a,#0F      (15)
028E E7   rl   a
028F A8   mov  r0,a
0290 B904 mov  r1,#04     (4)
0292 68   add  a,r0
0293 E992 djnz r1,92
0295 A8   mov  r0,a
0296 FA   mov  a,r2
0297 530F anl  a,#0F      (15)
0299 68   add  a,r0
029A A8   mov  r0,a
029B BE05 mov  r6,#05     (5)
029D 68   add  a,r0
029E E6A1 jnc  A1
02A0 19   inc  r1
```

```
02A1 EE9D djnz r6,9D
02A3 AA   mov  r2,a
02A4 F9   mov  a,r1
02A5 4452 jmp  252
02A7 B4F0 call 5F0
02A9 CF   dec  r7
02AA CF   dec  r7
02AB 4467 jmp  267
**********************************Print out buffer contents
02AD 27   clr  a
02AE B815 mov  r0,#15    (21)
02B0 B003 mov  @r0,#03   (3)
02B2 85   clr  f0
02B3 AB   mov  r3,a
02B4 BE18 mov  r6,#18    (24)
02B6 B81E mov  r0,#1E    (30)    Reset character counter
02B8 A0   mov  @r0,a
02B9 18   inc  r0
02BA A0   mov  @r0,a
02BB BD80 mov  r5,#80    (128)
02BD 6401 jmp  301               Start printout
**********************************
02BF FD   mov  a,r5
02C0 F2C4 jb7  C4
02C2 8400 jmp  400               Print character
02C4 65   stop tcnt
02C5 99FB anl  p1,#FB    (251)   C/R on, rest of system out
02C7 FC   mov  a,r4
02C8 538F anl  a,#8F     (143)
02CA AC   mov  r4,a
02CB 27   clr  a
02CC AD   mov  r5,a
02CD A8   mov  r0,a
```

```
02CE 0A   in   a,p2                Printhead at start position?
02CF 92D7 jb4  D7                   yes----->
02D1 D410 call 610                  Time control
02D3 E8CE djnz r0,CE
02D5 0407 jmp  007                  Deadlock
02D7 D46D call 66D                  Wait until printhead has
02D9 F0   mov  a,@r0                left start position
02DA C6E0 jz   E0
02DC 32E9 jb1  E9
02DE 8400 jmp  400                  Print character
02E0 F9   mov  a,r1
02E1 03F3 add  a,#F3      (243)
02E3 F6DE jc   DE
02E5 D4A1 call 6A1                  Timewheel waiting..
02E7 jmp  400                       Print character
02E9 F9   mov  a,r1
02EA 03E7 add  a,#E7      (231)
02EC F6DE jc   DE
02EE 44E5 jmp  2E5

02F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

*********************************Print out buffer
0300 AC   mov  r4,a
0301 FF   mov  a,r7
0302 A8   mov  r0,a
0303 F0   mov  a,@r0                Get byte
0304 1F   inc  r7                   Buffer pointer +1
0305 AA   mov  r2,a                 Byte ->R2
0306 03F0 add  a,#F0      (240)     Print position set up?
0308 961A jnz  1A                   no----->
030A 85   clr  f0
030B 95   cpl  f0
```

```
030C FF   mov  a,r7
030D A8   mov  r0,a
030E F0   mov  a,@r0
030F 1F   inc  r7
0310 B917 mov  r1,#17     (23)
0312 A1   mov  @r1,a
0313 C9   dec  r1
0314 18   inc  r0
0315 F0   mov  a,@r0
0316 A1   mov  @r1,a
0317 1F   inc  r7
0318 8472 jmp  472                Calculate start position
*********************************
031A 17   dec  a                  Text mode on?
031B 9623 jnz  23                 no----->
031D FC   mov  a,r4
031E 537C anl  a,#7C      (124)
0320 17   dec  a
0321 6400 jmp  300
*********************************
0323 17   dec  a                  Reverse on?
0324 962D jnz  2D                 no----->
0326 FC   mov  a,r4
0327 537C anl  a,#7C      (124)
0329 4381 orl  a,#81      (129)
032B 6400 jmp  300
*********************************
032D 03FC add  a,#FC      (252)   =22?
032F 9636 jnz  36                 no----->
0331 FC   mov  a,r4
0332 4304 orl  a,#04      (4)
0334 6400 jmp  300
*********************************
```

```
0336 0380 add  a,#80     (128)   =150?
0338 9643 jnz  43                no
033A FC   mov  a,r4
033B 123F jb0  3F
033D 44BF jmp  2BF
033F 53F9 anl  a,#F9     (249)
0341 6400 jmp  300
*********************************
0343 038A add  a,#8A     (138)   =12?
0345 964D jnz  4D                no----->
0347 85   clr  f0
0348 FC   mov  a,r4
0349 537C anl  a,#7C     (124)   Quote flag off
034B 6400 jmp  300
*********************************
034D 03EE add a,#EE      (238)   =30?
034F 9660 jnz  60                no----->
0351 FF   mov  a,r7
0352 A8   mov  r0,a
0353 F0   mov  a,@r0
0354 AB   mov  r3,a
0355 1F   inc  r7
0356 18   inc  r0
0357 F0   mov  a,@r0
0358 AA   mov  r2,a
0359 1F   inc  r7
035A FC   mov  a,r4
035B 4302 orl  a,#02     (2)
035D AC   mov  r4,a
035E 44BF jmp  2BF
*********************************
0360 0310 add  a,#10     (16)    Double-width on?
0362 9666 jnz  66                no----->
```

```
0364 84FE jmp  4FE
*********************************
0366 FC   mov  a,r4
0367 126B jb0  6B
0369 44BF jmp  2BF
*********************************Any characters to be
                                  printed in enlarged format
                                  are placed in address 24.
036B B81F mov  r0,#1F    (31)
036D F0   mov  a,@r0
036E C684 jz   84
0370 C8   dec  r0
0371 FC   mov  a,r4
0372 F27B jb7  7B
0374 F0   mov  a,@r0
0375 0325 add  a,#25     (37)
0377 F682 jc   82
0379 6484 jmp  384
037B F0   mov  a,@r0
037C 032B add  a,#2B     (43)
037E F682 jc   82
0380 6484 jmp  384                Linefeed
0382 849A jmp  49A
0384 FA   mov  a,r2
0385 0340 add  a,#40     (64)     Character >191?
0387 E696 jnc  96                 no----->
0389 D461 call 661                Position in character
038B F8   mov  a,r0                 generator computed
038C F5   sel  mb1
038D 74D8 call BD8                Get matrix column
038F E5   sel  mb0
0390 18   inc  r0
0391 19   inc  r1
```

```
0392 EB8B djnz r3,8B            Repeat process until 6
0394 64D6 jmp  3D6              columns have printed out
0396 0320 add  a,#20    (32)    Character >159?
0398 E6A7 jnc  A7               no----->
039A D461 call 661              Position in character
039C F8   mov  a,r0               generator computed
039D F5   sel  mb1
039E 54C0 call AC0              Get matrix column
03A0 E5   sel  mb0
03A1 18   inc  r0
03A2 19   inc  r1
03A3 EB9C djnz r3,9C            Repeat until 6 columns
03A5 64D6 jmp  3D6              are printed out
03A7 0340 add  a,#40    (64)    Character >95?
03A9 E6B8 jnc  B8               no----->
03AB D461 call 661              Position in character
03AD F8   mov  a,r0             generator computed
03AE F5   sel  mb1
03AF B4C0 call DC0              Get matrix column
03B1 E5   sel  mb0
03B2 18   inc  r0
03B3 19   inc  r1
03B4 EBAD djnz r3,AD            Repeat until 6 cols. are
03B6 64D6 jmp  3D6              printed out
03B8 0320 add  a,#20    (32)    Character >63?
03BA E6C9 jnc  C9               no----->
03BC D461 call 661              Position in char. gen.
03BE F8   mov  a,r0              calculated
03BF F5   sel  mb1
03C0 34C0 call 9C0              Get matrix column
03C2 E5   sel  mb0
03C3 18   inc  r0
03C4 19   inc  r1
```

```
03C5 EBBE djnz r3,BE                Repeat until 6 cols. are
03C7 64D6 jmp  3D6                  printed out
03C9 0320 add  a,#20      (32)
03CB D461 call 661                  Calculate position in char.
03CD F8   mov  a,r0                 generator
03CE F5   sel  mb1
03CF 14C0 call 8C0                  Get matrix column
03D1 E5   sel  mb0
03D2 18   inc  r0
03D3 19   inc  r1
03D4 EBCD djnz r3,CD                Repeat
03D6 FE   mov  a,r6                  until
03D7 A9   mov  r1,a                   six
03D8 F1   mov  a,@r1                   columns
03D9 AA   mov  r2,a                     have
03DA FC   mov  a,r4                      been
03DB 52DF jb2  DF                   printed out
03DD 44DF jmp  2BF
03DF FA   mov  a,r2
03E0 37   cpl  a
03E1 AA   mov  r2,a
03E2 44BF jmp  2BF                  print out

03E4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

**********************************Print characters
0400 D4E8 call 6E8                  End-of-line reached??
0402 F690 jc   90                   yes----->
0404 FA   mov  a,r2
0405 537F anl  a,#7F      (127)
0407 A9   mov  r1,a
0408 B815 mov  r0,#15     (21)
```

```
040A 10   inc  @r0
040B F0   mov  a,@r0
040C D304 xrl  a,#04     (4)
040E C61B jz   1B
0410 B8F0 mov  r0,#F0    (240)
0412 42   mov  a,t
0413 D302 xrl  a,#02     (2)
0415 C61E jz   1E
0417 E812 djnz r0,12
0419 0407 jmp  007             Deadlock
041B A0   mov  @r0,a
041C 8420 jmp  420
041E 461E jntl 1E              Timer waiting
0420 65   stop tcnt
0421 23C0 mov  a,#C0     (192)
0423 62   mov  t,a             Timer control
0424 55   strt t                 on
0425 25   en   tcnti
0426 F9   mov  a,rl            Get the next point in the
0427 77   rr   a                 matrix column
0428 A9   mov  rl,a
0429 37   cpl  a
042A 562A jtl  2A              Timer waits...
042C F23D jb7  3D              Point printed? no----->
042E 99FE anl  pl,#FE    (254) Fire hammer
0430 27   clr  a
0431 B842 mov  r0,#42    (66)
0433 F9   mov  a,rl
0434 D239 jb6  39
0436 00   nop
0437 B84E mov  r0,#4E    (78)
0439 E839 djnz r0,39
043B 8901 orl  pl,#01    (1)   Hammer off
```

```
043D 1D   inc  r5                     Point counter +1
043E 463E jntl 3E                     Time gap waiting
0440 65   stop tcnt                   Control off
0441 35   dis  tcnti
0442 23F9 mov  a,#F9      (249)       7 points printed??
0444 6D   add  a,r5
0445 E621 jnc  21                     no----->next point
0447 C651 jz   51                     yes----->
0449 03F9 add  a,#F9      (249)       Double-width ready?
044B 9621 jnz  21                     no----->
044D F4CD call 7CD                    Column-count +1 & timer
044F 8464 jmp  464                        start
0451 F4CD call 7CD                      (same as 044D)
0453 B672 jf0  72
0455 FC   mov  a,r4
0456 1262 jb0  62
0458 37   cpl  a
0459 325D jb1  5D
045B EB00 djnz r3,00
045D 4302 orl  a,#02      (2)
045F 37   cpl  a
0460 6400 jmp  300
0462 F26E jb7  6E
0464 1E   inc  r6
0465 FE   mov  a,r6
0466 03E2 add  a,#E2      (226)
0468 9670 jnz  70
046A BE18 mov  r6,#18     (24)
046C 6401 jmp  301                    Get next byte
046E BD07 mov  r5,#07     (7)
0470 64D6 jmp  3D6
*********************************Calculate print start pos.
0472 B81F mov  r0,#1F     (31)
```

```
0474 B917 mov  r1,#17    (23)
0476 F1   mov  a,@r1
0477 AA   mov  r2,a
0478 F0   mov  a,@r0
0479 37   cpl  a
047A 6A   add  a,r2
047B F68C jc   8C
047D 37   cpl  a
047E 9689 jnz  89
0480 C8   dec  r0
0481 C9   dec  r1
0482 F1   mov  a,@r1
0483 AA   mov  r2,a
0484 F0   mov  a,@r0
0485 37   cpl  a
0486 6A   add  a,r2
0487 F68C jc   8C
0489 85   clr  f0
048A 6401 jmp  301
048C 27   clr  a
048D AA   mov  r2,a
048E 44BF jmp  2BF
*********************************
0490 FC   mov  a,r4
0491 37   cpl  a
0492 329A jb1  9A
0494 CF   dec  r7
0495 CF   dec  r7
0496 FF   mov  a,r7
0497 A8   mov  r0,a
0498 FB   mov  a,r3
0499 A0   mov  @r0,a
049A CF   dec  r7
```

```
049B 65   stop tcnt
049C B4FC call 5FC                      Linefeed
049E B820 mov  r0,#20    (32)
04A0 F0   mov  a,@r0
04A1 37   cpl  a
04A2 F2AA jb7  AA
04A4 FC   mov  a,r4
04A5 538F anl  a,#8F     (143)
04A7 AC   mov  r4,a
04A8 44AD jmp  2AD                      print out buffer
04AA B822 mov  r0,#22    (34)
04AC FC   mov  a,r4
04AD 37   cpl  a
04AE 52B3 jb2  B3
04B0 B012 mov  @r0,#12   (18)
04B2 18   inc  r0
04B3 B008 mov  @r0,#08   (8)
04B5 12BD jb0  BD
04B7 B00F mov  @r0,#0F   (15)
04B9 F2BD jb7  BD
04BB B00E mov  @r0,#0E   (14)
04BD FF   mov  a,r7
04BE A9   mov  r1,a
04BF F1   mov  a,@r1
04C0 AA   mov  r2,a
04C1 18   inc  r0
04C2 F9   mov  a,r1
04C3 0381 add  a,#81     (129)
04C5 F6CC jc   CC
04C7 19   inc  r1
04C8 FA   mov  a,r2
04C9 A0   mov  @r0,a
04CA 84BF jmp  4BF
```

```
04CC C8   dec  r0
04CD F0   mov  a,@r0
04CE A9   mov  r1,a
04CF 27   clr  a
04D0 A0   mov  @r0,a
04D1 F9   mov  a,r1
04D2 03F6 add  a,#F6      (246)
04D4 96CC jnz  CC
04D6 F8   mov  a,r0
04D7 AF   mov  r7,a
04D8 27   clr  a
04D9 A0   mov  @r0,a
04DA 18   inc  r0
04DB F8   mov  a,r0
04DC 0380 add  a,#80      (128)
04DE E6D8 jnc  D8
04E0 B4C9 call 5C9
04E2 B820 mov  r0,#20     (32)
04E4 F0   mov  a,@r0
04E5 77   rr   a
04E6 77   rr   a
04E7 77   rr   a
04E8 77   rr   a
04E9 5307 anl  a,#07      (7)
04EB D5   sel  rb1
04EC A8   mov  r0,a
04ED C5   sel  rb0
04EE D305 xrl  a,#05      (5)
04F0 96F4 jnz  F4
04F2 04E5 jmp  0E5
04F4 F0   mov  a,@r0
04F5 4320 orl  a,#20      (32)
04F7 A44A jmp  54A
```

```
04F9 00   nop
04FA 00   nop
04FB 00   nop
04FC 00   nop
04FD 00   nop
********************************Double-width printing
04FE B820 mov  r0,#20    (32)
0500 F0   mov  a,@r0
0501 F251 jb7  51
0503 8908 orl  p1,#08    (8)
0505 FD   mov  a,r5
0506 D380 xrl  a,#80     (128)
0508 C60F jz   0F
050A 99FB anl  p1,#FB    (251)
050C 65   stop tcnt
050D B4CD call 5CD
050F B821 mov  r0,#21    (33)
0511 B000 mov  @r0,#00   (0)
0513 18   inc  r0
0514 B918 mov  r1,#18    (24)
0516 B100 mov  @r1,#00   (0)
0518 FC   mov  a,r4
0519 37   cpl  a
051A 521F jb2  1F
051C B012 mov  @r0,#12   (18)
051E 18   inc  r0
051F B008 mov  2r0,#08   (8)
0521 1229 jb0  29
0523 B00F mov  2r0,#0F   (15)
0525 F229 jb7  29
0527 B00E mov  @r0,#0E   (14)
0529 18   inc  r0
052A B010 mov  @r0,#10   (16)
```

```
052C 18   inc  r0
052D B91F mov  r1,#1F     (31)
052F F1   mov  a,@r1
0530 A0   mov  @r0,a
0531 C9   dec  r1
0532 18   inc  r0
0533 F1   mov  a,@r1
0534 A0   mov  @r0,a
0535 18   inc  r0
0536 F8   mov  a,r0
0537 AF   mov  r7,a
0538 B4C9 call 5C9
053A F8   mov  a,r0
053B 37   cpl  a
053C 0381 add  a,#81      (129)
053E A9   mov  r1,a
053F 27   clr  a
0540 A0   mov  @r0,a
0541 18   inc  r0
0542 E940 djnz r1,40
0544 B820 mov  r0,#20     (32)
0546 F0   mov  a,@r0
0547 5302 anl  a,#02      (2)
0549 4C   orl  a,r4
054A 532F anl  a,#2F      (47)
054C AC   mov  r4,a
054D BB01 mov  r3,#01     (1)
054F E400 jmp  700
0551 B4FC call 5FC
0553 B87F mov  r0,#7F     (127)
0555 F0   mov  a,@r0
0556 C662 jz   62
0558 0A   in   a,p2
```

```
0559 37   cpl  a
055A B264 jb5  C4
055C FC   mov  a,r4
055D 5371 anl  a,#71      (113)
055F 4301 orl  a,#01      (1)
0561 AC   mov  r4,a
0562 0444 jmp  044
0564 B87F mov  r0,#F7     (247)
0566 F0   mov  a,@r0
0567 966F jnz  6F
0569 B020 mov  @r0,#20    (32)
056B 23F7 mov  a,#F7      (247)
056D 5C   anl  a,r4
056E AC   mov  r4,a
056F B922 mov  r1,#22     (34)
0571 B10F mov  @r1,#0F    (15)
0573 19   inc  r1
0574 FC   mov  a,r4
0575 BC00 mov  r4,#00     (0)
0577 37   cpl  a
0578 727C jb3  7C
057A BC01 mov  r4,#01     (1)
057C BB50 mov  r3,#50     (80)
057E F0   mov  a,@r0
057F AA   mov  r2,a
0580 03BF add  a,#BF      (191)
0582 F6AD jc   AD
0584 0341 add  a,#41      (65)
0586 A1   mov  @r1,a
0587 10   inc  @r0
0588 19   inc  r1
0589 F0   mov  a,@r0
058A C6BF jz   BF
```

```
058C 0380 add  a,#80     (128)
058E 9692 jnz  92
0590 B0A0 mov  @r0,#A0   (160)
0592 EB7E djnz r3,#7E
0594 B10A mov  @r1,#0A   (10)
0596 BB81 mov  r3,#81    (129)
0598 BF22 mov  r7,#22    (34)
059A FC   mov  a,r4
059B C6A1 jz   A1
059D BC09 mov  r4,#09    (9)
059F A4A3 jmp  5A3
05A1 BC01 mov  r4,#01    (1)
05A3 B820 mov  r0,#20    (32)
05A5 B081 mov  @r0,#81   (129)
05A7 B810 mov  r0,#10    (16)
05A9 B018 mov  @r0,#18   (24)
05AB 44AD jmp  2AD
05AD FC   mov  a,r4
05AE C6B6 jz   B6
05B0 F5   sel  mb1
05B1 34D8 call 9D8
05B3 E5   sel  mb0
05B4 A4BA jmp  5BA
05B6 F5   sel  mb1
05B7 9400 call C00
05B9 E5   sel  mb0
05BA FA   mov  a,r2
05BB B87F mov  r0,#7F    (127)
05BC A486 jmp  586
05BE FC   mov  a,r4
05C0 BC01 mov  r4,#01    (1)
05C2 C6C5 jz   C5
05C4 CC   dec  r4
```

```
05C5 B020 mov   @r0,#20   (32)
05C7 A492 jmp   592
05C9 27   clr   a
05CA AD   mov   r5,a
05CB AE   mov   r6,a
05CC 83   ret
*********************************
05CD FC   mov   a,r4
05CE B2E6 jb5   E6
05D0 D425 call  625               call in timer
05D2 0A   in    a,p2
05D3 37   cpl   a                 Head in start position?
05D4 92E7 jb4   E7                no----->
05D6 D425 call  625
05D8 4320 orl   a,#20     (32)
05DA AC   mov   r4,a
05DB FC   mov   a,r4
05DC 37   cpl   a
05DD B2E6 jb5   E6
05DF D430 call  630
05E1 0A   in    a,p2
03E2 92DB jb4   DB
05E4 A4D2 jmp   5D2
05E6 83   ret
05E7 D430 call  630               Test timer
05E9 A4D2 jmp   5D2
*********************************Linefeed
05EB F5   sel   mb1
05EC C400 jmp   E00
05EE E5   sel   mb0
05EF 83   ret
*********************************
05F0 CF   dec   r7                Last byte
```

```
05F1 FF   mov  a,r7                     in buffer
05F2 A8   mov  r0,a                      zeroed out
05F3 27   clr  a
05F4 A0   mov  @r0,a
05F5 83   ret
*********************************
05F6 D410 call 610
05F8 D410 call 610
05FA C410 jmp  610
*********************************Linefeed
05FC 8908 orl  p1,#08    (8)      Headmotor off
05FE B820 mov  r0,#20    (32)
0600 F0   mov  a,@r0
0601 37   cpl  a
0602 F206 jb7  06
0604 120D jb0  0D
0606 99FB anl  p1,#FB    (251)    Carriage return on
0608 B4EB call 5EB                Linefeed
060A B4CD call 5CD                Wait until head is in
060C 83   ret                       start position
060D B4EB call 5EB                Linefeed
060F 83   ret
*********************************Time Control
0610 234F mov  a,#4F     (79)
0612 62   mov  t,a
0613 55   strt t
0614 1618 jtf  18
0616 C414 jmp  614
0618 161C jtf  1C
061A C418 jmp  618
061C 1620 jtf  20
061E C41C jmp  61C
0620 FC   mov  a,r4
```

```
0621 B233 jb5  33
0623 65   stop tcnt
0624 83   ret
*********************************
0625 27   clr  a
0626 D5   sel  rb1
0627 A9   mov  r1,a
0628 C5   sel  rb0
0629 62   mov  t,a
062A 55   strt t
062B FC   mov  a,r4
062C 53AF anl  a,#AF     (175)
062E AC   mov  r4,a
062F 83   ret
*********************************
0630 1635 jtf  35
0632 83   ret
0633 D5   sel  rb1
0634 19   inc  r1
0635 D5   sel  rb1
0636 19   inc  r1
0637 F9   mov  a,r1
0638 C5   sel  rb0
0639 9648 jnz  48
063B FC   mov  a,r4
063C 37   cpl  a
063D 9243 jb4  43
063F D25C jb6  5C
0641 0407 jmp  007
0643 37   cpl  a
0644 4310 orl  a,#10     (16)
0646 AC   mov  r4,a
0647 83   ret
```

```
0648 03F3 add  a,#F3     (243)
064A FC   mov  a,r4
064B 37   cpl  a
064C B232 jb5  32
064E E632 jnc  32
0650 FC   mov  a,r4
0651 538F anl  a,#8F     (143)
0653 AC   mov  r4,a
0654 8904 orl  p1,#04    (4)    Carriage return off
0656 27   clr  a
0657 D5   sel  rb1
0658 A9   mov  r1,a
0659 C5   sel  rb0
065A 65   stop tcnt
065B 83   ret
065C 37   cpl  a
065D 4340 orl  a,#40     (64)
065F AC   mov  r4,a
0660 83   ret
*********************************
0661 AA   mov  r2,a             Position
0662 97   clr  c                in
0663 F7   rlc  a                character
0664 AA   mov  r2,a             generator
0665 F7   rlc  a                calculated
0666 6A   add  a,r2
0667 A8   mov  r0,a
0668 B918 mov  r1,#18    (24)
066A BB06 mov  r3,#06    (6)
066C 83   ret
*********************************
066D 99F3 anl  p1,#F3    (243)  Head motor and c/r on
066F 27   clr  a
```

```
0670 62   mov  t,a
0671 B80D mov  r0,#0D    (13)
0673 55   strt t
0674 1678 jtf  78
0676 C474 jmp  674
0678 E874 djnz r0,74
067A 65   stop tcnt
067B 23C0 mov  a,#C0     (192)
067D 62   mov  t,a
067E 55   strt t
067F 25   en   tcnti             Deadlock--if no beat
0680 4684 jntl 84                follows two successive
0682 5682 jtl  82                beats,
0684 4684 jntl 84                the timewheel
0686 5686 jtl  86                pauses & waits
0688 65   stop tcnt
0689 35   dis  tcnti
068A 8904 orl  p1,#04    (4)     Carriage return off
068C 27   clr  a
068D 62   mov  t,a
068E 55   strt t
068F B920 mov  r1,#20    (32)
0691 0A   in   a,p2              Has the printhead
0692 92BD jb4  BD                left starting
0694 0A   in   a,p2              position?
0695 92BD jb4  BD                no----->
0697 B900 mov  r1,#00    (0)
0699 65   stop tcnt
069A 169C jtf  9C
069C 23C0 mov  a,#C0     (192)
069E 62   mov  t,a
069F 55   strt t
06A0 25   en   tcnti
```

```
**********************************Wait for beat fr. timer
06A1 56A1 jtl  A1
06A3 46A3 jntl A3
06A5 65   stop tcnt
06A6 35   dis  tcnti
06A7 19   inc  r1
06A8 27   clr  a
06A9 62   mov  t,a
06AA B81E mov  r0,#1E     (30)
06AC A0   mov  @r0,a
06AD 18   inc  r0
06AE A0   mov  @r0,a
06AF 55   strt t
06B0 46A3 jntl A3
06B2 42   mov  a,t
06B3 37   cpl  a
06B4 72B0 jb3  B0
06B6 12B0 jb0  B0
06B8 65   stop tcnt
06B9 B811 mov  r0,#11     (17)
06BB 97   clr  c
06BC 83   ret
**********************************
06BD 16C1 jtf  C1                 Timer running? yes----->
06BF C491 jmp  691
06C1 E991 djnz r1,91              Repeat until R1=0, then
06C3 0407 jmp  007                deadlock.
**********************************Linefeed
06C5 8908 orl  p1,#08     (08)    Head motor off
06C7 B810 mov  r0,#10     (16)
06C9 B018 mov  @r0,#18    (24)
06CB 0A   in   a,p2
06CC 92D2 jb4  D2
```

```
06CE 99FB anl  p1,#FB    (251)   Carriage return on
06D0 B4CD call 5CD
06D2 09   in   a,p1              Linefeed key pressed?
06D3 D2DD jb6  DD                nope----->
06D5 B4EB call 5EB
06D7 B4F6 call 5F6
06D9 B4F6 call 5F6
06DB C4D2 jmp  6D2
06DD E400 jmp  700               Wait-loop
*********************************
06DF 997F anl  p1,#7F    (127)   DATA flipflop released
06E1 90   movx @r0,a             DATA byte set low
06E2 83   ret
*********************************
06E3 8980 orl  p1,#80    (128)   DATA byte set high
06E5 997F anl  p1,#7F    (127)
06E7 83   ret
*********************************Check for end-of-line
06E8 97   clr  c
06E9 B81F mov  r0,#1F    (31)
06EB F0   mov  a,@r0
06EC C6F6 jz   F6
06EE 03FE add  a,#FE     (254)
06F0 F6F6 jc   F6
06F2 C8   dec  r0
06F3 F0   mov  a,@r0
06F4 0320 add  a,#20     (32)
06F6 83   ret

06F7 00 00 00 00 00 00 00 00 00

*********************************Wait-loop
0700 7613 jf1  13                Addressed as LISTENER? yes->
```

```
0702 0A   in   a,p2                ATN?
0703 F21A jb7  1A                  Yes----->
0705 D4E3 call 6E3                 DATA set high
0707 0A   in   a,p2                ATN?
0708 F21A jb7  1A                  yes----->
070A B20E jb5  0E                  Test mode?
070C A464 jmp  564                 yes----->
070E 09   in   a,pl                Linefeed key pressed?
070F D200 jb6  00                  no----->
0711 C4C5 jmp  6C5                 Linefeed cleared
0713 0A   in   a,p2                ATN?
0714 F21A jb7  1A                  yes----->
0716 364C jt0  4C                  CLK high? yes----->
0718 E400 jmp  700                 wait
*********************************Jump to ATN
071A D4DF call 6DF                 DATA set low
071C B8DC mov  r0,#DC     (220)    Counter set
071E D4DF call 6DF                 DATA set low
0720 E81E djnz r0,1E               wait
0722 09   in   a,pl                CLK    low?
0723 37   cpl  a
0724 9200 jb4  00                  no----->
0726 B814 mov  r0,#14     (20)
0728 23CF mov  a,#CF      (207)
072A 50   anl  a,@r0
072B A0   mov  @r0,a
072C B814 mov  r0,#14     (20)     counter set to wait by CLK
072E B908 mov  rl,#08     (8)      Bit counter set
0730 8980 orl  pl,#80     (128)    DATA opened for input
0732 BA01 mov  r2,#01     (1)
0734 863C jni  3C                  DATA high?  yes----->
0736 0A   in   a,p2                ATN still low?
0737 37   cpl  a
```

```
0738 F200 jb7  00                   no, Error Alarm--->
073A E434 jmp  734
073C 2643 jnt0 43                   CLK still low?   yes----->
073E 0A   in   a,p2                 ATN still low?
073F F248 jb7  48                   yes----->
0741 E400 jmp  700                  Error Alarm-->
0743 0A   in   a,p2                 ATN STILL low?
0744 F27D jb7  7D                   yes----->
0746 E400 jmp  700                  Error Alarm-->
0748 E83C djnz r0,3C                Wait until CLK=low
074A E469 jmp  769
********************************Wait for a byte
074C B814 mov  r0,#14     (20)
074E B908 mov  r1,#08     (8)       Set bit counter
0750 8980 orl  p1,#80     (128)     set DATA high
0752 BA14 mov  r2,#14     (20)
0754 0A   in   a,p2                 ATN low?
0755 F21A jb7  1A                   yes----->
0757 865B jni  5B                   DATA high? yes----->
0759 E452 jmp  752                   If not, wait
075B 2664 jnt0 64                   CLK still low? yes----->
075D 0A   in   a,p2                 ATN low?
075E F21A jb7  1A                   yes----->
0760 E85B djnz r0,5B
0762 E469 jmp  769
0764 0A   in   a,p2                 ATN low?
0765 F21A jb7  1A                   yes----->
0767 E47D jmp  77D
0769 D4DF call 6DF                  DATA set low
076B B821 mov  r0,#21     (33)
076D E68D djnz r0,6D                wait
076F 8980 orl  p1,#80     (128)     set DATA high
0771 0A   in   a,p2
```

```
0772 EA78 djnz r2,78
0774 F27A jb7  7A
0776 E400 jmp  700
0778 F21A jb7  1A
077A 1A   inc  r2
077B 3671 jt0  71
**********************************Get a byte
077D 267D jnt0 7D                          Wait 'til CLK=high
077F 97   clr  c                           Shift
0780 8683 jni  83                          bits
0782 A7   cpl  c                           to
0783 F8   mov  a,r0                        R0
0784 67   rrc  a
0785 A8   mov  r0,a
0786 E971 djnz r1,71                       Repeat until bitcounter=0
0788 0A   in   a,p2                        ATN=low?
0789 EA8F djnz r2,8F
078B F291 jb7  91                          yes----->
078D E400 jmp  700                         Wait-loop
078F F21A jb7  1A                          ATN low? yes----->
0791 1A   inc  r2
0792 3688 jt0  88                          Wait until CLK=low to
0794 F8   mov  a,r0                        invert byte received
0795 37   cpl  a                           and bring to
0796 AA   mov  r2,a                          R2.
0797 0A   in   a,p2                        ATN low?
0798 F2A7 jb7  A7                          yes----->
079A 769E jf1  9E                          addrsd as LISTENER? yes-->
079C E400 jmp  700                         Wait-loop
079E D4DF call 6DF                         DATA set low (received)
07A0 B814 mov  r0,#14    (20)
07A2 F0   mov  a,@r0
07A3 F200 jb7  00
```

```
07A5 0475 jmp  075
07A7 237F mov  a,#7F      (127)   Bring address
07A9 5A   anl  a,r2               received to R2
07AA AA   mov  r2,a               during ATN
07AB B814 mov  r0,#14     (20)
07AD 23C1 mov  a,#C1      (193)   UNLISTEN?
07AF 6A   add  a,r2
07B0 96B7 jnz  B7                 no----->
07B2 A5   clr  f1                 LISTEN-flag reset
07B3 B000 mov  @r0,#00    (0)
07B5 E4BA jmp  7BA
07B7 F5   sel  mb1
07B8 C478 jmp  E78                Check address
07BA D4DF call 6DF                DATA set low
07BC 36C3 jt0  C3                 CLK high? yes----->
07BE 0A   in   a,p2               ATN low?
07BF F2BC jb7  BC                 yes----->
07C1 E400 jmp  700                Wait-loop
07C3 0A   in   a,p2               ATN low?
07C4 F22C jb7  2C                 yes----->
07C6 E4C1 jmp  7C1
**********************************Wait until addressing
                                    cycle ends
07C8 0A   in   a,p2               ATN low?
07C9 F2C8 jb7  C8                 yes----->
07CB E400 jmp  700                Wait-loop
**********************************
07CD 45   strt cnt                Start beat-counter
07CE 27   clr  a                  and
07CF 62   mov  t,a                Line-
07D0 AD   mov  r5,a               counter
07D1 B81E mov  r0,#1E     (30)    +1
07D3 10   inc  @r0
```

```
07D4 F0   mov  a,@r0
07D5 18   inc  r0
07D6 96D9 jnz  D9
07D8 10   inc  @r0
07D9 83   ret
07DA 00 00 00 00 00 00
07E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
07F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
0800 00 00 00 00 00 00
```

```
0806 00 00 5F 00 00 00
  *
  *
  *
  *
  *

  *
```

```
080C 00 07 00 07 00 00
  * *
  * *
  * *
```

```
0812 14 7F 14 7F 14 00
 * *
 * *
*****
 * *
*****
 * *
 * *
```

```
0818 24 2A 7F 2A 12 00
  *
 ****
* *
 ***
  * *
****
  *
```

```
081E 63 13 08 64 63 00
**  *
**  *
   *
  *
 *
*  **
*  **
```

```
0824 30 4E 59 26 50 00
  *
 * *
 * *
 **
* * *
*  *
 ** *
```

```
082A 00 04 02 01 00 00
   *
  *
 *
```

```
0830 00 1C 22 41 00 00
           *
          *
         *
         *
         *
          *
           *
```

```
0836 00 41 22 1C 00 00
         *
          *
           *
           *
           *
          *
         *
```

```
083C 2A 1C 7F 1C 2A 00
          *
        * * *
         ***
        *****
         ***
        * * *
          *
```

```
0842 08 08 3E 08 08 00

          *
          *
        *****
          *
          *

```

```
0848 00 40 30 00 00 00




          *
          *
         *
```

```
084E 08 08 08 08 08 00



        *****



```

```
0854 00 60 60 00 00 00





         **
         **
```

```
085A 60 10 08 04 03 00
            *
            *
           *
          *
         *
        *
        *
```

```
0860 3E 51 49 45 3E 00
        ***
       *   *
       *  **
       * * *
       **  *
       *   *
        ***
```

```
0866 00 42 7F 40 00 00
         *
        **
         *
         *
         *
         *
        ***
```

```
086C 62 51 51 49 46 00
        ***
       *   *
           *
          *
        **
       *
       *****
```

```
0872 22 41 49 49 36 00
        ***
       *   *
           *
         **
           *
       *   *
        ***
```

```
0878 18 14 12 7F 10 00
          *
         **
        * *
       *  *
       *****
          *
          *
```

```
087E 27 45 45 45 39 00
       *****
       *
       ****
           *
           *
       *   *
        ***
```

```
0884 3C 4A 49 49 31 00
         ***
        *
       *
       ****
       *   *
       *   *
        ***
```

```
088A 01 71 09 05 03 00
       *****
           *
          *
         *
        *
        *
        *
```

```
0890 36 49 49 49 36 00
       ***
      *   *
      *   *
       ***
      *   *
      *   *
       ***

0896 46 49 49 29 1E 00
       ***
      *   *
      *   *
       ****
          *
         *
      ***

089C 00 00 12 00 00 00

        *


        *



08A2 00 40 32 00 00 00

        *


        *
        *
       *

08A8 08 14 22 41 41 00
         **
        *
       *
      *
       *
        *
         **

08AE 14 14 14 14 14 00


      *****

      *****



08B4 41 41 22 14 08 00
      **
        *
         *
          *
         *
        *
      **

08BA 02 01 51 09 06 00
       ***
      *   *
          *
         *
        *

        *
```

```
*********************************
08C0 A3   movp a,@a               Get matrix
08C1 A1   mov  @r1,a              column
08C2 83   ret
*********************************
08C3 FA   mov  a,r2
08C4 F2D7 jb7  D7
08C6 03E0 add  a,#E0     (224)    Char.>31?
08C8 F6D1 jc   D1                 yes----->
08CA 0313 add  a,#13     (19)     CR?
08CC C6D4 jz   d4                 yes----->
08CE FC   mov  a,r4
08CF 12D7 jb0  D7
08D1 E5   sel  mb0
08D2 E400 jmp  700                Wait-loop
08D4 E5   sel  mb0
08D5 2407 jmp  107                Utilize control chars.
08D7 34C3 call 9C3                Execute RVS-ON--
08D9 FA   mov  a,r2               change control chars.
08DA 0340 add  a,#40     (64)     into
08DC AA   mov  r2,a               printable chars.
08DD C470 jmp  E70
08DF FC   mov  a,r4
08E0 72E6 jb3  E6
08E2 9400 call C000
08E4 04E8 jmp  8E8
08E6 34D8 call 9D8
08E8 FF   mov  a,r7
08E9 A8   mov  r0,a
08EA FA   mov  a,r2
08EB A0   mov  @r0,a
08EC 1F   inc  r7
08ED FC   mov  a,r4
```

```
08EE 53FB anl  a,#FB     (251)
08F0 AC   mov  r4,a
08F1 B821 mov  r0,#21    (33)
08F3 B004 mov  @r0,#04   (4)
08F5 BB01 mov  r3,#01    (1)
08F7 B818 mov  r0,#18    (24)
08F9 B001 mov  @r0,#01   (1)
08FB BA92 mov  r2,#92    (146)
08FD E5   sel  mb0
08FE 04B5 jmp  0B5
```

```
0900 3E 41 5D 55 5E 00
      ***
     *   *
     * ***
     * * *
     * ***
     *
      ****
```

```
0906 7E 09 09 09 7E 00
      ***
     *   *
     *   *
     *****
     *   *
     *   *
     *   *
```

```
090C 41 7F 49 49 36 00
     ****
      *  *
      *  *
      ***
      *  *
      *  *
     ****
```

```
0912 3E 41 41 41 22 00
      ***
     *   *
     *
     *
     *
     *   *
      ***
```

```
0918 41 7F 41 41 3E 00
     ****
      *  *
      *  *
      *  *
      *  *
      *  *
     ****
```

```
091E 7F 49 49 49 41 00
     *****
     *
     *
     ****
     *
     *
     *****
```

```
0924 7F 09 09 09 01 00
     *****
     *
     *
     ****
     *
     *
     *
```

```
092A 3E 41 41 49 3A 00
      ***
     *   *
     *
     *  **
     *   *
     *   *
      ***
```

```
0930 7F 08 08 08 7F 00
       *   *
       *   *
       *   *
       *****
       *   *
       *   *
       *   *

0936 00 41 7F 41 00 00
        ***
         *
         *
         *
         *
         *
        ***

093C 20 40 41 3F 01 00
         ***
          *
          *
          *
          *
       *  *
        **

0942 7F 08 14 22 41 00
       *   *
       *  *
       * *
       **
       * *
       *  *
       *   *

0948 7F 40 40 40 40 00
       *
       *
       *
       *
       *
       *
       *****

094E 7F 02 0C 02 7F 00
       *   *
       ** **
       * * *
       * * *
       *   *
       *   *
       *   *

0954 7F 04 08 10 7F 00
       *   *
       *   *
       **  *
       * * *
       *  **
       *   *
       *   *

095A 3E 41 41 41 3E 00
        ***
       *   *
       *   *
       *   *
       *   *
       *   *
        ***
```

```
0960 7F 09 09 09 06 00
       ****
       *   *
       *   *
       ****
       *
       *
       *
```

```
0966 3E 41 51 21 5E 00
        ***
       *   *
       *   *
       *   *
       * * *
       *  *
        ** *
```

```
096C 7F 09 19 29 46 00
       ****
       *   *
       *   *
       ****
       * *
       *  *
       *   *
```

```
0972 26 49 49 49 32 00
        ***
       *   *
       *
        ***
           *
       *   *
        ***
```

```
0978 01 01 7F 01 01 00
       *****
         *
         *
         *
         *
         *
         *
```

```
097E 3F 40 40 40 3F 00
       *   *
       *   *
       *   *
       *   *
       *   *
       *   *
        ***
```

```
0984 07 18 60 18 07 00
       *   *
       *   *
       *   *
        * *
        * *
         *
         *
```

```
098A 7F 20 18
       *   *
       *   *
       *   *
       * * *
       * * *
       ** **
       *   *
```

```
0990 63 14 08 14 63 00
       *   *
       *   *
        * *
         *
        * *
       *   *
       *   *
```

```
0996 07 08 78 08 07 00
       *   *
       *   *
       *   *
        ***
         *
         *
         *
```

```
099C 61 51 49 45 43 00
       *****
           *
          *
         *
        *
       *
       *****
```

```
09A2 00 7F 41 41 00 00
        ***
        *
        *
        *
        *
        *
        ***
```

```
09A8 48 7E 49 49 42 00
         **
        *  *
        *
       ****
        *
        *
       *****
```

```
09AE 00 41 41 7F 00 00
        ***
          *
          *
          *
          *
          *
        ***
```

```
09B4 00 04 02 7F 02 04
          *
         ***
        * * *
          *
          *
          *
          *
```

```
09BA 08 1C 2A 08 08 08

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

```
**********************************
09C0 A3   movp a,@a                  Get
09C1 A1   mov  @r1,a                 matrix column
09C2 83   ret
**********************************
09C3 FC   mov  a,r4                  Insert
09C4 52D2 jb2  D2
09C6 B821 mov  r0,#21     (33)       RVS
09C8 F0   mov  a,@r0
09C9 CF   dec  r7                    character
09CA 52D2 jb2  D2
09CC 1F   inc  r7                    in
09CD FF   mov  a,r7
09CE A8   mov  r0,a                  buffer
09CF B012 mov  @r0,#12    (18)
09D1 1F   inc  r7                    to convert
09D2 82   ret
**********************************
09D3 FA   mov  a,r2                  control codes
09D4 0340 add  a,#40      (64)       into
09D6 AA   mov  r2,a                  printable
09D7 83   ret                        characters

**********************************Character conversion
09D8 B814 mov  r0,#14     (20)
09DA F0   mov  a,@r0
09DB 37   cpl  a
09DC 32E0 jb1  E0
09DE 8409 jmp  409
09E0 9409 call 409
09E2 FA   mov  a,r2                  Character > 219?
09E3 0325 add  a,#25      (37)
09E5 F6FE jc   FE                    yes----->
```

```
09E7 031B add  a,#1B      (27)     =192?
09E9 C6FE jz   FE                  yes----->
09EB EF61 jnc  F1                  <192 ? yes----->
09ED 0340 add  a,#40      (64)
09EF 24FD jmp  1FD
09F1 0320 add  a,#20      (32)     >160?
09F3 F6FE jc   FE                  yes----->
09F5 FA   mov  a,r2
09F6 03A5 add  a,#A5      (165)    >91?
09F8 F6FE jc   FE                  yes----->
09FA FA   mov  a,r2
09FB 0380 add a,#80       (128)
09FD AA   mov  r2,a
09FE C44B jmp  64B
```

```
0A00 00 00 00 00 00 00
```

```
0A06 7F 7F 7F 00 00 00
       ***
       ***
       ***
       ***
       ***
       ***
       ***
```

```
0A0C 78 78 78 78 78 78



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

```
0A12 01 01 01 01 01 01
       ******
```

```
0A18 40 40 40 40 40 40






       ******
```

```
0A1E 7F 00 00 00 00 00
       *
       *
       *
       *
       *
       *
       *
```

```
0A24 55 2A 55 2A 55 2A
       * * *
        * * *
       * * *
        * * *
       * * *
        * * *
       * * *
```

```
0A2A 00 00 00 00 00 7F
            *
            *
            *
            *
            *
            *
            *
```

```
0A30 50 28 50 28 50 28



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

```
0A36 3F 1F 0F 07 03 01
       ******
       *****
       ****
       ***
       **
       *
```

```
0A3C 00 00 00 00 00 7F
            *
            *
            *
            *
            *
            *
            *
```

```
0A42 00 00 00 7F 08 08
           *
           *
           *
           ***
           *
           *
           *
```

```
0A48 00 00 00 78 78 78



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

```
0A4E 00 00 00 0F 08 08
           *
           *
           *
           ***
```

```
0A54 08 08 08 78 00 00



       ****
          *
          *
          *
```

```
0A5A 60 60 60 60 60 60





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

```
0A60 00 00 00 78 08 08



        ***
        *
        *
        *
```

```
0A66 08 08 08 0F 08 08
        *
        *
        *
     ******
```

```
0A6C 08 08 08 78 08 08



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

```
0A72 08 08 08 7F 00 00
        *
        *
        *
     ****
        *
        *
        *
```

```
0A78 7F 00 00 00 00 00
     *
     *
     *
     *
     *
     *
     *
```

```
0A7E 7F 7F 00 00 00 00
     **
     **
     **
     **
     **
     **
     **
```

```
0A84 00 00 00 00 7F 7F
         **
         **
         **
         **
         **
         **
         **
```

```
0A8A 01 01 01 01 01 01
     ******
```

```
0A90 03 03 03 03 03 03       0A96 70 70 70 70 70 70
       ******
       ******


                                    ******
                                    ******
                                    ******
0A9C 40 40 40 40 40 7F       0AA2 78 78 78 00 00 00
            *
            *
            *
            *                       ***
            *                       ***
            *                       ***
       ******                       ***
0AA8 00 00 00 07 07 07       0AAE 08 08 08 0F 00 00
          ***                          *
          ***                          *
          ***                          *
                                    ****



0AB4 07 07 07 00 00 00       0ABA 07 07 07 78 78 78
       ***                          ***
       ***                          ***
       ***                          ***
                                       ***
                                       ***
                                       ***
                                       ***
```

```
*********************************
0AC0 A3   movp a,@a               Get
0AC1 A1   mov  @rl,a              matrix column
0AC2 83   ret
*********************************
0AC3 00 00 00 00 00 00 00 00 00 00 00 00 00
0AD0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0AE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0AF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0B00 08 08 08 08 08 08       0B06 1C 4E 7F 4E 1C 00
                                      *
                                     ***
                                    *****
       ******                       *****
                                    * * *
                                      *
                                     ***
0B0C 00 00 7F 00 00 00       0B12 04 04 04 04 04 04
         *
         *
         *                          ******
         *
         *
         *
         *
0B18 02 02 02 02 02 02       0B1E 01 01 01 01 01 01
                                    ******
       ******
```

```
0B24 20 20 20 20 20 20





       ******

```

```
0B2A 00 7F 00 00 00 00
        *
        *
        *
        *
        *
        *
        *
```

```
0B30 00 00 00 00 7F 00
           *
           *
           *
           *
           *
           *
           *
```

```
0B36 08 08 10 60 00 00



       **
         *
          *
          *
```

```
0B3C 00 00 00 03 04 08
          *
          *
           *
            *



```

```
0B42 08 08 04 03 00 00
          *
          *
         *
       **



```

```
0B48 7F 40 40 40 40 40
       *
       *
       *
       *
       *
       *
       ******
```

```
0B4E 02 04 08 10 20 40

       *
        *
         *
          *
           *
            *
```

```
0B54 40 20 10 08 04 02

            *
           *
          *
         *
        *
       *
```

```
0B5A 7F 01 01 01 01 01
       ******
       *
       *
       *
       *
       *
       *
```

```
0B60 01 01 01 01 01 7F
       ******
            *
            *
            *
            *
            *
            *
```

```
0B66 3C 7E 7E 7E 3C 00

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

```
0B6C 20 20 20 20 20 20





       ******
```

```
0B72 1C 3E 7C 3E 1C 00

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

```
0B78 00 7F 00 00 00 00
        *
        *
        *
        *
        *
        *
        *
```

```
0B7E 00 00 00 60 10 08



            *
           *
          *
          *
```

```
0B84 42 24 18 18 24 42

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

```
0B8A 3C 42 42 42 3C 00

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

```
0B90 1C 0A 7F 0A 1C 00
  *
 ***
* * *
*****
* * *
  *
  *
```

```
0B96 00 00 00 00 7F 00
*
*
*
*
*
*
*
```

```
0B9C 18 3C 7E 3C 18 00

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

```
0BA2 08 08 08 7F 08 08
   *
   *
   *
******
   *
   *
   *
```

```
0BA8 55 2A 55 00 00 00
* *
 *
* *
 *
* *
 *
* *
```

```
0BAE 00 00 00 7F 00 00
*
*
*
*
*
*
*
```

```
0BB4 08 7C 04 7C 04 00


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

```
0BBA 01 03 07 0F 1F 3F
       ******
        *****
         ****
          ***
           **
            *
```

```
0BC0 00 00 00 7F 7F 7F
          ***
          ***
          ***
          ***
          ***
          ***
          ***
```

```
0BC6 10 10 10 10 10 10




       ******
```

```
0BCC 78 20 10 08 04 00


           *
       *  *
       * *
       **
       *
```

```
0BD2 12 49 24 12 49 24
        *  *
       *  *
         *  *
        *  *
       *  *
         *  *
        *  *
```

```
*********************************
0BD8 A3   movp a,@a              Get
0BD9 A1   mov  @r1,a             matrix column
0BDA 83   ret
*********************************

0BDB 00 00 00 00 00
0BE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0BF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0C00 B814 mov  r0,#14    (20)
0C02 F0   mov  a,@r0
0C03 1207 jb0  07
0C05 8409 jmp  C09
0C07 24E0 jmp  9E0
*********************************Char. conversion
0C09 FA   mov  a,r2              Character
0C0A 0360 add  a,#60     (96)    <160?
0C0C E620 jnc  20                yes----->
0C0E 03C0 add  a,#C0     (192)   <224?
0C10 E634 jnc  34                yes----->
0C12 03E1 add  a,#E1     (225)   =255?
0C14 C61D jz   1D                yes----->
0C16 0309 add  a,#09     (9)     =246?
0C18 9630 jnz  30                no----->
0C1A BAE0 mov  r2,#E0    (224)
0C1C 83   ret
0C1D BADE mov  r2,#DE    (222)
0C1F 83   ret
0C20 0340 add  a,#40     (64)    <96?
0C22 E634 jnc  34                yes----->
0C24 03FA add  a,#FA     (250)   =102?
0C26 962B jnz  2B                no----->
```

```
0C28 BAE1 mov  r2,#E1    (225)
0C2A 83   ret
0C2B FA   mov  a,r2
0C2C 0360 add  a,#60     (96)
0C2E 8433 jmp  C33
0C30 FA   mov  a,r2
0C31 03C0 add  a,#C0     (192)
0C33 AA   mov  r2,a
0C34 83   ret

0C35 00 00 00 00 00 00 00 00 00 00 00
0C40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0C50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0C60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0C70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0C80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0C90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0CA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0CB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0CC0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0CD0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0CE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0CF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
0D00 08 08 08 08 08 08       0D06 20 54 54 3C 40 00


                                     ***
       ******                          *
                                     ***
                                    *  *
                                     ** *
0D0C 7F 48 44 44 38 00       0D12 38 44 44 44 44 00
       *
       *
       * **                          ****
       **  *                        *
       *   *                        *
       *   *                        *
       ****                          ****
0D18 38 44 44 48 7F 00       0D1E 38 54 54 54 08 00
           *
           *
        ** *                         ***
       *  **                        *   *
       *   *                        ****
       *   *                        *
        ****                         ***
0D24 08 7E 09 01 02 00       0D2A 0C 52 52 52 3C 00
         **
        *  *                         ***
        *                           *   *
       ***                          *   *
        *                            ****
        *                               *
        *                            ***
```

```
0D30 7F 08 04 04 78 00         0D36 00 44 7D 40 00 00
       *                                 *
       *
       * **                             **
       **  *                             *
       *   *                             *
       *   *                             *
       *   *                            ***
0D3C 00 20 40 3D 00 00         0D42 7F 10 28 44 00 00
          *                            *
                                       *
          *                            *  *
          *                            * *
          *                            **
        * *                            * *
         *                             *  *
0D48 00 01 7F 40 00 00         0D4E 7C 04 38 04 78 00
        **
         *
         *                             ** *
         *                             * * *
         *                             * * *
         *                             * * *
         **                            * * *
0D54 04 78 04 04 78 00         0D5A 38 44 44 44 38 00


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

```
0D60 7E 12 12 0C 00 00

***
*  *
*  *
***
*
*

0D66 0C 12 12 7E 00 00

 ***
*  *
*  *
 ***
   *
   *

0D6C 7C 08 04 04 08 00


* **
**  *
*
*
*

0D72 48 54 54 54 20 00


 ***
*
 ***
    *
 ***

0D78 04 3F 44 44 20 00
 *
 *
****
 *
 *
 *  *
  **

0D7E 3C 40 40 3C 40 00


*  *
*  *
*  *
*  *
 ** *

0D84 0C 30 40 30 0C 00


*   *
*   *
 * *
 * *
  *

0D8A 3C 40 30 40 3C 00


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

```
0D90 44 28 10 28 44 00

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

```
0D96 4C 50 50 3C 00 00

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

```
0D9C 44 64 54 4C 44 00

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

```
0DA2 08 08 08 7F 08 08
          *
          *
          *
       ******
          *
          *
          *
```

```
0DA8 55 2A 55 00 00 00
       * *
        *
       * *
        *
       * *
        *
       * *
```

```
0DAE 00 00 00 7F 00 00
          *
          *
          *
          *
          *
          *
          *
```

```
0DB4 73 73 0C 0C 73 73
       **  **
       **  **
         **
         **
       **  **
       **  **
       **  **
```

```
0DBA 24 49 12 24 49 12
         *  *
          *  *
        *  *
         *  *
          *  *
        *  *
         *  *
```

```
*********************************
0DC0 A3   movp a,@a               Get
0DC1 A1   mov  @r1,a              matrix column
0DC2 83   ret
*********************************

0DC3 00 00 00 00 00 00 00 00 00 00 00 00 00
0DD0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0DE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0DF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

*********************************Linefeed & step # in
                                  memory location 16
0E00 B910 mov  r1,#10    (16)
0E02 F1   mov  a,@r1
0E03 A9   mov  r1,a
0E04 99FD anl  p1,#FD    (253)    Linefeed motor on
0E06 D437 call E37                Delay
0E08 D424 call E24                Motor advances one step
0E0A E90E djnz r1,0E
0E0C C41D jmp  E1D
0E0E D43B call E3B                Delay
0E10 C419 jmp  E19
0E12 D43F call E3F                Delay
0E14 E918 djnz r1,18
0E16 D43F call E3F                Delay
0E18 19   inc  r1
0E19 D424 call E24                Motor advances one step
0E1B E912 djnz r1,12
0E1D D437 call E37                Delay
0E1F 8902 orl  p1,#02    (2)      Linefeed-motor off
0E21 E5   sel  mb0
0E22 A4EE jmp  5EE
```

```
********************************* Linefeed motor moves
                                        one step back
0E24 0A   in   a,p2
0E25 122C jb0  2C
0E27 3234 jb1  34
0E29 8A02 orl  p2,#02    (2)
0E2B 83   ret
0E2C 3231 jb1  31
0E2E 9AFE anl  p2,#FE    (254)
0E30 83   ret
0E31 9AFD anl  p2,#FD    (253)
0E33 83   ret
0E34 8A01 orl  p2,#01    (1)
0E36 83   ret
*********************************Delay
0E37 2330 mov  a,#30     (48)
0E39 C441 jmp  E41
0E3B 2351 mov  a,#51     (81)
0E3D C441 jmp  E41
0E3F 23A7 mov  a,#A7     (167)
0E41 62   mov  t,a
0E42 55   strt t
0E43 1645 jtf  45
0E45 1649 jtf  49
0E47 C445 jmp  E45
0E49 65   stop tcnt
0E4A 83   ret
********************************* Character conversion
0E4B FA   mov  a,r2                Character --
0E4C 031F add  a,#1F     (31)      =225?
0E4E 9653 jnz  53                  no----->
0E50 BA46 mov  r2,#46    (70)
0E52 83   ret
```

```
0E53 0301 add  a,#01     (1)      =224?
0E55 9658 jnz  58                 no----->
0E57 83   ret
0E58 0320 add  a,#20     (32)     =192?
0E5A C657 jz   57                 yes----->
0E5C E662 jnc  62                 <192? yes----->
0E5E 0360 add  a,#60     (96)
0E60 AA   mov  r2,a
0E61 83   ret
0E62 0306 add  a,#06     (6)      =186?
0E64 9669 jnz  69                 no----->
0E66 BAE2 mov  r2,#e2    (226)
0E68 83   ret
0E69 0311 add  a,#11     (17)     =169?
0E6B 9657 jnz  57                 no----->
0E6D BAE3 mov  r2,#E3    (227)
0E6F 83   ret
**********************************
0E70 03C0 add  a,#C0     (192)
0E72 C676 jz   76
0E74 04DF jmp  8DF
0E76 04E8 jmp  8E8
**********************************Test address
0E78 F0   mov  a,@r0              --already addressed??
0E79 9295 jb4  95                 yes--> Does the secondary
0E7B 0A   in   a,p2               address have printer ad-
                                    dress of 4??
0E7C D282 jb6  82                 yes----->
0E7E 23DB mov  a,#DB     (219)
0E80 C484 jmp  E84
0E82 23DC mov  a,#DC     (220)
0E84 6A   add  a,r2               Proper LISTENER-address?
0E85 C68A jz   8A                 yes----->
```

```
0E87 E5   sel  mb0                Otherwise,
0E88 E4C8 jmp  7C8                return
*********************************Printer is addressed
0E8A 2310 mov  a,#10     (16)
0E8C 40   orl  a,@r0              LISTENER-FLAGS
0E8D A0   mov  @r0,a
0E8E A5   clr  f1                 set
0E8F B5   cpl  f1
0E90 C4AE jmp  EAE
*********************************
0E92 E5   sel  mb0
0E93 E4BA jmp  7BA
*********************************Check secondary address--
0E95 B292 jb5  92                 does it already exist? yes>
0E97 2320 mov  a,#20     (32)
0E99 40   orl  a,@r0
0E9A A0   mov  @r0,a
0E9B FA   mov  a,r2
0E9C 0396 add  a,#96     (150)
0E9E C6BE jz   BE
0EA0 0302 add  a,#02     (2)
0EA2 C6AE jz   AE
0EA4 17   dec  a
0EA5 C6B4 jz   B4
0EA7 17   dec  a
0EA8 C6BE jz   BE
0EAA 0306 add  a,#06     (6)
0EAC 96BE jnz  BE
0EAE 23F7 mov  a,#F7     (247)
0EB0 5C   anl  a,r4
0EB1 AC   mov  r4,a
0EB2 C4B8 jmp  EB8
0EB4 2308 mov  a,#08     (8)
```

```
0EB6 4C   orl  a,r4
0EB7 AC   mov  r4,a
0EB8 237C mov  a,#7C     (124)
0EBA 50   anl  a,@r0
0EBB A0   mov  @r0,a
0EBC C4C2 jmp  EC2
0EBE 2380 mov  a,#80     (128)
0EC0 40   orl  a,@r0
0EC1 A0   mov  @r0,a
0EC2 E5   sel  mb0
0EC3 E4BA jmp  7BA
*****************************END*****************************
```

The memory registers remaining (from here to $0FFF) read only $00.

{This page left blank intentionally}

8.0 THE VIC-1520 PRINTER-PLOTTER

8.1 Usefulness and Operation

It is important to go through the workings of this device, since, as with any piece of equipment, it can be used to best advantage if it's thoroughly understood. For openers, let's have a look at some technical data:

4 ballpoint pens (black, blue, green, red)
4 different character sizes
10, 20, 40 or 80 characters per plotter line
13 characters per second in the smallest print mode
96 different characters
50 character steps per centimeter
480 character steps in x-direction
+/- 999 character steps in y-direction
264 character steps per second
5 centimeters character distance per second (x- or y-direction)
7 centimeters character distance per second in 45-degree angles
11.4 centimeter-wide paper (4.5 inches)

These statements should be enough to give you an idea of the plotter's capabilities. This device is a small miracle of technology. For example, you already know about its fine resolution from the list above: The plotter can draw a square in 480x480 steps (230,400 points), with each point independent of one another. The printing speed is nothing great--about 5 cm. per second, so any plotting takes a good deal of time.

One thing that you may have found confusing in the above list is the difference in printing speeds: The standard x- and y- registers run at 5 cm per second, while the speed in 45-degree angles increases to 7 cm per second. Although this may seem contradictory, we can explain it in mathematical terms:

Imagine a line drawn for one second in the x-register, then a connecting line drawn for one second in the y-register. We have here a pair of 5 cm lines drawn perpendicular to one another, with a resulting 90-degree line. Now we turn to the Pythagorean Theorem -- we have a right angle, two sides, and need to calculate the length of the missing side.

The theorem used is $X^2+Y^2=Z^2$
--or, with our numbers: $5^2+5^2=Z^2$

The solution is 50, or about seven cm. The combination of both character times leaves us with a time of about one second to print this longer line. This concept is important for understanding the programming of the plotter. Once you're through typing in the programs which follow, you'll understand thoroughly the workings of this device.

Next, you should GENTLY remove the protective cover on the plotter, so that we can take a short tour of the machine's interior.

The first thing you'll probably see is the drum containing the ballpoint pens, which controls the pen(s) contact with the paper. There is another "mechanical part" that will make or break the plotter's accuracy -- a complicated gearing set to the left and right of the rubber roller, which controls the roller's movement. You see, the drum can only move in the x-direction; the paper itself must move in the y-direction.

On the roller, you'll find a needle-tractor drive (with points so tiny you'll barely be able to see them), which accurately moves the paper without any damage to the paper. The last item in our tour of the plotter's "innards" is a small green block in the right-hand corner of the drive. This is a small electromagnet which places pen to paper, and removes a pen when ordered to do so.

This should be enough technical talk to give you the basics of the 1520 plotter. However, there are some rules of etiquette to be followed when working with this machine.

- The plotter is sturdily constructed, but don't let any curious souls play with the delicate cables and gears.
- Don't turn the roller except when absolutely necessary (e.g., when putting in a new roll of paper).
- Never turn the pen-drum by hand.
- Mechanical parts shouldn't be handled, or the accuracy of the plotter may go downhill.
- The printer shouldn't be operated without paper.
- It should be self-evident that the printer shouldn't be unplugged while printing.

Now, here are some hints on the care and feeding of your 1520 plotter:

- Dust is a printer's worst enemy; it slows up the action, and therefore fouls up the functions of the machine.

- If the roller has been marked up by pens, immediately remove the cover and wipe the roller with a damp cloth.
- In the unlikely event of a paper jam, take the cover off and remove the paper by hand (DO NOT PRESS THE PAPER-FEED KEY!!!).

If you look at the plotter in terms of its basic features, you'll notice from the preceding materials that there's more to this machine than simple geometric figures. As with so many aspects of life, you get out of the plotter what you put into it. This chapter will help you learn what to put into the machine--you'll find simple programs, possible solutions to some difficulties, and suggestions to aid you in plotter programming, all in the pages to follow.

The examples to come make no claim to completeness. They simply scratch the surface of what the plotter can do-- but will give you a lot of experience on the device.

One thing before we start: The Plotter can only run using specific program routines, which we will list soon, along with the plotter command set.

The 1520 plotter falls under the category of "intelligent" peripheral devices. That is, no memory within the Commodore 64 (or VIC-20) is set aside for controlling the plotter: Rather, the plotter has its own memory, operating system, and command set. Communicating with the plotter is done much like the other devices (1541 disk drive or Datasette), using OPEN, PRINT# and CLOSE. These commands are combined with addresses, and can be diversified still more using secondary addresses.

- OPEN

This command opens a channel to the device numbered. After OPENing the device, you can instruct the device to perform certain tasks, either in direct mode or program mode.

The OPEN command is usually followed by a set of numbers separated by commas (either a set of two or three numbers, depending on the work being asked of the device):

OPEN nr, dv, sc

nr represents the number later used in PRINT# or CLOSE commands. This number can be anywhere from 1 to 255, but the numbers must match in later commands (e.g., OPEN 15,8,15, --- PRINT#15, --- CLOSE 15), or you'll get a FILE NOT OPEN error message. We'll discuss this number a little later.

dv stands for the number of the device: Each peripheral has its specific device number, and the user cannot change this number on the 1520 plotter. The 1520 plotter, for example, holds the device number 6 (so as not to confuse it with the tape drive [1], the printer [4], or the disk [8]). You'll find 6 to be the second number in the OPEN commands of every example.

One number remains -- the secondary address sc. This number gives specific information to the 1520 concerning special "extras" that the user might want in printing. You can choose that feature now for later printing. Below are the secondary addresses used by the 1520 plotter:

sc=0 Normal printing of text or lists
sc=1 Execute a character routine
sc=2 Control the color drum
sc=3 Choice of character size
sc=4 Turn the character 90 degrees.
sc=5 Choose type of line desired
sc=6 Switches plotter between upper-case, lower-case and graphic modes
sc=7 Returns plotter to starting position, clearing all previously stored data from plotter memory.

With that in mind, here are a few sample OPEN commands, and what they'll do:

OPEN 34,6,2 opens the file number 34 relating to the plotter device (6), and controls the operation of the color drum.

OPEN 1,6,1 opens file number 1, device number 6, and prepares the device for working through the character routine.

OPEN 99,9,9 doesn't do anything for the plotter -- 99 could conceivably be used as the first number, but device # 9 doesn't open the plotter (rather, for those who own two disk drives, the second drive will open), and a secondary address of 9 is nonsensical. You should receive a DEVICE NOT PRESENT error, unless you have two disk drives.

Complete explanations of the secondary addresses will be offered later in this book. The plotter documentation is somewhat sketchy on these numbers, and even ignores some secondary addresses.

- PRINT#

This command executes instructions given to the plotter. It is responsible for sending data to the plotter to be put on paper, as well as selecting the color in which to print.

The command usually sounds like this:

PRINT# nr, command

Remember the previous paragraphs? nr is the number with which you had OPENed the file, which can lie between 1 and 255. In order to use PRINT# properly, the value nr after OPEN nr must equal the nr in PRINT# nr. In other words, if you open a file with OPEN 1,6,1, you would use PRINT# 1 in conjunction with OPEN 1.

The command mentioned above in PRINT# nr, command should need no detailed explanation. If you're still unclear on this item, please see the section on secondary addresses.

One thing barely mentioned in the C-64 handbook is the abbreviation for PRINT#: Logic would tell you to use the abbreviation ?#, but this will give you a syntax error. Remember to abbreviate PRINT# by using P[shift-R], and you'll have no problems in that respect.

- CLOSE

This command closes an OPENed file. As with PRINT#, CLOSE uses the value nr, with the syntax

CLOSE nr

So, you would close a file OPENed with OPEN 2,6,2 by using CLOSE 2. Now, after closing the file, try typing PRINT# 2,2; the plotter can't respond, and the computer will give you a FILE NOT OPEN message.

- CMD

Basically, CMD is used by the plotter for listing out programs and routines. How can we list a program? We can do so in two ways. First, after loading the program, type in these commands in direct mode (not program mode).

```
OPEN 4,6
CMD 4
LIST
PRINT# 4
CLOSE 4
```

CMD can also be used in program mode: Here is a short program routine which shows CMD in this application.

```
10 OPEN4,6
20 CMD4,"PLOTTER 1520"
30 LIST
```

Line 10 opens the plotter for writing, line 20 operates under the CMD 4, command. In our example the text string, placed in quotes, is printed. Once the routine has finished listing, you'll have to enter PRINT# 4 and CLOSE4 by hand. Or, if you prefer, type it into the program:

```
40 PRINT#4
50 CLOSE4
```

RUN the routine. Once it has run, input CMD 4, "PLOTTER 1520", but be forewarned that you won't get an error message, despite the file having been closed in line 50. There's nothing wrong with your program, or even with your plotter. A peculiarity of the 64's operating system causes the program to break when a LIST is called for, rendering lines 40 and 50 useless, and requiring you to close the file by hand.

8.2 Secondary Addresses on the VIC-1520

You'll recall that the secondary address is the last number in the address sequence. It can be a number between 0 and 7, and fulfills certain tasks when called with PRINT# nr. If you've forgotten which secondary address is which, flip back a few pages to the paragraph on OPEN, where you'll find a complete list of these numbers. Now we'll take a short look at these addresses, and some examples.

a) sc=1

This number activates the command set for the draw routine in the plotter, which is especially of interest to the beginner.

As you probably know, the secondary address is sent to the plotter with the OPEN command. In our example, let's use OPEN 1,6,1. Now we can use the command set in the plotter, which is started by PRINT#. The proper form for input looks like this:

```
PRINT# 1, "COMMAND"; X-data;Y-data [or]
PRINT# 1, "COMMAND", X-data,Y-data [or]
PRINT# 1, "COMMAND" X-data,Y-data
```

The designations for X-data and Y-data can also be separated by a semicolon, comma or a space. There are a total of six instructions acknowledged by the plotter.

1. H (Home) brings the ballpoint pen drum into start position, i.e., at the left side of the roller. This location is the "absolute zero-point"-- X=0; Y=0. Two

additional commands can be given when at zero-point; "D" and "M".

2. M (Move) shifts the drum from the home position at the left side of the plotter. The starting point always remains x=0;y=0. The pen is withdrawn from the paper, and the drum will move to whatever position you tell it to (x=0-480: y=+999\-999), but it will not draw.

3. D (Draw) is essentially the same as M, but a pen specified by you is put into operation, so the plotter will draw at the requested coordinates. D will always start from the home point (0/0).

Before giving you some tangible examples, let's finish the list of commands.

4. I There is a relative zero-point that exists in addition to the absolute zero (0-0), which you can change at will. Simply move the drum to the position of your choice (see M) and input PRINT# 1, "I". Whatever coordinate that the plotter is at, becomes the "relative home". Move the plotter to another position, and PRINT# 1, "I", again; the old "relative zero" coordinates have changed to those of the new position. As with H, two other commands work in concert with I.

5. R is the equivalent of the M-command, i.e., the pen moves away from the paper and the drum moves to the given x/y coordinates. The main difference between R and M is that R is used with the relative home point.

6. J is the draw command used with the relative zero-point, and operates much like D: The pen is put to paper, and the desired line is drawn.

Below is a routine to show you the plotter's command set in action. Here we wanted to make a triangle, and show you the differences between absolute and relative drawing routines.

```
10 OPEN1,6,1
20 PRINT#1,"M";100;0
30 PRINT#1,"D";300;0
40 PRINT#1,"D";200;100
50 PRINT#1."D";100;0
60 PRINT#1,"H"
70 PRINT#1,"M";0;-100
80 CLOSE1
```

Start the program; you should get the above mentioned picture. Please note that the commands in the above program are absolute ones (H,M,D). Line 20 moves the head to the position 100/0. Lines 30-50 draw the triangle, whereby all positions are calculated from 0/0. Lines 60-70 shift the paper up so that your work is visible. You've probably figured out that complex drawings are not possible with absolute nullpoint as a base; rather, they're a mixture of absolute and relative homing. Now, try this program:

```
10 OPEN1,6,1
20 PRINT#1,"M";100,0
30 PRINT#1,"I"
40 PRINT#1,"J";200;0
50 PRINT#1,"J";100;100
```

```
60 PRINT#1,"J";0;0
70 PRINT#1,"R";-100;-100
80 CLOSE1
```

The result looks the same as the above program, but a different method was used to get that result. The starting point is still 100/0 (line 20); however, this coordinate is immediately turned into a relative nullpoint with PRINT# 1, "I" (line 30). Lines 40-60 draw the triangle ("J"). The same end is reached a little more quickly.

We recommend that you work in mixed commands as much as possible during this chapter, so that these procedures will become second nature to you. Let's try the triangle program again, only this time, using mixed commands:

```
10 OPEN1,6,1
20 PRINT#1,"R";100,0
30 PRINT#1,"I"
40 PRINT#1,"J";200;0
50 PRINT#1,"J";100;100
60 PRINT#1,"J";0;0
70 PRINT#1,"R";0;-100
80 CLOSE1
```

Line 20 has an R command rather than an M. This can be done since both absolute and relative nullpoints are identical, and thus no programming changes are necessary. Remember to use the combinations of commands most convenient to you and the computer. This is important as figures become more complicated, or when mathematical formulas are involved. A great help will be careful study of this chapter; don't rush through!

b) sc=2

This secondary address lets you exchange the pen for one of a different color within a program. This address activates the COLOR CHANGE key on your plotter. You'll need to open the file with OPEN nr, 6, 2 (nr =1 to 255). PRINT#2 allows you to choose from one of four colors. The colors are represented by the following numbers:

```
BLACK PEN = 0,4,8,12,16  ..... (0+N*4<255)
BLUE  PEN = 1,5,9,13,17  ..... (1+N*4<255)
GREEN PEN = 2,6,10,14,18 ..... (2+N*4<255)
RED   PEN = 3,7,11,15,19 ..... (3+N*4<255)
```

If you choose a number larger than 255, the default color will be in effect (black). For later reference, remember that black is always the starting color, based on the mathematical processes above.

You can choose your pen color through a number in quotes, a normal number, or a variable. This next program shows the colors available. A hardcopy of this will help you remember the numeric sequence.

```
100 REM=====1520 SETUP=========
110 OPEN4,4
120 OPEN2,6,2
130 REM====READ NAMES=======
140 FOR F=0 TO 3
150 READ F$(F)
160 NEXT
170 REM=====PRINT COLOR TO PLOTTER=====
180 FOR F=0 TO 3
```

```
190 PRINT#2,F
200 PRINT#4,F$(F)
210 NEXT
220 REM======DATA=============
230 DATABLACK,BLUE,GREEN,RED
240 REM=====END============
250 CLOSE2
260 CLOSE4
```

The first section of the program stores the names of the four colors in an array. The second section (from line 180) prints the name of the respective color.

The second program in this set takes the color number which you input (NB: between 0 and 255), and prints in that color, supplying the name of the color. The final program in the sequence reverses the procedure, giving you all possible numbers for each color.

```
100 REM====READ DATA===================
110 FORF=0TO3
120 READF$(F)
130 NEXT
140 REM====INPUT/CALCULATION===========
150 INPUT"sCOLOR NUMBER";K$
160 K=VAL(K$)
170 K1=VAL(K$)
180 IFK>255ORK<0THEN150
190 IFK>3THENK=K-4
200 IFK>=4THEN190
210 REM====OUTPUT=====================
220 PRINT
230 PRINTK1"= "F$(K)
240 REM=====RESTART===================
250 FORN=1TO1000
260 NEXT
270 RESTORE
280 GOTO100
290 REM====DATA=======================
300 DATABLACK,BLUE,GREEN,RED

READY.
```

```
100 REM====INPUT=======================
110 PRINT"sWHICH COLOR?"
120 PRINT
130 PRINT"BRLrACK RBrLUE RGrREEN RRrED"
140 PRINT
150 GETA$:WAIT203,63
160 REM====ANSWER======================
170 IFA$="L"THENK=0:GOTO230
180 IFA$="B"THENK=1:GOTO230
190 IFA$="G"THENK=2:GOTO230
200 IFA$="R"THENK=3:GOTO230
210 GOTO150
220 REM====CALCULATION=================
230 FORF=0TO63
240 G=K+F*4
250 PRINTG,:
260 NEXT
270 REM====DO IT AGAIN=================
280 PRINT
290 PRINT"<HIT A KEY>"
300 WAIT203,63
310 GOTO100

READY.
```

c) sc=3

This address controls the print routine for the character set built into the plotter, offering you four different-sized character sets, changeable within a line. Open files with OPEN nr,6,3, and change character sets using PRINT#3, number [of print size]. The numbers are figured under the same mathematical conditions as the pen colors in PRINT#2; however, for the sake of brevity, we'll use 0,1,2,3.

The number can be input within quotes, or represented by a variable. The following is the basic information on these numbers:

PRINT#3,0 produces the smallest character size of 80 characters per line.

PRINT#3,1 gives normal printing (40 characters per line).

PRINT#3,2 yields middle-sized lettering (20 characters per line).

PRINT#3,3 is the largest character set at 10 per line.

The default (i.e., when you turn the plotter on) is normal-sized black characters. To change the default, use the routine below.

```
100 REM====1520 SETUP==================
110 OPEN4,6
120 OPEN3,6,3
130 PRINT#3,0
140 REM====CHARACTER SIZES=============
150 FORZ=0TO3
160 PRINT#3,Z
170 REM====PRINT=======================
180 PRINT#4,"PLOTTER"
190 Y=X+2
200 IFX<YTHENX=X+1:PRINT#4:GOTO200
230 NEXTZ
240 REM====END=========================
250 PRINT#3,0
260 CLOSE3
270 CLOSE4

READY.
```

The following will be printed when you run this program:

PLOTTER

PLOTTER

PLOTTER

PLOTTER

Now that we've covered some basics of characters, let's go on to some technical thoughts. How is a character drawn on a plotter, anyway? To answer this question, let's take the letter Y as an example. On the 1520, Y is drawn from the bottom to the upper right. Beginning at the bottom, the pen draws to the upper-left-hand corner of the letter, jumps to the right with the pen withdrawn, sets the pen down, and draws the last section of the letter.

Other, more complicated letters have a structure that allows for plotting without the pen leaving the paper, e.g., the @. Still others require much movement of the pen (on paper and off), with the philosophy of taking the shortest route.

One good example is the #: Starting from the left, the plotter draws the upper horizontal and the right vertical in more-or-less one movement. Next, the drum and paper shift, and the left vertical and bottom line is drawn.

We're telling you all this to remind you of why the plotter is so slow; with a character speed of 13 characters per second at the smallest character size, it's easy to become impatient with the plotter. Remember how much work it has to accomplish to draw these characters, though.

Another thing to bear in mind is the fact that the plotter was never specifically designed for lettering; that's best accomplished with a standard dot-matrix printer. The plotter has abilities that no dot-matrix machine has, though!

d) sc=4

The 1520 plotter has some other functions, which are controlled by secondary address 4. For example, this address will turn all characters 90 degrees in the printout. This may seem an unnecessary move -- but many problems can be solved with this turn. For example, say you're printing a chart which requires a longer-than normal axis: The X-axis can be handled in normal mode, while the Y-axis is covered in the turnaround of characters.

Input format looks like this:

```
OPEN nr,6,4
PRINT# 4,number
```

Number can be anywhere between 0 and 255, although the two numbers usually used are 0 and 1. The default value is 0, which represents normal mode. If, however, you typed PRINT#4,1, you would get that 90 degree turn. All ASCII characters to follow will automatically printed 90 degrees to the right, following the previously stated color and character size. The designation number can be given as a variable, or as a number in quotes.

This next program will give you a good look at what secondary address 4 can do -- feel free to experiment.

```
100 REM====1520 SETUP==================
110 OPEN5,6
120 OPEN4,6,4
130 PRINT#4,0
140 REM====PRINT=======================
150 PRINT#5,"1520 PLOTTER"
160 PRINT#4,1
170 PRINT#5,"1520 PLOTTER"
180 REM====END=========================
190 PRINT#4,0
200 CLOSE4
210 CLOSE5

READY.
```

This is the simplest method of turning characters toward the X-axis, but also the most awkward. In order to decipher the string "PLOTTER 1520", you will have to read from bottom to top. Obviously, reading text from top to bottom (or left-to-right) is preferable, so we'll have to do some reworking. One way of fixing the program is to reverse the text in line 170, i.e., "0251 RETTOLP". The text will print correctly, but there must be a better way, for the expenditure of time involved in spelling words in reverse.

The next program shows one way of solving this problem.

```
100 REM====1520 SETUP==================
110 OPEN5,6
120 OPEN4,6,4
130 OPEN1,6,1
140 PRINT#4,1
150 REM====CALCULATE CHARACTERS========
160 X$="1520 PLOTTER"
170 FORX=LEN(X$)TO1STEP-1
180 Y$=MID$(X$,X,1)
190 REM====PRINT CHARACTERS============
200 PRINT#1,"M",20*(LEN(X$)-X);0
210 PRINT#5,Y$;:
220 NEXT
230 REM====END=========================
240 PRINT#4,0
250 PRINT#1,"M";0;0
260 CLOSE1
270 CLOSE4
280 CLOSE5

READY.
```

The secret to this program lies in lines 150-220: First, the text string "PLOTTER 1520" is stored in the variable X$ (line 160). A FOR-NEXT loop then picks up in line 170, which views the characters from right-to-left. Next, the text is printed in what the computer believes to be right-to-left, thanks to the FOR-NEXT loop (STEP-1; see line 170), but of course, the text prints normally to us. This procedure is dependent on LEN(X$)-X and the MID$ command.

Look again at the parentheses; LEN(X$) signifies the length of the text, nothing more. It writes from left to right from home point. This example must allow for subtraction on the LEN(X$) using a constant, in this case, 20. Line 210 gives us our printout. Remember, though, to put the semicolon in, or the text will be printed diagonally!

As we've said, this is one way of solving the problem--see the plotter handbook for another method.

Let's depart our work on the X-axis, and look to the Y-axis. If you typed in the last program, don't waste your time typing this one in from scratch; these programs are designed with similar routines and line numbers, so that changing one program into another is a simple matter.

```
100 REM====1520 SETUP==================
110 OPEN5,6
120 OPEN4,6,4
130 OPEN1,6,1
140 PRINT#4,1
150 REM====CALC. CARACTERS=============
160 X$="1520 PLOTTER"
170 FORX=1TOLEN(X$)
180 Y$=MID$(X$,X,1)
190 REM====PRINT CHARACTERS============
200 PRINT#5,Y$
210 NEXT
220 REM====END=========================
230 PRINT#4,0
240 PRINT#5
250 CLOSE1
260 CLOSE4
270 CLOSE5
```

Each character in "PLOTTER 1520" is individually read and printed (lines 150-210). So, now you have three methods of performing the same task--you may want to review the preceding programs.

We will be confronted with turning the text later on, when we'll be working on a statistics program with a bar chart.

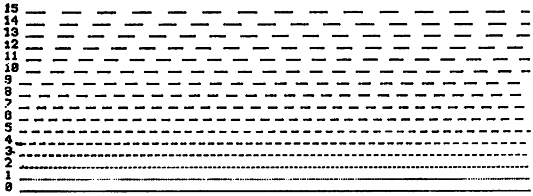
e) sc=5

One of the most useful functions in the 1520's operating system is the line function, called with secondary address 5. This address allows you to call and plot 16 different types of lines. These lines can vary in appearance from a solid line to a series of dashes. The file is opened for this using OPEN nr,6,5.

Next follows PRINT#5, number command, with a number between 0 and 255, but the most useful numbers for this address are 0-15.

Default is 0 (solid line). Inputting a value from 1 to 15 will print a line in varying stages of breaking. Below are examples of such lines and the program to produce them.

```
100 REM====1520 SETUP==================
110 OPEN4,6
120 OPEN5,6,5
130 OPEN1,6,1
140 PRINT#5,0
150 REM====INPUT=======================
160 INPUT"sWHICH LINE NUMBER";A$
170 A=VAL(A$)
180 IFA<0ORA>15THEN160
190 PRINT
200 INPUT"LENGTH IN CM (MAX. 8 CM)";B$
210 B=VAL(B$)
220 IFB<0ORB>8THENPRINT"qq":GOTO200
230 REM====DRAWING=====================
240 L=50*(B+1)
250 PRINT#4,A;
260 PRINT#5,A;
270 PRINT#1,"D";L;0
280 PRINT#5,0
290 PRINT#4
300 REM====END=========================
310 CLOSE1
320 CLOSE5
330 CLOSE4

READY.
```



Lines 100-140 set the plotter for input of line values, and opens the necessary files. Line 160 waits for your input of a number from 0 to 15. Line 160 has something else of interest -- instead of the reverse heart signifying CLEAR/HOME, the 1520 plotter prints an underlined letter s. Looking ahead at line 220, the underlined q's represent two cursor-down characters, following a request for line length. The lines are printed out by lines 240-290. If you measure the lines with a metric ruler, you'll find that the measurements are quite accurate.

The line function has almost limitless possibilities for creating figures, as you'll see in the next program.

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 OPEN5,6,5
130 PRINT#5,5
140 REM====DRAW========================
150 PRINT#1,"M";100;0
160 PRINT#1,"I"
170 PRINT#1,"J";200;0
180 PRINT#1,"J";100;100
190 PRINT#1,"J";0;0
200 PRINT#1,"R";-100;-100
210 REM====END=========================
220 PRINT#5,0
230 CLOSE1
240 CLOSE5
```

This program draws a familiar figure (the triangle), which we used a bit earlier in relative and absolute nullpoints. This routine draws identical sides with a specific type of line -- dotted lines.

f) sc=6

You've probably run across upper- and lower-case lettering in your playing around on Commodore computers. The plotter can also do this lettering. As you've seen, the plotter has no graphic characters per se, so the device constantly operates in upper-case only, unless otherwise told. As a matter of fact, holding down the SHIFT key gives you lower-case lettering (the opposite of what we're accustomed to!). Nothing changes in the plotter system by typing in PRINTCHR$(14), just the computer changes case. Secondary address 6, however, allows us to control this problem. There are but two additional numbers you have to know:

1) 0= upper-case with SHIFTed lower-case (default).
2) 1= lower-case with SHIFTed upper-case.

The input format runs as follows: Open the file with OPEN nr,6,6, and proceed with PRINT#6, dg, with dg equalling either 0 or 1. If you send PRINT#6,1, you'll immediately be in lower-case mode, and be able to print upper-case characters by pressing SHIFT.

Now you can use the plotter and computer just as you would a typewriter. Incidentally, it makes no difference to the plotter whether you have a graphic symbol or shifted character onscreen (depends on the mode, though) -- it will

print an upper-case character either way. To return the plotter to its original status, send PRINT#6,0.

To make the differences between the two modes clear, try out the program below. We suggest that you switch the C-64 to lower-case mode before typing the routine in, just to make finding graphic characters and such easier on you. The listing itself is in lower-case mode (1).

```
100 rem====1520 setup==================
110 open4,6
120 open6,6,6
130 print#6,0
140 rem====text========================
150 print#4,"6,0 = normal mode"
160 gosub270
170 print#6,1
180 gosub270
190 print#4,"6,1 = Lower-case mode"
200 gosub270
210 print#6,0
220 gosub270
230 rem====end=========================
240 close6
250 close4:end
260 rem====plott=======================
270 forx=1to35
280 print#4,chr$(90-x);
290 next
300 print#4
310 return

ready.
```

The following printout will result after running the program:

```
6,0 = NORMAL MODE
YXWVUTSRQPONMLKJIHGFEDCBA@?>=<;:987
yxwvutsrqponmlkjihgfedcba—?>=<;:987
6,1 = Lower-case mode
yxwvutsrqponmlkjihgfedcba—?>=<;:987
YXWVUTSRQPONMLKJIHGFEDCBA@?>=<;:987
```

The result from the program confirms these two modes: Normal mode has upper-case with shifted lower-case, while "lower-case" mode does the exact opposite. We'll need this secondary address a bit later when we create a typing program in direct mode.

g) sc=7

At last, we come to the seventh and final secondary address. To be perfectly honest with you, you won't have much need for this address. OPEN 7,6,7 and PRINT#7, and you'll see: The 1520 immediately puts itself through a self-test. If we input this material again, the system will reset to its original value and delete any data found in memory.

Now that you've read the descriptions in this section, even novice plotter owners should be able to understand, analyze, and even design plotter subroutines. The 1520 is such a versatile instrument that a book the size of this volume could be filled with routines for drawing figures. This, however, isn't everyone's cup of tea. Instead, we'll spend the rest of the book with basic figures, made with routines that you can tailor to your own needs.

Some of the more observant readers have probably noticed that we left out secondary address 0, and started our list of definitions with 1 instead. Essentially, address 0 is responsible for the output of ASCII characters, and the differences between screen and printer. Let's ponder the 1520's "normal" mode first: The following table demonstrates the character sets, both normal and shifted.

! " # $ % &
' ( ) * + ,
- . / 0 1 2
3 4 5 6 7 8
9 : ; < = >
? @ A B C D
E F G H I J
K L M N O P
Q R S T U V
W X Y Z [£
] ↑ ←
! " # $ % &
' ( ) * + ,
- . / 0 1 2
3 4 5 6 7 8
9 : ; < = >
? — a b c d
e f g h i j
k l m n o p
q r s t u v
w x y z | _
△ π □

You'll notice that the character set corresponds with the screen output from CHR$(33) to CHR$(95). Shifted characters which appear onscreen as graphic symbols (from CHR$(161)) appear in lower-case on a plotter printout. Switching in PRINT#6,1 will do the exact opposite -- CHR$(33) will be in lower-case, and the graphic characters will appear in upper-case.

There are no other real differences in output between screen and plotter, with one exception: What about the control characters? What's wrong with the plotter that it doesn't print reverse-video characters?

The C-64 has many reverse-video characters, from the cursor control symbols to the color symbols. We already know that the plotter isn't equipped to print these characters, since too much time would be involved in printing out those little boxes. On the other hand, you might wonder, wouldn't it take almost as long to print out all the symbols in character-string (CHR$(X)) form?

To circumvent this problem, the 1520 plotter prints the control characters as standard characters with an underscore to distinguish them from standard characters (e.g., CURSOR DOWN would be an underlined Q). Here is a chart of these characters, side-by-side with a 1525 printout:

```
"E = function key 1"
"I = function key 2"
"F = function key 3"
"J = function key 4"
"G = function key 5"
"K = function key 6"
"H = function key 7"
"L = function key 8"
"t = delete"
"s = home"
"S = clear"
"q = cursor down"
"Q = cursor up"
"△ = cursor right"
"] = cursor left"
"P = control/blk"
"e = control/wht"
"_ = control/red"
"← = control/cyn"
"£ = control/pur"
"π = control/grn"
"□ = control/blu"
"r = control/rvs on"
"R = control/rvs off"
"A = c=/blk"
"U = c=/wht"
"V = c=/red"
"W = c=/cyn"
"X = c=/pur"
"Y = c=/grn"
"Z = c=/blu"
"[ = c=/yel'
```

```
"e = FUNCTION KEY 1"
"I = FUNCTION KEY 2"
"f = FUNCTION KEY 3"
"J = FUNCTION KEY 4"
"g = FUNCTION KEY 5"
"k = FUNCTION KEY 6"
"h = FUNCTION KEY 7"
"L = FUNCTION KEY 8"
"T = DELETE"
"S = HOME"
"s = CLEAR"
"Q = CURSOR DOWN"
"q = CURSOR UP"
"] = CURSOR RIGHT"
"△ = CURSOR LEFT"
"p = CONTROL/BLK"
"E = CONTROL/WHT"
"£ = CONTROL/RED"
"□ = CONTROL/CYN"
"_ = CONTROL/PUR"
"↑ = CONTROL/GRN"
"← = CONTROL/BLU"
"R = CONTROL/RVS ON"
"r = CONTROL/RVS OFF"
"a = C=/BLK"
"u = C=/WHT"
"v = C=/RED"
"w = C=/CYN"
"x = C=/PUR"
"y = C=/GRN"
"z = C=/BLU"
"[ = C=/YEL'
```

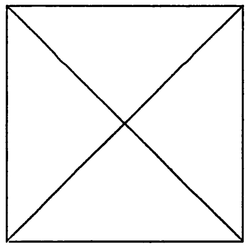
```
"■ = FUNCTION KEY 1"
"■ = FUNCTION KEY 2"
"■ = FUNCTION KEY 3"
"■ = FUNCTION KEY 4"
"■ = FUNCTION KEY 5"
"■ = FUNCTION KEY 6"
"■ = FUNCTION KEY 7"
"■ = FUNCTION KEY 8"
"■ = DELETE"
"■ = HOME"
"■ = CLEAR"
"■ = CURSOR DOWN"
"■ = CURSOR UP"
"■ = CURSOR RIGHT"
"■ = CURSOR LEFT"
"■ = CONTROL/BLK"
"■ = CONTROL/WHT"
"■ = CONTROL/RED"
"■ = CONTROL/CYN"
"■ = CONTROL/PUR"
"■ = CONTROL/GRN"
"■ = CONTROL/BLU"
"■ = CONTROL/RVS ON"
"■ = CONTROL/RVS OFF"
"■ = C=/BLK"
"■ = C=/WHT"
"■ = C=/RED"
"■ = C=/CYN"
"■ = C=/PUR"
"■ = C=/GRN"
"■ = C=/BLU"
"■ = C=/YEL"
```

8.3 Constructing Figures of All Kinds

Now that we've covered some basics, it's time to put the theories into practice. This chapter covers programming a myriad of geometric figures, from squares to arcs and ellipses. Just for the sake of unity, we'll be using measurements in centimeters throughout this segment. The program will always come first, followed by a drawing of the figure, then a description of what the program does and at what line number the action occurs. If the program doesn't work the first time, just keep debugging.

8.3.1 Square

```
100 REM====SQUARE=====================
110 OPEN1,6,1
120 OPEN2,6,2
130 PRINT#2,1
140 REM====DRAW=======================
150 PRINT#1,"M";100;-50
160 PRINT#1,"I"
170 S=200
180 PRINT#1,"J";S;0
190 PRINT#1,"J";S;S
200 PRINT#1,"J";0;S
210 PRINT#1,"J";0;0
220 REM====DIAGONALS==================
230 PRINT#2,3
240 PRINT#1,"J";S;S
250 PRINT#1,"R";S;0
260 PRINT#1,"J";0;S
270 REM====END========================
280 PRINT#1,"R";-100;-100
310 CLOSE1
320 CLOSE2
```



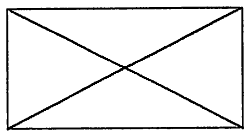
As you can see from the listing, drawing a square is fairly easy to accomplish, since all sides are identical in length. Line 170 determines length of the sides (S), while lines 180 to 210 perform the plotting process. This example uses the latter command letters, which might be worth your reviewing. The next figure, a rectangle, is a bit more complicated, but it requires most of the above program logic.

8.3.2 Rectangle

First, the listing for a rectangle with diagonals:

```
100 REM====RECTANGLE===================
110 OPEN1,6,1
120 OPEN2,6,2
130 PRINT#2,1
140 REM====DRAW========================
150 PRINT#1,"M";100;-50
160 PRINT#1,"I"
170 S=200:H=100
180 PRINT#1,"J";S;0
190 PRINT#1,"J";S;H
200 PRINT#1,"J";0;H
210 PRINT#1,"J";0;0
220 REM====DIAGONALS===================
```

```
230 PRINT#2,3
240 PRINT#1,"J";S;H
250 PRINT#1,"R";S;0
260 PRINT#1,"J";0;H
270 REM====END========================
280 PRINT#1,"R";-100;-100
310 CLOSE1
320 CLOSE2
```



When you compare the square and rectangle programs, you'll immediately see the differences between the two: Line 170 contains two variables for the two pairs of sides, and the draw routine has been altered accordingly. This program is so similar to the previous one, though, that detailed explanations would be all repetition here.

Now we'll try to construct a spatial figure, using a standard cube as a basis. Before you dive into this project, remember that most of the programs in this section have similar structures, mostly to save you typing time, but also to help you grasp the program logic. The readability of these programs will cut down on execution time, however.

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 OPEN2,6,2
130 OPEN5,6,5:PRINT#5,0
140 REM====DRAW=======================
```

```
150 X=100:Y=-50:S=200
160 PRINT#1,"M";X;Y
170 PRINT#1,"I"
180 PRINT#1,"J";S;0
190 PRINT#1,"J";S;S
200 PRINT#1,"J";0;S
210 PRINT#1,"J";0;0
220 X=70:Y=70
230 PRINT#1,"R";S;0
240 PRINT#1,"I";
250 PRINT#1,"J";X;Y
260 PRINT#1,"J";X;Y+S
270 PRINT#1,"J";0;S
280 PRINT#1,"R";-S;S
290 PRINT#1,"J";-S+X;Y+S
300 PRINT#1,"J";X;Y+S
310 REM====DRAW DASHED=================
320 PRINT#5,7
330 PRINT#1,"R";X;Y
340 PRINT#1,"J";-S+X;Y
350 PRINT#1,"J";-S;0
360 PRINT#1,"R";-S+X;Y+S
370 PRINT#1,"J";-S+X;Y
380 REM====END=========================
390 PRINT#1,"M";0;-100
400 CLOSE1
410 CLOSE1
420 CLOSE2
430 CLOSE5
```

In lines 150 and 220 the side-lengths are set into the variables X,Y and S. Our figure is printed using the coordinate variables in PRINT#1,COMMAND, both in solid lines and dotted lines. Any changes to this program mean that you'll have to figure out new lengths mentally. Our routine here is downright uneconomical in its logic, simply because of all the PRINT# statements--watching the pen drum go through all its movements in this program can be pretty amusing. However, now that the program has served its purpose, we can shorten this routine in two ways, leaving us with a more efficient program.

First, we should reduce the number of PRINT# statements, and work on making the pen drum travel as little as possible. We'll put our relative null-point at 100/200, to make things move a bit faster for the pen. The other change we can make in this area is to read the coordinates directly as DATA statements, ridding us of variable output and all those PRINT# calls.

All the 1520 commands can be realized in DATA statements (M,D,J,H). Using our previous program as a pattern, look at it now after these changes:

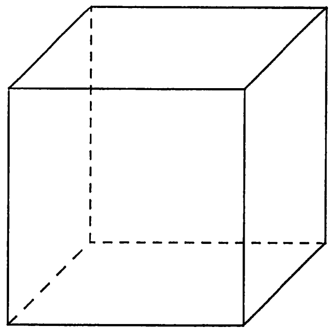
```
100 REM====1520 SETUP=================
110 OPEN5,6,5
120 OPEN1,6,1
130 PRINT#5,0
140 PRINT#1,"M";100;200
150 PRINT#1,"I"
160 REM====READ DATA/LINES=============
170 FORX=0TO7
180 READY,Z
190 PRINT#1,"J";Y;Z
200 NEXT
210 REM====READ DATA/LINES & STROKES===
220 FORX=0TO7
230 READX$,Y,Z
240 IFX=2THENPRINT#5,7
250 PRINT#1,X$,Y,Z
260 NEXT
270 REM====END=========================
280 PRINT#5,0
290 CLOSE1
300 CLOSE5
310 REM====DATA (LINES)================
320 DATA0,-200,200,-200,270,-130
330 DATA270,70,70,70,0,0,200,0
340 DATA270,70
350 REM====DATA(LINES&STROKES)=========
360 DATAR,200,0,J,200,-200,R,270,-130
370 DATAJ,70,-130,J,0,-200,R,70,-130
380 DATAJ,70,70,M,0,-100

READY.
```

Start the program with RUN. You'll get exactly the same figure as before, but with considreably less work than the other program, which means shortened execution time.

There is another procedure we can follow to make these routines run better: If you aren't completely clear on the first steps, reread the last few paragraphs, since most of the programs to follow contain DATA lines.

To shorten our cube routine further, you can cut down the number of READ loops, giving the entire responsibility of drawing the cube to the DATA statements.

```
100 REM====1520 SETUP==================
110 OPEN5,6,5
120 OPEN1,6,1
130 PRINT#5,0
140 PRINT#1,"M";100;200
150 PRINT#1,"I"
160 REM====READ /DRAW DATA=============
170 FORX=0TO15
180 READX$,Y,Z
190 IFX=10THENPRINT#5,7
200 PRINT#1,X$,Y,Z
210 NEXT
220 REM====END=========================
230 PRINT#5,0
240 CLOSE1
250 CLOSE5
260 REM====DATA========================
270 DATAJ,,-200,J,200,-200,J,270,-130
280 DATAJ,270,70,J,70,70,J,,,J,200,0
290 DATAJ,270,70,R,200,,J,200,-200
300 DATAR,270,-130,J,70,-130,J,,-200
310 DATA R,70,-130,J,70,70,M,,-100

READY.
```



Compare this last listing with the first example; you can see that they all perform the same function, but each version is more economical -- better written -- and faster than its predecessor.

So, efficiently-written routines for the 1520 can be made with the following criteria:

- Accurate planning for the figure (drawing)
- Working out the path of least movement for the plotter (worked out on paper first)
- Avoid large numbers of PRINT# statements (use DATA statements instead)
- Try to keep the program readable (REMs and decent program structure)

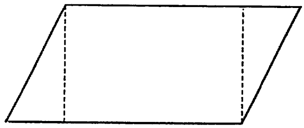
If you keep these rules in mind when designing plotting programs, you'll have more time to enjoy your programming efforts, and explore the farthest borders of your plotter.

The last three programs document the developmental process in "streamlining" BASIC. The remaining programs use these streamlining techniques, i.e., subdividing the program into two READ loops and attaching DATA blocks.

Now we present two examples of simple figures which have a solid place in geometry; the parallelogram and the diamond. Dotted lines represent heights and widths..

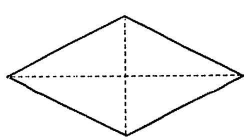
8.3.3 Parallelogram

```
100 REM====PARALLELOGRAM===============
110 OPEN1,6,1
120 OPEN5,6,5
130 PRINT#5,0
140 REM====START POSITION==============
150 PRINT#1,"M";100;0
160 PRINT#1,"I"
170 REM====DRAW PARALLELOGRAM==========
180 FORX=0TO3
190 READY,Z
200 PRINT#1,"J";Y;Z
210 NEXT
220 REM====DRAW HEIGHT LINES===========
230 PRINT#5,3
240 FORX=0TO4
250 READX$,Y,Z
260 PRINT#1,X$,Y,Z
270 NEXT
280 REM====END=========================
290 PRINT#5,0
300 CLOSE1
310 CLOSE5
320 REM====PARALLELOGRAM DATA==========
330 DATA200,0,250,100,50,100,0,0
340 REM====HEIGHT DATA=================
350 DATAR,50,0,J,50,100,R,200,100
360 DATAJ,200,0,M,0,-100

READY.
```



Obviously the program presents no problems, so we'll pass on giving you any detailed explanations. The diamond is equally as simple to make:

```
100 REM====DIAMOND=====================
110 OPEN1,6,1
120 OPEN5,6,5
130 PRINT#5,0
140 REM====STARTING POSITION===========
150 PRINT#1,"M";100;0
160 PRINT#1,"I"
170 REM====READ DATA===================
180 FORX=0TO7
190 READX$,Y,Z
200 IFX=4THENPRINT#5,3
210 PRINT#1,X$,Y,Z
220 NEXT
230 REM====END=========================
240 PRINT#5,0
250 CLOSE1
260 CLOSE5
270 REM====DATA========================
280 DATAJ,100,-50,J,200,,J,100,50,J,,
290 DATAJ,200,,R,100,50,J,100,-50
300 DATAM,,-150

READY.
```



You'll notice that in addition to the figure itself, we've included dotted lines drawn from the diagonals. Also, note that the DATA is read in large blocks (e.g., line 190, where READ is followed by X$, a comma, Y, comma, Z). The remainder of our programs use this form.

8.3.4 Triangle

Before we get on with this figure, let's backtrack a bit and have a look at triangle construction. There are essentially three important forms of triangle:

a) Equilateral
b) Isosceles
c) Right (angle)

Let's begin with a). As its name implies, an equilateral triangle has three equal sides, and three 60 degree angles (you'll recall that all angles in a triangle must add up to 180 degrees). There is a problem in transcribing this to a plotter, though; how do we determine how to plot the sides (the base is no problem), since the Y-direction doesn't really run at an angle?

One small equation will help us around this difficulty:

You know the length of the sides -- if you draw a line perpendicular to the base (from the center of the base) , you get a subdivision of the equilateral triangle into two right triangles standing back-to-back. Now you can calculate the height using that height-line, and fill in the two remaining sides.

There are two ways to calculate the height: You could use the Pythagorean Theorem; or, using the computer, calculate the sine.

If you remember any geometry, you'll remember the Pythagorean Theorem, which helps determine length of a side, or size of an angle, using the remaining materials of the figure. For example:

Side length = 1
Height = H

$H^2+(1/2)^2=1^2$
$H^2=1^2-(1/2)^2$
$H^2=3/4*1^2$

$H=SQR(3/4*1^2)$

This formula gives you the height of an equilateral triangle. Now, let's use a more concrete example -- 4 cm. sidelength, which gives us H=SQR (12), or 3.464. Now, get a sheet of paper, draw a 4-centimeter base, and a center-line of 3.464 cm. The result will give you sides of exactly 4 cm. in length.

A practical drawing program can be formed from these basics. Here's such a program:

```
100 REM====TRIANGLE====================
110 OPEN5,6,5
120 OPEN1,6,1
130 PRINT#1,"M";200;-50
140 PRINT#1,"I"
150 REM====INPUT=======================
160 PRINT"sL = 1 - 8 CM"
170 INPUT"QLENGTH OF SIDES";A$
```

```
180 L=VAL(A$)
190 IFL<1ORL>8THEN160
200 L=L*50
210 REM====CALCULATE================
220 H=SQR(3/4*L↑2)
230 PRINT#1,"J";-L/2;0
240 PRINT#1,"J";0;H
250 PRINT#1,"J";L/2;0
260 PRINT#1,"J";0;0
270 REM====DRAW HEIGHT================
280 PRINT#5,5
290 PRINT#1,"J";0;H
300 PRINT#5,0
310 REM====END========================
320 PRINT#1,"M";0;-100
330 CLOSE1
340 CLOSE5

READY.
```

The formula proper for this routine appears in line 220 of the listing. Input a couple of different values for side length when prompted (line 170); the routine will work out the triangle size, and draw the figure in centimeters (see line 200 -- 1 cm=50 steps of the drum). The description of the right triangle (see below) will not be quite as detailed as this, since the procedures are quite similar.

You could adapt this program to accept any three line lengths (one per side) by creating a subroutine which performs a task similar to a compass for measuring angles.

Always remember that thinking out the program first (before typing in a new program) will lead to a better program!

The concept of the isosceles triangle is not much trouble. The height formula is not enough here: We'll need to add two more input routines to handle:

1) Base length; and
2) Side length (sides are equal to each other, but not equal to the base)

Once this input is created, the center-point of the base can be calculated. Next, the height of the "center pole" can be computed, and the sides drawn accordingly. The program's structure follows this format:

Algorithms

1) OPEN and set up plotter
2) Base / Sides prompted
3) Height computed
4) Draw triangle
5) Draw height mark (dotted line)
6) End

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 OPEN5,6,5
130 PRINT#1,"M";200;-50
140 PRINT#1,"I"
150 REM====BASE/SIDES==================
160 PRINT"sMAX. 8 CM FOR BASE"
170 PRINT"QMAX.15 CM FOR SIDES"
180 INPUT"QQBASE ";A$
190 INPUT"QQSIDES";B$
200 A=VAL(A$):A=ABS(A)
210 B=VAL(B$):B=ABS(B)
220 IFA>8ORB>15ORA<1THEN160
230 REM====COMPUTE HEIGHT==============
240 A=A*50
250 B=B*50
260 H=SQR(B↑2-(A/2)↑2)
270 REM====DRAW TRIANGLE===============
280 PRINT#1,"J";-A/2;0
290 PRINT#1,"J";0;H
300 PRINT#1,"J";A/2;0
310 PRINT#1,"J";0;0
320 REM====DRAW HEIGHT=================
```

```
330 PRINT#5,5
340 PRINT#1,"J";0;H
350 REM====END========================
360 PRINT#5,0
370 PRINT#1,"M";0;-100
380 CLOSE1
390 CLOSE5

READY.
```

There are a couple of points in this program that are worth your analysis:

You've probably noticed that the INPUT statements (lines 180 & 190) have string variables, rather than number variables. The main reason for this is to avoid a REDO FROM START? error; in other words, if a letter or graphic symbol is inadvertently input, lines 200-220 will simply check for this input, and start the input routine all over again (line 160 on). If numbers are given within the range stated (see lines 160-170), the program continues on. A base value of 0 will also restart the input routine; however, a base of more than 0 and a sidelength of 0 WILL run the routine -- all you'll get is a horizontal line, though (the base alone).

The more observant readers have surely seen the change in the height formula (line 260):

$H=SQR(3/4*L^2)$ becomes $H=SQR(B^2-(A/2)^2)$.

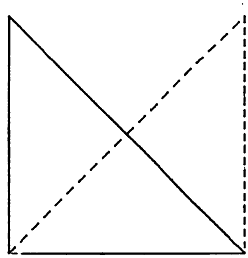
This is correct; the previous formula (Pythagorean Theorem) was specifically for an equilateral triangle. This new height formula will serve our isosceles figure best.

The rest of the program is material we've already covered.

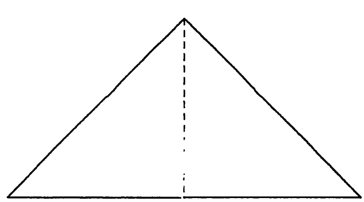
Our final item in this section is the right triangle, which must have a 90-degree angle somewhere within the figure. This triangle is very easy to construct, but in order to make any right triangle, we should run through a little groundwork on such figures, for your reference.

There are three possibilities in plotting a right triangle:

1) You can put the right angle at the left side of the base (point A).
2) You can set the right angle at the right end of the base.



3) With some effort, you can place the 90-degree angle at the top of the triangle ("point").



In this pair of illustrations, the base is represented by points A - B, with the 90-degree angle at point C. The remaining two angles add up to 90 degrees (each angle is 45 degrees). This means that this program will turn out right triangles that are also isosceles triangles (extra note; the height will be one-half the size of the base).

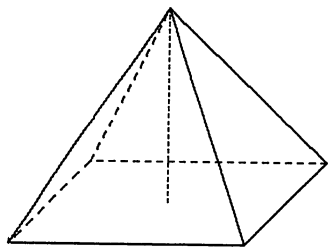
We've given you no routines for the first two triangles, since you have the raw materials from the last few pages to write one yourself. The following program will create a duplicate of Example 3.

```
10 REM ====TRIANGLE W/RIGHT ANGLE======
20 REM
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 OPEN5,6,5
130 PRINT#5,0
140 PRINT#1,"M";240;-50
150 PRINT#1,"I"
160 REM====INPUT=======================
170 PRINT"sBASE-MAX. 8 CM"
180 INPUT"QQBASE LENGTH";A$
190 A=VAL(A$):A=ABS(A)
200 IFA>8THEN170
210 B=A*25
220 REM====DRAW TRIANGLE===============
230 PRINT#1,"J";-B;0
240 PRINT#1,"J";0;B
250 PRINT#1,"J";B;0
260 PRINT#1,"J";0;0
270 REM====DRAW HEIGHT=================
280 PRINT#5,5
290 PRINT#1,"J";0;B
300 REM====END=========================
310 PRINT#5,0
320 PRINT#1,"M";0;-100
330 CLOSE1
340 CLOSE5
```

Lines 170-210 prompt for base length in centimeters, then check for valid input; line 210 initializes the plotter routine. Next, the triangle itself is drawn, followed by the height-line (dotted).

Before going on to circles and ellipses, we'll touch on one last spatial figure -- the pyramid. It consists of a square base and four triangles set at the perimeter of the base. The program below will draw a pyramid in perspective:

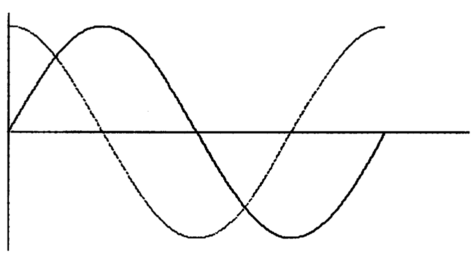
```
10 REM ====PRYAMID=====================
20 REM
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 OPEN5,6,5
130 PRINT#1,"M";100;0
140 PRINT#1,"I"
150 REM====READ DATA===================
160 FORX=0TO11
170 READX$,Y,Z
180 IFX=6THENPRINT#5,5
190 IFX=10THENPRINT#5,3
200 PRINT#1,X$,Y,Z
210 NEXT
220 REM====END=========================
230 PRINT#5,0
240 CLOSE1
250 CLOSE5
260 END
270 REM====DATA========================
280 DATAJ,200,0,J,270,70,J,135,200
290 DATAJ,200,0,R,135,200,J,0,0
300 DATAJ,70,70,J,270,70,R,70,70
310 DATAJ,135,200,J,135,35,M,0,-100

READY.
```



Lines 280-290 draw the foreground of the pyramid, while lines 300-310 sketch out the height and background. Lines 160-210 have been made as efficient as they can be; lines 180-190 change the line types used by the plotter.

8.3.5 Arcs

You have probably had enough of drawing straight lines -- now we combine both X- and Y-registers to draw curves. In order to make curves here, though, you'll have to disregard any formulas you've learned up until now; SINE and COSINE are incompatible. Look at our sample drawing below, which uses both registers; the sine curve is a connected line, while the cosine is a series of dots, through a procedure which we'll come to in a moment.



In order to produce both lines in the same routine, we need a new routine used by the 1520 plotter for circles and ellipses. We'll begin with a listing which will give you the basics of arc drawing, and a detailed analysis of this listing.

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 OPEN5,6,5
130 PRINT#1,"M";240;0
140 PRINT#1,"I"
150 REM====DRAW CIRCLE=================
160 FORX=-1.6TO4.8STEP.04
170 Y=150*COS(X)
```

```
180 Z=150*SIN(X)
190 IFX=-1.6THENPRINT#1,"R";Y;Z
200 PRINT#1,"J";Y;Z
210 NEXT
220 REM====RADIUS======================
230 PRINT#5,5
240 PRINT#1,"J";0;0
250 REM====END=========================
260 PRINT#5,0
270 PRINT#1,"M";0;-200
280 CLOSE1
290 CLOSE5
```

This explanation begins with lines 160-210, since the OPEN routine should pose no problem.

160 The range of numbers needed in the sine and cosine are set entirely into the STEP loop. Our example gives you STEP 0.4, or 160 moves. The step length determines the resolution of the drawing. Larger values aren't recommended, since the figure will be drawn much too large for the paper -- and the printout time increases dramatically. Thus, -1.6 is our starting point for a quarter-circle to be drawn. If you changed the line to FOR X=0 TO 6.4, you'd get roughly the same result with the starting point placed elsewhere on the page.

170 Variable Y holds the respective X-value for COS. The result is multiplied by a constant (150 in our example). This constant fixes the radius for the X-direction. Part of this arc will use this function.

180 Variable Z arranges the SINE-portion of the arc. The SINE-value must then be multiplied by a constant factor, to determine the radius of the Y-direction. The combined radii give us an arc.

190 Line 130 gave us the middle point of the arc (240/0). Before the plotter begins the draw routine, it will draw a dashed line showing the midpoint and radius; it starts this procedure immediately after the first value (-1.6) is given. If you start the loop (line 160) with 0, line 190 must also be changed to 0!

200 This line draws our curve.

210 The entire procedure begins anew.

This analysis will be of particular interest to the neophyte 1520 owner, since a similar (undocumented) routine is printed in the 1520 handbook. Experiment with this listing, change numbers around, play with this program until you have an idea of what works and what doesn't.

The next listing is for those of you who feel a bit more comfortable with the arc function:

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 OPEN5,6,5
130 PRINT#1,"M";240;0
140 PRINT#1,"I"
150 REM====INPUT=======================
160 PRINT"sRADIUS MAX 4 CM"
170 PRINT"QRESOLUTION .01 - .1"
180 INPUT"QRADIUS";A$
190 INPUT"QRESOLUTION";B$
200 A=VAL(A$):A=ABS(A)
210 B=VAL(B$):B=ABS(B)
220 IFA>4THEN160
230 IFB>.1ORB<.01THEN160
240 A=A*50
250 REM====
 CURVE==================
```

```
260 FORX=-1.6TO4.8STEPB
270 Y=A*COS(X)
280 Z=A*SIN(X)
290 IFX=-1.6THENPRINT#1,"R";Y;Z
300 PRINT#1,"J";Y;Z
310 NEXT
320 REM====RADIUS======================
330 PRINT#5,5
340 PRINT#1,"J";0;0
350 REM====END=========================
360 PRINT#5,0
370 PRINT#1,"M";0;-200
380 CLOSE1
390 CLOSE5
```

Once the program has been started, you can input the radius and line resolution during the program run (affecting plotting time proportionately). The drawing will conclude with the radius drawn in dashes.

The analysis of the arc problem showed that two different radii are available (for X- and Y-registers). This knowledge will help us to construct an ellipse. The final listing in this section will draw an ellipse; if you want a circle instead, just make the two radius inputs equal.

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 OPEN5,6,5
130 PRINT#1,"M";240;0
140 PRINT#1,"I"
150 REM====INPUT=======================
160 PRINT"sRADIUS MAX 4 CM ←"
170 PRINT"QRADIUS MAX 8 CM ↑"
180 PRINT"QRESOLUTION .01 - .1"
190 INPUT"QRADIUS ←";A$
200 INPUT"QRADIUS ↑";A1$
210 INPUT"QRESOSUTION";B$
220 A=VAL(A$):A=ABS(A)
230 REM====INPUT======================
240 A1=VAL(A1$):A1=ABS(A1)
250 B=VAL(B$):B=ABS(B)
260 IFA>4THEN160
270 IFA1>8THEN160
280 IFB>.1ORB<.01THEN160
290 A=A*50
300 A1=A1*50
```

```
310 REM====DRAW ELLIPSE===============
320 FORX=-1.6TO4.8STEPB
330 Y=A*COS(X)
340 Z=A1*SIN(X)
350 IFX=-1.6THENPRINT#1,"R";Y;Z
360 PRINT#1,"J";Y;Z
370 NEXT
380 REM====RADII======================
390 PRINT#5,5
400 PRINT#1,"R";0;0
410 PRINT#1,"J";A;0
420 PRINT#1,"R";0;0
430 PRINT#1,"J";0;A1
440 REM====END========================
450 PRINT#5,0
460 PRINT#1,"M";0;-200
470 CLOSE1
480 CLOSE5

READY.
```

If any problems arise with this program, it is suggested that you reread the theory of constructing a circle.
Now, we'll put past theories into practice and draw two spatial objects combining curves and straight lines: the cylinder and the cone. Both objects will be drawn in perpective.

A cylinder is made up of two parallel ellipses connected by two parallel lines. The two ellipses are plotted one-half at a time (-3.2 to 0, then 0 to 3.2).

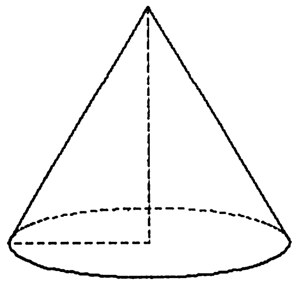
```
100 REM====1520 SETUP=================
110 OPEN1,6,1
120 OPEN5,6,5
130 PRINT#1,"M";240;0
140 PRINT#1,"I"
150 REM====
 ELLIPSE================
160 FORX=-3.2TO3.2STEP.04
170 Y=120*COS(X)
180 Z=30*SIN(X)
190 IFX=-3.2THENPRINT#1,"R",Y,Z
200 IFBN=1THENV=V+1
210 IFV=80THENPRINT#5,5
220 PRINT#1,"J";Y;Z
230 NEXT
240 IFBN=1THEN320
250 REM====SETUP 2ND ELLIPSE===========
260 PRINT#1,"J";-120;-300
270 PRINT#1,"R";0;-300
280 PRINT#1,"I"
290 BN=1
300 GOTO160
310 REM====DRAW HEIGHT=================
320 PRINT#1,"R";120;0
330 PRINT#5,0
340 PRINT#1,"J";120;300
350 PRINT#1,"R";0;300
360 PRINT#5,5
370 PRINT#1,"J";0;0
380 REM====END=========================
390 PRINT#5,0
400 PRINT#1,"R";-240;-100
410 CLOSE1
420 CLOSE5

READY.
```

The program begins with the upper ellipse (from line 160). The factor for the radius lies in 120 for the X-register (line 170), and in 30 for the Y-register (line 180). Remember that any alterations will have to encompass both these values.

Lines 260 to 300 draw the second ellipse; a shift occurs in the relative null-point, and we return to the ellipse routine in line 150, with the upper half of ellipse 2 drawn as a dotted line (line 210). The remainder of the cylinder is draw in lnes 320-370.

The procedure for plotting a cone is somewhat easier, since only one ellipse is needed to give us our perspective drawing.

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 OPEN5,6,5
130 PRINT#1,"M";240;0
140 PRINT#1,"I"
150 REM====DRAW ELLIPSE================
160 FORX=-3.2TO3.2STEP.04
170 Y=120*COS(X)
180 Z=30*SIN(X)
190 IFX=-3.2THENPRINT#1,"R",Y,Z
200 V=V+1
210 IFV=80THENPRINT#5,5
220 PRINT#1,"J";Y;Z
230 NEXT
240 REM====DRAW CONE===================
250 PRINT#1,"J";0;0
260 PRINT#1,"J";0;200
270 PRINT#5,0
280 PRINT#1,"J";-120;0
290 PRINT#1,"R";120;0
300 PRINT#1,"J";0;200
310 REM====END=========================
320 PRINT#1,"M";0;-100
330 CLOSE1
340 CLOSE5

READY.
```



The ellipse subroutine (lines 160-220) needs no explanation here, since it's nearly identical to the cylinder routine. Not only that, the height and sides of the cone itself are quite similar to the triangle routines seen earlier.

8.3.6 Triangle 2

To conclude this section on geometric figures, we should give you some materials for plotting the right triangle with different side lengths. We really couldn't give you this before now, because some knowledge of circle construction is needed to produce this last figure.

The program below will show how we add the arc routine to make such a triangle. Its logic contains the following:

Algorithm

1) Starting point defined
2) Half-circle drawn, and starting point of triangle drawn
3) half-circle ended
4) base drawn

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 OPEN5,6,5
130 PRINT#1,"M";240;0
140 PRINT#1,"I":E=50
150 REM====DRAW SEMICIRCLE=============
160 FORX=0TO3.2STEP.04
170 Y=120*COS(X)
180 Z=120*SIN(X)
190 IFX=0THENPRINT#1,"R",Y,Z
200 V=V+1
210 IFV=ETHENPRINT#1,"J";120;0
220 IFV=ETHENPRINT#1,"R";Y-.04;Z-.04
230 IFV=ETHENPRINT#1,"J";-120;0
240 PRINT#1,"J";Y;Z
250 NEXT
260 REM====BASE========================
270 PRINT#1,"R";120;0
280 PRINT#1,"J";-120;0
290 REM====END=========================
300 PRINT#1,"M";0;-100
310 CLOSE1
320 CLOSE5

READY.
```

Line 140 defines the starting point of the half-circle (line 140; E=50); the triangle start is also taken from here. The arc is scribed in 80 small sections (3.2/0.04=80). Line 200 increments variable V by one, so that E will equal 50 when time comes to draw the triangle. Once the jump point is reached (lines 210-230), the corner of base AB is set-up and a right angle placed in point C.

The base is drawn, and the routine ends. Try any value between 1 and 89, and watch what types of triangles you get.

The abovementioned geometric figures are just examples of what can be done in terms of program design and problem-solving. If there's a figure you wish to see plotted, we've given you the fundamentals -- it's up to you to look up the mathematical factors involved, and design your own plotter routine. Here are a few suggestions:

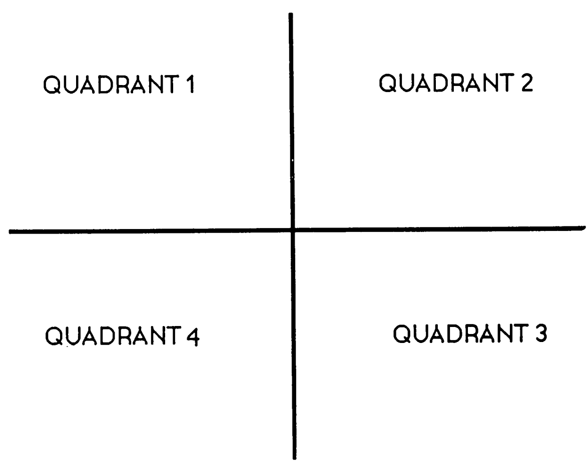
a) prisms
b) pyramids / truncated cones
c) ball or sphere in perspective
d) ellipsoid

All you need to design one of these figures is contained in the last few pages: It may be hard work in the design stages, but the plotter CAN draw these -- and more.

8.4 Representations of Functions

The term "function graph" is normally associated with the SINE, COSINE and X-functions. We've already described the latter to some extent (see the discussion on curves in the previous section). One range of interest here is the range of X to X, often found in math problems.

On to the operation of functions. Graphs are normally drawn in a system of coordinates, in this case, the X- and Y-axes. These axes are often divided into four quadrants (partial ranges), which are numbered clockwise here, beginning with the negative X-axis.



All references in this section will use this illustration as a guide. Now, there are different types of coordinate systems to choose from in plotting graphs; the dividing cross here is not always necessary. For example, if you were drawing a parabola, the X-register would not have much importance -- at least, not so much importance as the positive Y-axis in this figure.

A sine or cosine almost always moves in the positive X-axis, and both sides of the Y-axis.

X=X

This function would give you a horizontal line, with a second line intersecting the nullpoint, resulting in a 45-degree angle. Y=X would put the horizontal line in Quadrant 2, and the intersecting line in Quadrant 4. This program can be altered and the angles turned to your own tastes:

```
100 REM====1520 SETUP==================
110 OPEN4,6
120 OPEN1,6,1
130 OPEN2,6,2
140 REM====COORDINATE SYSTEM===========
150 FORA=1TO6
160 READX$,Y,Z
170 PRINT#1,X$;Y;Z
180 NEXT
190 DATAM,0,0,D,480,0,M,240,240
200 DATAD,240,-240,M,240,0,I,0,0
210 REM====DEFINITION OF FUNCTION======
220 DEFFNA(X)=X
230 REM====DRAW========================
240 FORA=200TO-200STEP-1
250 IFA=200THENPRINT#1,"R";A;FNA(A)
260 PRINT#1,"J";A;FNA(A)
270 NEXT
280 PRINT#1,"M";0;0
290 REM====END=========================
300 CLOSE1
310 CLOSE2
320 CLOSE4

READY.
```

Here's an analysis of this routine, showing you the X^n-function.

150-180 Data for the coordinate division is read in and drawn.

190-200 Data for the coordinate system

220 The C-64 has the ability to define functions (DEF) within programs. You can define functions in later developments of this program by adding PRINT FNA (number), etc., with (number) referring to the previously-defined function. If you don't define the function, you must keep specifying the function throughout the program.

240 This line specifies a limit to our range of movement for the function. The 1520's maximum for the X-axis is 480, so we gave the program a -200 -by- +200 range for both X- and Y-registers, and still have some "room" remaining.

260 The value calculated by the Y-value is drawn here, in conjuntcion with the X-value (Y=X).

Now, let's experiment with this program, and make some changes. Try this: Change line 240 from "200TO-200" to "100TO-100"; change line 250 to "FORA=100..." to match; multiply A (lines 250/260) by 2. Finally, change line 260 to read 260 PRINT#1, "J";2-A; FNA(A). Change line 250 to match.

When you start this version of the program, you'll note that the angle is decreased considerably. You can exercise a lot more control by adding the equation Y=mx+b. m represents the angle, b is the nullpoint marker (0/0) across the Y-axis.

One other change that can be made is the degree of resolution, which is done by altering certain aspects of the program. The most elegant method is to multiply FNA (A) by a constant, diminishing the raised exponent of the function, until the starting point of the graph is almost identical to the old version.

There is another item that we should consider: The comparison of functions from $x - x^4$. You can see immediately the principle behind these. If we have the function Y=X, and have X= 100, the result will be 100. We used this when considering the starting point. If, in the above function, we reached the maximum of 100, the plotter would only draw up to 100, and remain there however many steps above 100 the counter goes. An answer to this problem lies in the ability to change nullpoints in the X-register (0-480). However, we would raise the function in exponents of one, giving us the function $Y=X^2$. If we reach 100, the result for Y will already be $100^2 = 10{,}000$; so, the result of our increase is therefore 100 times greater.

This is confirmed in the functions $Y=X^3$ and $Y=X^4$.

The proportionality factor is easier to find for non-integer exponents, using a common formula. There is one factor that should remain constant; the exponentiation of X and Y should be identical (exponentiation is used for value ranges larger than 100), using FNA(A) as a reference. When working with parabolas, the notation X^2 should be accompanied by Y multiplied by .01. In this case, with a value greater than 100, 100 * 100 * 0.01 gives the proper result. X^3 gives a proportionality factor of .0001.

Now, you'll need the formula for computing the constant C: $X^n=Y$, with n equalling any number.

You are not limited to natural numbers in your applications -- rational numbers, etc., are possible. When trying to decide upon an exponent for FNA (A), use this expression:

$C=1/100^{(n-1)}$

Example: You want to graphically depict the function X^6. Setting 6 into ^{n}, you get

$C=1/100^{(6-1)}$

and the result is 10^{-10}.

This value of 6 is put into the function $Y=X^6$, with X equalling 100.

The next three routines turn these theories into practice, using X^2, X^3, and X^4.

The listings are fairly short to help give you a solid understanding of the equations, followed by the documentation.

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 REM====COORDINATE SYSTEM===========
130 FORA=1TO6
140 READX$,Y,Z
150 PRINT#1,X$;Y;Z
160 NEXT
170 DATAM,0,0,D,480,0,M,240,240
180 DATAD,240,-20,M,240,0,I,0,0
190 REM====DEFINITION OF FUNCTION======
200 DEFFNA(X)=X^2
210 C=.02
220 REM====DRAW========================
230 FORA=100TO-100STEP-1
240 IFA=100THENPRINT#1,"R";A;C*FNA(A)
250 PRINT#1,"J";A;C*FNA(A)
```

```
260 NEXT
270 PRINT#1,"M";0;0
280 REM====END==========================
290 CLOSE1
```

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 REM====COORDINATE SYSTEM===========
130 FORA=1TO6
140 READX$,Y,Z
150 PRINT#1,X$;Y;Z
160 NEXT
170 DATAM,0,0,D,480,0,M,240,240
180 DATAD,240,-240,M,240,0,I,0,0
190 REM====DEFINITION OF FUNCTION======
200 DEFFNA(X)=X^3
210 C=.0002
220 REM====DRAW=========================
230 FORA=100TO-100STEP-1
240 IFA=100THENPRINT#1,"R";A;C*FNA(A)
250 PRINT#1,"J";A;C*FNA(A)
260 NEXT
270 PRINT#1,"M";0;0
280 REM====END==========================
290 CLOSE1
```

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 REM====COORDINATE SYSTEM===========
130 FORA=1TO6
140 READX$,Y,Z
150 PRINT#1,X$;Y;Z
160 NEXT
170 DATAM,0,0,D,480,0,M,240,240
180 DATAD,240,-20,M,240,0,I,0,0
190 REM====DEFINITION OF FUNCTION======
```

```
200 DEFFNA(X)=X^4
210 C=.000002
220 REM====DRAW========================
230 FORA=100TO-100STEP-1
240 IFA=100THENPRINT#1,"R";A;C*FNA(A)
250 PRINT#1,"J";A;C*FNA(A)
260 NEXT
270 PRINT#1,"M";0;0
280 REM====END=========================
290 CLOSE1
```

As you can plainly see, these three programs are almost identical, with the exception of three lines. Line 180: This line contains the DATA for the coordinate system. The X^2 function we only need for the upper section of the coordinate cross.

The second program brings X^3 into play, giving us uneven quadrants above and below the X-axis, while X^4 affects quadrants 1 and 2. The X^3 function in line 180 changes line 180 to

```
180 DATAD,240,-240....
```

(NOTE: The negative number in the DATA line gives the length of the negative-Y-axis).

Line 200: This line defines the respective function to be performed, and line 210 fixes the proportionality factor to be used in the formula, multiplied by the value range.

Function	Factor
x^2	0.02
x^3	0.0002
x^4	0.000002

In order to apply the formula for C in the program, however, here is a short routine which utilizes the natural numbers from 0 - 11, which you insert as exponents in the formula and the function. Once the coordinate cross is drawn, the exponent will be prompted, C calculated, and the graph printed out.

```
100 REM====1520 SETUP=================
110 OPEN1,6,1
120 REM====COORDINATE SYSTEM===========
130 FORA=1TO6
140 READX$,Y,Z
150 PRINT#1,X$;Y;Z
160 NEXT
170 DATAM,0,0,D,480,0,M,240,240
180 DATAD,240,-240,M,240,0,I,0,0
190 REM====DEFINITION OF FUNCTION======
200 INPUT"sTO WHAT POWER (X↑?)";N$
210 N=VAL(N$):N=ABS(N)
220 IFN>11THEN200
230 FORP=NTO0STEP-1
240 IFP=0THEN270
250 NEXT
260 GOTO200
270 DEFFNA(X)=X↑N
280 C=2/100↑(N-1)
290 REM====DRAW=======================
300 FORA=100TO-100STEP-1
310 IFA=100THENPRINT#1,"R";A;C*FNA(A)
320 PRINT#1,"J";A;C*FNA(A)
330 NEXT
340 PRINT#1,"M";0;0
350 REM====END========================
360 CLOSE1

READY.
```

The input routine can be found in lines 200-250. Line 220 tests input for overflow, to avoid OVERFLOW ERROR messages. Lines 230-250 limit the numerical range at No. as well, with zero included in the allowable input range.

If you've typed in correct input, x^n defines the function in line 270. The formula to figure this constant is used in line 280, with line 300 insuring that the range stays within -100 to +100. If the two items in line 280 is a one, the function will be fairly low and wide. The draw routines can be altered as much as you wish.

Before we go on to sine and cosine curves, the folks who are having problems following what has preceded are best advised to reread the section on arcs. Most of what follows has already been covered there. The program to draw a complete sine-cosine curve set would be impossible to include in this book, since it takes up 61 blocks of memory (roughly 15K). However, we can take the more important factors in the program (zeropoint, minimum, maximum, and turning point) and give you a skeleton version for your experimentation.

This program uses functions with whole exponents drawn from $Y=X^5$.

A quick review of derivation:

The variable with the specific exponent (e.g., X^4) will deviate from the variable multiplied by the exponent, and the old exponent decremented by 1 ($4*X^3$).

In other words, X^n is derived from $n*X^{(n-1)}$.

The entire procedure is reciprocal if you have the function description, but not the trunk function. For example, let's say the first equation is $3*X^2$. You can ascertain the second equation, but you're still missing the basic function.

$3*X^2$ is what we have on hand. Now, we can raise the exponent by 1 -- or

$3*X^{(2+1)}$

and multiply that by the reciprocal of the exponent --

$(3*X^{(2+1)})/(2+1)$

which gives us X^3, which is the main function in concert with $3*X^2$.

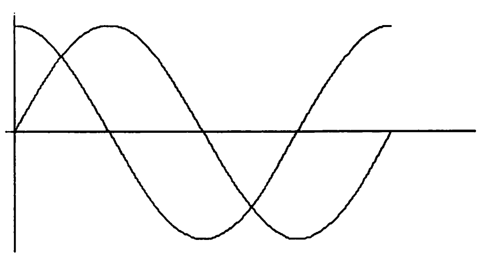
```
100 REM====INPUT=======================
110 INPUT"sENTER FUNCTION";X$
120 REM====FUNCTION SETUP==============
130 FORX=1TOLEN(X$)
140 Y$=MID$(X$,X,1)
150 Y=VAL(Y$)
160 IFY$=""THENNEXT
170 IFY$="X"THENE=E+1
180 IFY$="↑"THENNEXT
190 IFE=1ANDY>0THENA=Y
200 IFE<1ANDY>0THENB=Y
210 IFY$="-"THEN110
220 NEXT
230 REM====DERIVE======================
240 PRINT"Q1ST DRIVATIVE";A*B;"*X↑";A-1
250 C=A*B*(A-1)
260 PRINT"Q2ND DERIVATIVE";C;"*X↑";A-2

READY.
```

This program serves to give you a good example of what we've been talking about. You can only try small functions, though, such as $9*X^7$, etc., but the first and second deviations are printed out.

Now for the material promised above; the sine and cosine functions. These functions are in C-64 BASIC, so a short routine is all we need for the fundamental materials.

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 OPEN2,6,2
130 PRINT#2,0
140 REM====COORDINATE SYSTEM===========
150 PRINT#1,"D";400;0
160 PRINT#1,"M";10;100
170 PRINT#1,"D";10;-100
180 PRINT#1,"M";10;0
190 PRINT#1,"I"
200 REM====FUNCTION====================
210 DEFFNA(X)=SIN(X)
220 REM====CALCULATE/PLOT==============
230 FORX=0TO320
240 PRINT#1,"J";X;FNA(X/51)*90
250 NEXT
260 REM====END=========================
270 PRINT#1,"R";0;0
280 CLOSE1
290 CLOSE2
```

Line 210 defines the sine function; you can insert the cosine without too much trouble. An added size factor (line 240) lets you stretch the height. Once you've added the cosine routine and run the program, the result should be:



The sine is drawn from the point of origin, with the cosine phasing at 90 degrees. The complete routine looks like this:

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 OPEN2,6,2
130 PRINT#2,0
140 REM====COORDINATE SYSTEM===========
150 PRINT#1,"D";400;0
160 PRINT#1,"M";10;100
170 PRINT#1,"D";10;-100
180 PRINT#1,"M";10;0
190 PRINT#1,"I"
200 REM====FUNCTION====================
210 DEFFNA(X)=SIN(X)
220 DEFFNB(X)=COS(X)
230 REM====CALCULATE/PLOT==============
240 FORX=0TO320 STEP 5
250 PRINT#1,"J";X;FNA(X/51)*90
260 PRINT#1,"J";X;FNB(X/51)*90
270 NEXT
280 REM====END=========================
290 PRINT#1,"R";0;0
300 CLOSE1
310 CLOSE2

READY.
```

Lines 210 and 220 define the sine and cosine functions. If you only want crosshatching, add a STEP factor to line 240 (e.g., FORX=0TO320STEP4).

Then the computed points will be placed farther apart. There are other uses for functions, but electronic functions alone would fill an entire book in themselves.

8.5 Peculiarities of the 1520

We'll be going on to statistical functions shortly. For now, however, we are going to take a look at a few characteristics built into the 1520 plotter. As we know from earlier chapters, PRINT USING is not a feature of BASIC 2.0 or most Commodore printers. The 1520 is no exception, but PRINT USING can be simulated, i.e., proper formatting of text and/or numbers in a column.

This sample program prints two numbers in a row, lining up their respective decimal points, and beginning a new line.

```
100 REM====1520 SETUP==================
110 OPEN4,6
120 PRINT#4
130 REM====LENGTH OF NUMBER ALLOWED====
140 GZ=4
150 DZ=3
160 A=0
170 REM====INPUT/SETUP=================
180 INPUT"NUMBER";NR
190 A=A+1
200 NR$=STR$(NR)
210 NN=LEN(NR$):W=0
220 FORX=1TONN
230 IFMID$(NR$,X,1)="."THEN250
240 W=W+1:NEXT
250 REM====PRINT THE NUMBER============
260 PRINT#4,TAB(GZ+1-W);NR;:
270 PRINT#4,TAB((DZ+GZ)-(GZ-W+NN)+1);:
280 IFA<2THEN180
290 PRINT#4
300 A=0
310 GOTO180

READY.
```

140 Variable GZ sets the number of whole digits.

150 Sets the number of decimal places allowed (DZ). Please remember to limit yourself to these two variables when giving input.

200-240 Activates the print routine and lines the numbers up accordingly.

260 Prints the number; line 270 will precede the next number with a number of spaces (TAB).

When the novelty of this program wears off, you can halt it using RUN/STOP-RESTORE -- otherwise, the program will simply continue by asking for another number. Then, press the paper-feed key to return the drum to its starting-point. The PRINT USING program can be used for any number of things; for example, you can list items in text strings, followed by prices (limit text length to 25 characters).

The input form for this would read

PRINT#4,"TEXT";(defined text length = practical text length)

The PRINT USING function is no big deal on the 1520 because of the narrow paper used (see Chapter 5.1.2 for a program for formatting on a "big" printer). The same goes for the program below, which allows you to use your plotter as a typewriter of sorts:

```
100 REM====1520 SETUP==================
110 OPEN4,6
120 REM====READ KEYS===================
130 PRINT"s"
140 POKE198,0
150 WAIT198,1
160 GETX$:PRINTX$;
170 IFX$="e"THEN220
```

```
180 REM====PRINT CHAR==================
190 PRINT#4,X$;:
200 GOTO140
210 REM====END=========================
220 CLOSE4

READY.
```

The input goes to the keyboard buffer, and is placed in $00C6 (198 decimal).

140 Clears any buffer contents, writing a 0 to the memory cell.

150 Keeps the program at a standstill until a character is fed into location 198. The WAIT 198,3 means that the system will wait until three characters reach the buffer before going on. The corresponding character is read by GET and printed out in line 190.

170 The graphic symbol you see here is the abbreviation for f1 (function key 1), which will end the program if pressed.

Direct mode is a necessary evil to statistics on the 1520 (e.g., for drawing a bar graph). For the moment, though, let's have a little fun with a program called "JOYDRAWER".

We can read the joystick by means of the addresses 56320 or 56321 (dependent on which port you wish to use), and determine the direction the stick is pointing. A subroutine of this type looks like:

```
100 REM====READ JOYSTICK===============
110 X=PEEK(56320)
120 IF(XAND1)=0THENGOTO *LINE NUMBER*
130 IF(XAND2)=0THENGOTO *LINE NUMBER*
140 IF(XAND4)=0THENGOTO *LINE NUMBER*
150 IF(XAND8)=0THENGOTO *LINE NUMBER*
160 IF(XAND16)=0THENGOTO*LINE NUMBER*
170 GOTO110
```

Once the directions have been laid out, the *LINE NUMBER* designations can be used to order up certain commands.

This isn't the main reason we're playing with this routine, though -- joystick tests themselves aren't much fun. As you know, most joysticks are made up of two membrane keys of a sort -- one for the firebutton and one for the stick direction. Pushing the stick in a diagonal direction clicks in two switches simultaneously (one horizontal, one vertical). The above routine will not read two directions at the same time, so we'll have to do some improving to the program, or we'll only be able to draw up, down, left and right -- no diagonals.

We can get around this by creating a routine that reads two keys simultaneously (or, it will seem that way to you -- it will alternately read the two positions many times per second). Here is one solution:

```
100 REM====1520 SETUP==================
110 SW=3:C=2
120 OPEN4,6
130 OPEN1,6,1
140 OPEN2,6,2:PRINT#2,C
150 REM====READ JOYSTICK===============
160 J=PEEK(56320)
170 IF(JAND1)=0THENY=Y+SW
180 IF(JAND2)=0THENY=Y-SW
190 IF(JAND4)=0THENX=X-SW
200 IF(JAND8)=0THENX=X+SW
```

```
210 IF(JAND16)=0ANDKK=0THENKK=1:GOTO240
220 IF(JAND16)=0ANDKK=1THENKK=0
230 REM====TEST FOR OVERFLOW===========
240 IFX<0THENX=0:GOTO160
250 IFY<-480THENY=-480:GOTO160
260 IFX>480THENX=480:GOTO160
270 IFY>480THENY=480:GOTO160
280 REM====DRAW=======================
290 IFKK=1THENK$="D"
300 IFKK=0THENK$="M"
310 PRINT#1,K$;X;Y
320 GOTO160

READY.
```

110 Two variables (SW=step width, C=color) can choose resolution (and consequently, drawing speed) and the pen color. Step width 1 corresponds to the highest resolution.

160-200 The four main directions of the joystick are read here, and variables Y and X determine the directions in between.

210 The firebutton is read here, and has a value of 0 or 1. This is important, since the firebutton sets the pen to paper. If the variable (KK) equals 1, line 220 takes effect, followed by a direct jump to the draw routine.

220 If the firebutton is pressed a second time, the program returns to line 210, chnaging KK to 0.

240-270 To prevent the drum from passing the border, variables X and Y test for overflow.

290-300 The draw and move commands (D/M) are stored in K$.

310 The draw routine is executed, and the routine runs again.

RUNning the program will immediately show you the advantages and disadvantages of this routine. The draw-speed is slow, and the diagonals look vaguely like little stairs.

A few words on operating this program:

When the firebutton is pressed, the pen drum does nothing. Once the lever is moved, the pen is switched onto the paper. This has the advantage that you can press the firebutton at the same time as the pen is moving and make dashed lines if you wish. The joystick, by the way, should be plugged into Port 2.

Those readers without joysticks can still use the program. Change the address in line 160 from 56320 to 56321, which will now allow you to read the keyboard. Here are the keys involved:

KEY	FUNCTION
2	right
CTRL	left
1	up
<	down
SPACE	firebutton

Once you have a better idea of this program, try the next one, which gives us even more capabilities, and adds the function keys.

```
100 REM====READ DATA===================
110 PRINT"s"
120 FORF=0TO3
130 READB$(F)
140 NEXT
150 REM====1520 SETUP==================
160 SW=3:C=0
170 OPEN4,6
180 OPEN1,6,1
190 OPEN2,6,2
200 PRINT#2,C
210 REM====READ JOYSTICK===============
220 GETA$:J=PEEK(56320)
230 IF(JAND1)=0THENY=Y+SW
240 IF(JAND2)=0THENY=Y-SW
250 IF(JAND4)=0THENX=X-SW
260 IF(JAND8)=0THENX=X+SW
270 IF(JAND16)=0ANDKK=0THENKK=1:GOTO350
280 IF(JAND16)=0ANDKK=1THENKK=0
290 IFA$=CHR$(133)THENCLOSE1:CLOSE2:END
300 IFA$=CHR$(134)THENSW=SW+1
310 IFA$=CHR$(138)ANDSW>1THENSW=SW-1
320 IFA$=CHR$(135)THENC=C+1
330 IFC=4THENC=0
340 REM====TEST FOR OVERFLOW===========
350 IFX<0THENX=0:GOTO200
360 IFY<-480THENY=-480:GOTO200
370 IFX>480THENX=480:GOTO200
380 IFY>480THENY=480:GOTO200
390 REM====DRAW========================
400 IFKK=1THENK$="D"
410 IFKK=0THENK$="M"
420 PRINT#1,K$;X;Y
430 PRINT"STEP";SW,B$(C)"
440 PRINT"qq"
450 IFSW<10THENPOKE1070,32
460 GOTO200
470 DATABLACK,BLUE,GREEN,RED
```

KEY	**FUNCTION**
F1	**end program**
F2	**raise step-width by 1**
F4	**decrement step-width by 1**
F5	**turns the pen drum one-quarter turn**

This version of the program gives us "real-time" control over the color and the resolution.

8.6 Statistics on the 1520 Plotter

Often the time comes when a non-statistician has to visually depict a few numbers in graph form. This may involve monetary sums, or percentages. We'll try to show you how to do such graphs on your 1520 Plotter, using two typical examples -- the bar chart, and the pie chart (named for obvious reasons). Both programs use percentages. We'll begin with the bar chart:

```
100 REM====1520 SETUP==================
110 OPEN4,6
120 OPEN1,6,1
130 OPEN2,6,2
140 REM====INPUT=======================
150 INPUT"sHOW MANY ITEMS (MAX.7)";A$
160 A=VAL(A$)
170 IFLEN(A$)>1ORA<1ORA>7THEN150
180 FORS=1TOA
190 PRINT
200 PRINTS;". ITEM (IN %)";:INPUTB(S)
210 IFB(S)>100THEN150
220 NEXT
230 REM====COORDINATES=================
240 PRINT#1,"D";450;0
250 PRINT#4,"X"
260 PRINT#1,"M";20;-20
270 PRINT#1,"D";20;520
280 PRINT#4,"Y"
290 PRINT#1,"M";20;-480
300 PRINT#1,"I"
310 REM====SCALE=======================
320 FORE=0TO500STEP50
330 PRINT#1,"R";0;E
340 PRINT#1,"J";-10;E
350 NEXT
360 PRINT#1,"H"
370 REM====COLUMNS=====================
380 FORS=1TOA
390 PRINT#1,"R";50;0:PRINT#1,"I"
400 Y(S)=B(S)*5
410 PRINT#1,"J";0;Y(S)
420 PRINT#1,"J";40;Y(S)
430 PRINT#1,"J";40;0
440 NEXT
```

```
450 PRINT#1,"R";-A*50;0
460 PRINT#1,"I"
470 PRINT#1,"R";50;-20
480 PRINT#1,"I"
490 REM====CAPTION=====================
500 POKE198,0:WAIT198,1
510 GETX$:IFX$<>"e"THENPRINT#4,X$;
520 IFX$<>"e"THEN500
530 REM====END=========================
540 CLOSE1
550 CLOSE2
560 CLOSE4

READY.
```

You can draw seven columns with this program, with 10 cm = 100 %. There will be marks on the Y-axis at 1-centimeter intervals (10%). Once the program has ended, it will shift into direct "typewriter" mode, allowing you to put text, numbers, etc. underneath the columns. Here is a description of the program logic:

150 **prompts for the number of columns that you want printed.**

170 **checks that the input is within the allowable range.**

180-220 **makes up the rest of the INPUT routine. Remember that this section accepts numeric input ONLY, and will give you an error message if you give it anything else. The values given are stored in a one-dimensional array, and are tested as to whether they are less than or equal to 100% (line 210). If all is well, the program goes on.**

240-300 **designs a coordinate system and draws the X- and Y-axes.**

320-350 marks the 1-centimeter steps on the Y-axis (see above), with 10 cm equalling 100% (line 320 contains STEP50, with 1 step equal to 0.2 mm; so, 50*0.02 mm =1 cm). This routine is called upon ten times to give us the ten markings.

380-440 takes the parameters given and prints the columns proper.

The next section of the program is probably the most important; before making any changes, be sure you understand the routine thoroughly.

380 The draw-loop counts up the number of columns input, from 1 to A.

390 A new relative null-point is defined as each column is plotted (running from left to right). Each column is drawn using the same routine, with only the size of the columns changed as specified.

400 Percentages are input into B (S), taken by the computer, and translated into centimeters for the plotter (500 steps = 10 cm = 100%).

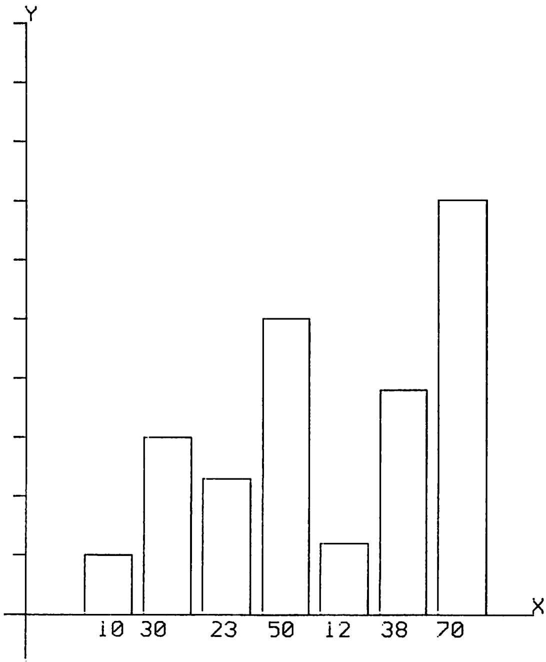
410-440 Columns are put onto paper, each a width of 40 steps and a height of Y (S), as specified by the user.

450-480 The plotter shifts into direct mode; the pen drum positions itself beneath the first column; and a new relative null-point is set.

500-520 This section waits for a key to be pressed -- the character pressed will be printed on paper (typewriter mode). When done typing, press F1 to end the program.

For review purposes, have a look at line 500: The keyboard buffer (location 198) is cleared using a POKE 198,0. WAIT

198,1 has the system wait until a key is pressed to act. When a key is pressed, the character goes into the variable in the GET statement (line 510), and is printed. This program gives decent resolution, and plots at a passable speed. You may wish to make some changes such as raise or lower maximum height (line 390 -- 50 is the present maximum), or change the width of the columns (lines 420-430). Here is a sample picture of the graph:



The pie-chart also works in percentiles; the percentages are divided off in pieces of a circle, similar to cutting slices of pie (hence the name). The entire circle represents 100%. This routine takes some time to run (ten inputs take about a minute), but the result is well worth the wait!

```
100 REM====1520 SETUP==================
110 OPEN1,6,1
120 OPEN4,6
130 PRINT#1,"M";240;0
140 PRINT#1,"I"
150 REM====INPUT=======================
160 DIMV(30)
170 INPUT"sQQHOW MANY SECTION <=30!";M$
180 M=VAL(M$):M=ABS(M)
190 FORN=1TOM
200 PRINT"SIDE";N;:
210 INPUTV(N)
220 NEXT
230 REM====TEST FOR OVERFLOW===========
240 FORN=1TOM
250 P=P+V(N)
260 IFP>100THENCLR:GOTO170
270 PRINT"SCHECKSUM";TAB(20)P;"   "
280 NEXT:N=0
290 REM====SUBDIVISION=================
300 N=N+1
310 K=128*V(N)/100
320 FORL=0TO6.4STEP.05:Z=Z+1
330 X=200*SIN(L)
340 Y=200*COS(L)
350 IFZ-K<1ANDZ-K>=0THEN400
360 IFL=0THENPRINT#1,"J";X;Y
370 PRINT#1,"R";X;Y
380 NEXT:GOTO470
390 REM====NEXT VALUE==================
400 Z=0
410 PRINT#1,"J";0;0
420 PRINT#1,"R";X;Y
430 N=N+1
440 K=128*V(N)/100
450 GOTO370
460 REM====DRAW CURVE==================
470 FORL=0TO6.4STEP.05
480 X=200*SIN(L)
490 Y=200*COS(L)
500 IFL=0THENPRINT#1,"R";X;Y
510 PRINT#1,"J";X;Y
520 NEXT
530 REM====CAPTION=====================
540 PRINT#1,"R";0;200
550 POKE198,0:WAIT198,1
560 GETA$:IFA$="e"THEN600
570 PRINT#4,A$;
580 GOTO550
590 REM====END=========================
600 CLOSE1
610 CLOSE4
620 POKE214,12
630 PRINT
```

160 The percentages given by the user are stored in a one-dimensional array. Normal arrays will take up to 11 inputs before the computer gives a BAD SUBSCRIPT error. Our array allows for up to thirty inputs (DIMV(30)).

170-220 The user is prompted here for input for the array mentioned above. The prompt tells you to limit your inputs to 30 or less, although it actually accepts 31 inputs (includng 0, which is a reserve, nothing more).

240-280 The contents of the array are summed up here, and checked for overflow (i.e., insuring that the sum <=100%).

310-350 The value K determines the size of the radius. The circle itself is drawn using FORL=0TO6.4STEP.05. -- which gives a total of 128 individual steps. Variable Z (line 320) counts each step, and increments it until 128 is reached.

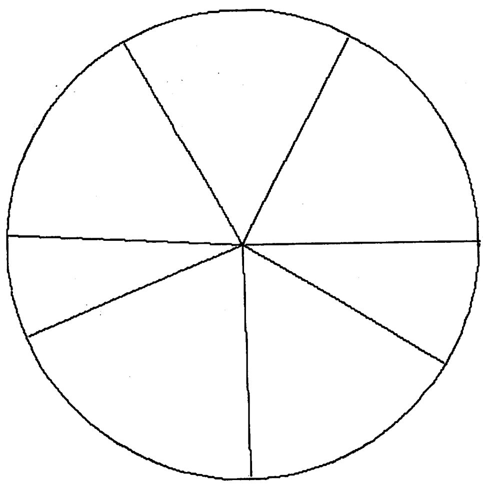
Lines 330-350 will look familiar to the reader (see Chapter 8.3.5).

400-450 The next input is read in, and the drawing routine restarts.

470-520 need no explanation: center-point is shown.

540-580 The system returns to typewriter mode -- press F1 to end the program.

There; you now have the basic materials for creating your own statistical charts with the 1520 plotter. Good luck!



8.7 PLOTTER UTILITY PROGRAM

The following BASIC utility program is a long one, and will take plenty of typing time; but it will be well worth your trouble. The program splits up the secondary addresses of the 1520, and even gives you routines such as the lines and circles drawn earlier. Essentially, it makes plotter software development a bit easier for the novice. (NOTE: As with so many of these programs, the execution speed is not very fast). The pages following the program give details on commands and program logic. If you get impatient for commands, here's a short guide:

CIRCLE,Mx,My,Rx,Ry,L draws a circle, giving you options on midpoint, radii (X- and Y-registers) and type of line.

LINE,X1,X2,Y1,Y2,L draws a line from one point to another (again, choice of line type).

PAPER UP/DOWN,N moves the paper N steps up or down.

TURN/TEXT turns any text 90 degrees either to the X- or Y-axis.

Changes in color, character size, etc. will be pointed out to you in the full command set later, and in sample runs.

```
10 OPEN1,6,1
20 OPEN2,6,2
30 OPEN3,6,3
40 OPEN4,6,4
50 OPEN5,6,5
60 OPEN6,6,6
70 OPEN7,6,7
80 OPEN8,6:PRINT"s":DIMEN$(100):DIMW1(10
0):RESTORE
90 GOSUB9000:POKE53280,0:POKE53281,2:PRI
NT"E"
100 REM====INPUT======================
110 E$="←":PO=56320
120 PRINT"< ";:
130 GETA$:IFA$=""THEN130
140 IFA$="Q"ORA$="q"ORA$="]"ORA$="△"ORA$
=CHR$(34)ORA$="s"ORA$="S"THEN130
150 IFA$=CHR$(13)THENPRINT:PRINT"q<":Z=0
:S=S+1:GOTO240
160 IFA$=CHR$(20)THENZ=Z-1:IFZ<0THENZ=0:
GOTO130
170 IFA$=CHR$(20)THEN190
180 Z=Z+1:IFZ>70THENA$=CHR$(13):GOTO150
190 PRINTTAB(3)"△"A$+E$;:
200 IFA$="*"THEN330
210 T$=T$+A$
220 GOTO130
230 IFA$=CHR$(20)THENTT$=LEFT$(TT$,LEN(T
T$)-2)
240 Y=LEN(T$):E=E+1
250 FORX=1TOY
260 S$=MID$(T$,X,1)
270 IFS$=" "ORS$="↑"ORS$="←"THENNEXT
280 EN$(E)=EN$(E)+S$
290 IFS$<>" "ANDS$<>"↑"ANDS$<>"←"THENNEX
T
300 T$="":A$="":S$=""
310 GOTO120
320 REM====WORK OUT===================
330 E=1
350 REM====READ COMMENT===============
360 FORX=1TOLEN(EN$(E))
370 BF$=MID$(EN$(E),X,1)
380 IFBF$<>","THENB1$=B1$+BF$:NEXT
390 REM====READ NUMBERS===============
400 FORX=LEN(B1$)+2TOLEN(EN$(E))
410 Z1$=MID$(EN$(E),X,1)
420 IFZ1$<>","THENN1$=N1$+Z1$:NEXT
430 REM====READ NUMBERS===============
440 FORX=LEN(N1$)+2+LEN(B1$)+1TOLEN(EN$(
E))
```

```
450 Z2$=MID$(EN$(E),X,1)
460 IFZ2$<>","THENN2$=N2$+Z2$:NEXT
470 REM====READ NUMBERS================
480 FORX=LEN(N1$)+1+LEN(B1$)+1+LEN(N2$)+
2TOLEN(EN$(E))
490 Z3$=MID$(EN$(E),X,1)
500 IFZ3$<>","THENN3$=N3$+Z3$:NEXT
510 REM====READ NUMBERS================
520 FORX=LEN(N1$)+1+LEN(B1$)+1+LEN(N2$)+
1+LEN(N3$)+2TOLEN(EN$(E))
530 Z4$=MID$(EN$(E),X,1)
540 IFZ4$<>","THENN4$=N4$+Z4$:NEXT
550 REM====READ NUMBERS================
570 FORX=LEN(N1$)+1+LEN(B1$)+1+LEN(N2$)+
1+LEN(N3$)+1+LEN(N4$)+2TOLEN(EN$(E))
580 Z5$=MID$(EN$(E),X,1)
590 IFZ5$<>" "THENN5$=N5$+Z5$:NEXT
600 REM====STRINGS IN NUMBERS==========
610 N1=VAL(N1$):N1=ABS(N1)
620 N2=VAL(N2$):N2=ABS(N2)
630 N3=VAL(N3$):N3=ABS(N3)
640 N4=VAL(N4$):N4=ABS(N4)
650 N5=VAL(N5$):N5=ABS(N5)
660 REM====EXECUTE=====================
670 REM====COMMANDS====================
680 IFB1$<>"COLOR"THENGOTO710
690 IFN1>255THENPRINT#1,0:GOTO710
700 PRINT#2,N1
710 IFB1$<>"PEN"THEN740
720 IFN1>255THENN1=0
730 PRINT#2,N1:PRINT#1,"M";480,0
740 IFB1$<>"LINE"THEN780
750 IFN1>480ORN2>999ORN3>480ORN4>999ORN5
>15THEN780
760 PRINT#5,N5
770 PRINT#1,"M";N1;N2:PRINT#1,"D";N3;N4
780 IFB1$<>"SIGN"THENGOTO810
790 IFN1>255THENN1=1
800 PRINT#3,N1
810 IFB1$<>"TEST"THEN830
820 PRINT#8,"T E S T"
830 IFB1$<>"PAPERUP"THEN870
840 IFN1>999THENN1=998
850 PRINT#1,"M";0;N1
860 PRINT#8
870 IFB1$<>"PAPERDOWN"THEN910
880 IFN1>999THENN1=998
890 PRINT#1,"M";0;-N1
900 PRINT#8
910 IFB1$<>"HOME"THEN930
920 PRINT#1,"H"
```

```
930 IFB1$<>"CIRCLE"THENGOTO1010
940 IFN1+N3>480ORN2+N4>999ORN5>15ORN1-N3
<0THENPRINT"OUT OF BOUNDS"
950 PRINT#1,"M";N1;N2:PRINT#1,"I":PRINT#
5,N5
960 FORX=-3.2TO3.2STEP.04
970 Y=N3*COS(X):Z=N4*SIN(X)
980 IFX=-3.2THENPRINT#1,"R";Y;Z
990 PRINT#1,"J";Y;Z:NEXT
1000 PRINT#1,"M";0;0:PRINT#1,"I"
1010 IFB1$<>"BASIC"THEN1030
1020 SYS64738
1030 IFLEFT$(B1$,5)<>"TURN-"THEN1090
1040 FORX=LEN(B1$)TO6STEP-1:PRINT#4,1
1050 Y$=MID$(B1$,X,1):
1060 PRINT#1,"M",N1*(LEN(B1$)-X)
1070 PRINT#8,Y$;:NEXT:PRINT#4,0
1080 PRINT#8
1090 IFLEFT$(B1$,5)<>"TURN/"THEN1130
1100 PRINT#4,1:FORX=6TOLEN(B1$)
1110 Y$=MID$(B1$,X,1)
1120 PRINT#8,Y$:NEXT:PRINT#8:PRINT#4,0
1130 IFB1$<>"SCREEN"THEN1160
1140 IFN1>15ORN2>15THENN1=0:N2=2
1150 POKE53280,N1:POKE53281,N2
1160 IFB1$<>"0-SET"THEN1190
1170 PRINT#2,0:PRINT#3,1:PRINT#4,0:PRINT
#5,0:PRINT#6,0
1180 POKE53280,0:POKE53281,2:PRINT"E"
1190 IFB1$<>"DIRECT"THEN1250
1200 IFN1>255ORN2>255THEN1230
1210 PRINT#6,N1:PRINT#3,N2
1230 POKE198,0:WAIT198,1
1240 GETA$:IFA$<>"e"THENPRINTA$;:PRINT#8
,A$;:GOTO1230
1250 IFB1$<>"PORT"THEN1290
1260 IFN1>2ORN1<1THENPO=56320
1270 IFN1=1THENPO=56321
1280 IFN1=2THENPO=56320
1290 IFB1$<>"JOY"THEN1680
1320 PRINT:RESTORE:FORF=0TO3
1330 READB$(F)
1340 NEXT
1350 REM====1520 SETUP=================
1360 SW=3:C=0
1400 PRINT#2,C
1410 REM====READ QSTICK==============
1420 GETA$:J=PEEK(PO)
1430 IF(JAND1)=0THENY=Y+SW
1440 IF(JAND2)=0THENY=Y-SW
```

```
1450 IF(JAND4)=0THENX=X-SW
1460 IF(JAND8)=0THENX=X+SW
1470 IF(JAND16)=0ANDKK=0THENKK=1:GOTO155
0
1480 IF(JAND16)=0ANDKK=1THENKK=0
1490 IFA$=CHR$(133)THEN8000
1500 IFA$=CHR$(134)THENSW=SW+1
1510 IFA$=CHR$(138)ANDSW>1THENSW=SW-1
1520 IFA$=CHR$(135)THENC=C+1
1530 IFC=4THENC=0
1540 REM====X FOR OVERFLOW==========
1550 IFX<0THENX=0:GOTO1400
1560 IFY<-480THENY=-480:GOTO1400
1570 IFX>480THENX=480:GOTO1400
1580 IFY>480THENY=480:GOTO1400
1590 REM====
=======================
1600 IFKK=1THENK$="D"
1610 IFKK=0THENK$="M"
1620 PRINT#1,K$;X;Y
1630 PRINT"STEP";SW,B$(C)"
1640 PRINT"qq"
1650 IFSW<10THENPOKE1070,32
1660 GOTO1400
1670 DATABLACK,BLUE,GREEN,RED
1680 IFB1$<>"PAUSE"THEN8000
1690 WAIT203,63
8000 REM====0======================
8010 N1$="":N2$="":N3$="":N4$="":N5$=""
8020 X$="":Y$="":N1=0:N2=0:N3=0:N4=0:N5=
0
8030 E=E+1:IFB1$<>""THENB1$="":GOTO350
8040 CLR:RUN
9000 REM====STATUS ================
9010 PRINT"sE   *** 1520 PLOTTER UTILITY
 PROGRAM ***"
9020 RETURN

READY.
```

Now that you've typed the listing in, here are a few fundamentals about the program. Lines 10-650 contain the initialization and input routines, followed by the command set mentioned above (lines 660-1690). Lines 8000-8040 reset variables; lines 9000- form the status line. All of these subprograms are executed in concert with numerous string operations.

The program was designed to make certain items in plotting available to the user as "single-word" commands, and to let the user add his or her own. This is the reason for the huge gap between line 1700 and line 7990; once you've worked with the program for a while, you may want to extend its abilities to suit your own needs.

Commands are input as a command word, followed by a series of up to five numbers. Each command and number sequence must be separated by commas (e.g., CIRCLE,100,100,90,80,0).

LIST OF COMMAND WORDS:

COLOR,X X=0=black
X=1=blue
X=2=green
X=3=red

PEN,X moves the drum with the specified pen color to the change position at the right end of the roller (X represents pen color [see above]).

LINE,X1,X2,Y1,Y2,L draws a line from X1,Y2 to X2,Y2; line type (L) can be from 0 to 15.

SIGN,X designates the character size: Values between 0 (small print) and 3 (large print) work best.

TEST prints the word TEST on paper.

PAPER UP,X moves the paper up X plotter steps (one step = 0.2 mm).

PAPER DOWN,X moves the paper down X steps.

HOME moves the drum to home position, indicating the absolute or relative null-point.

CIRCLE,Mx,My,Rx,Ry,L draws a circle or ellipse:
Mx = middle-point coordinate for X-axis
My = middle-point coordinate for Y-axis
Rx = X-register radius in plotter steps
Ry = Y-register radius in plotter steps
L = line type (0-15)

BASIC forces a system reset, which ends and clears the program.

TURN-TEXT,X turns the word TEXT or any other input 90 degrees toward the X axis, printed at X plotter steps.

TURN/TEXT turns the word or other input 90 degrees to the Y-axis.

SCREEN,R,B changes screen and border color (numbers from 0-15). R stands for border or Rim, B for screen proper or Background. The colors match the following numbers:

0	black	8	orange
1	white	9	brown
2	red	10	light red
3	cyan	11	gray 1
4	purple	12	gray 2
5	green	13	light green
6	blue	14	light blue
7	yellow	15	gray 3

0-SET will reset all values to default value.

DIRECT,X,Y allows us to use the plotter in typewriter mode:
X = 0 upper-case, with SHIFTed lower-case
X = 1 lower-case, with SHIFTed upper-case
Y controls character size (between 0 and 3).

PORT,X allows use of the joystick ports (X=1 or 2). Default is Port 2.

JOY switches joystick reading on and off, also defining the function keys as follows:

F1	=	continue program
F3	=	raise plotter step-width by 1
F4	=	decrease step-width by 1
F5	=	change color

PAUSE stops the program until a key is pressed.

Type in a command, then the <RETURN> key. To execute the command, press * -- the plotter will execute, and wait for the next command. If * isn't pressed, the program awaits the next command(s) attached to the first. It's possible to use up to 100 commands a line this way.

On to the program itself:

Lines 10-90 open the necessary plotter files and dimension the arrays. The GOSUB to 9000 sets up the status line, and places it onscreen.

The input routine from line 100 is a string operation that reads individual characters using a GET statement. Line 140 blocks off any acknowledgement of cursor keys or screen operations (such as CLR). Line 210 places all inputs into T$. If command chains exceed 70 characters, a <RETURN> is forced automatically. Line 200 checks whether the command to execute is given. If so, a jump is made to 330; if not, the system waits for the next command.

Every <RETURN> sends the program from line 150 to 240, where the command is stored in EN$(E). Line 270 checks for spaces, up-arrow or left-arrow, and compresses the command accordingly. For example, if your typed CIRCLE as C IR C LE, the program would remove all blank spaces. All our input flows as one long string into EN$(E).

If all commands are ended with an asterisk (*), the read routine proceeds (line 330). This has the job of separating and storing command and number sequence. This is why the commas are used for separation.

Lines 350-380 will read the command word, and store it in B1$. Line 380 will read the first number after the first comma, and store it in N1$; the second in N2$, and so on up to N5$.

Lines 600-650 only handle numerical variables (N1-N5).

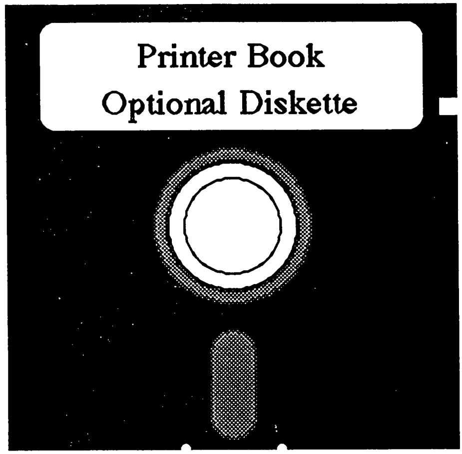
Now the input routine has ended. Individual commands are worked out from line 660. If you wish to improve this program, here is an example. Let's take CUBE, and show you which material goes into which variable:

CUBE	=B1$
side-length	=N1
starting pt.	=N2
color	=N3
turn	=N4
end position	=N5

Any variable can be used; any bad input will simply start the input routine again.

Lines 8000- reset all variables to starting value, to get the arrays ready for the next command.

You'll find these commands a great help in your programming efforts. We hope that you've found out just how useful your 1520 plotter can be -- limited only by your own creativity.



For your convenience, the programs that are listed in this book are available on a 1541 formatted diskette. If you want to use the programs, without typing them in from the listings in the book, you may want to order this diskette.

All programs on the diskette have been fully tested. The diskette is available for $14.95 + $2.00 ($5.00 foreign) for postage and handling charges.

When ordering, Please specify the title of the diskette, your name and shipping address and enclose a check, money order or credit card information. Mail your order to:

ABACUS Software
P.O. Box 7211
Grand Rapids, MI 49510

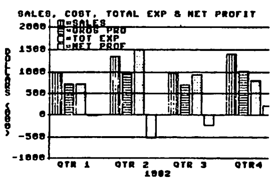
Call today for the name of your nearest local dealer
Phone (616) 241-5510

Make your '64 work fulltime

MAKE YOUR OWN CHARTS...

CHARTPAK-64

produces professional quality charts and graphs instantly from your data. 8 chart formats. Hardcopy in two sizes to popular dot matrix printers. $39.95
ISBN# 0-916439-19-4

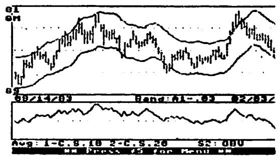


Also Available **CHARTPLOT-64** for unsurpassed quality charts on plotters. ISBN# 0-916439-20-8 $84.95

CHART YOUR OWN STOCKS...

TAS-64

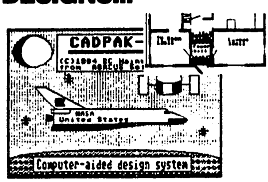
sophisticated technical analysis charting package for the serious stock market investor. Capture data from DJN/RS or Warner services or enter and edit data at keyboard. 7 moving averages, 3 oscillators, trading bands, least squares, 5 volume indicators, relative charts, much more. Hardcopy in two sizes, most printers. ISBN# 0-916439-24-0 $84.95



DETAIL YOUR DESIGNS...

CADPAK-64

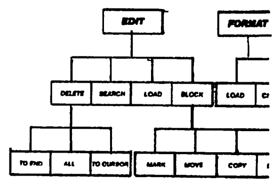
superb lightpen design tool. exact placement of object using our Accu-Point positioning. Has two complete screens. Draw LINEs, BOXes, CIRCLEs, ELLIPSEs; pattern FILLing; freehand DRAW; COPY sections of screen; ZOOM in and do detail work. Hard copy in two sizes to popular dot matrix printers. ISBN# 0-916439-18-6 $49.95



DO YOUR OWN WORD PROCESSING

TEXTOMAT-64

flexible worprocessing package supporting 40 or 80 columns with horizontal scrolling. Commands are clearly displayed on the screen awaiting your choice. Quickly move from editing to formatting to merging to utilities. Will work with virtually any printer. ISBN# 0-916439-12-7 $39.95



CREATE SPREADSHEETS & GRAPHS...

POWER PLAN-64

not only a powerful spreadsheet packages available, but with built in graphics too. The 275 page manual has tutorial section and HELP screens are always available. Features field protection; text formatting, windowing; row and column copy, sort; duplicate and delete.
ISBN# 0-916439-22-4
$49.95

Coordinate: C/10 POWER PLAN-64

1:1	A	B	C
1	Sales	Jan	Feb
2	Distributors	47.2	54.2
3	Retailers	27.9	35.4
4	Mail Order	18.5	23.7
5		----------	----------
6		93.6	113.3
7			
8	Expenses		
9	Materials	8.2	9.2
10	Office	2.0	2.8
11	Shipping	4.4	5.0
12	Advertising	12.9	13.8
13	Payroll	10.5	10.7
14		----------	----------
15		38.0	41.5
16			
17	Profit	55.6	71.8

ORGANIZE YOUR DATA...

DATAMAT-64

powerful, yet easy-to-use data management package. Free form design of screen using up to 50 fields per record. Maximum of 2000 records per diskette. Complete and flexible reporting. Sorting on multiple fields in any combination. Select records for printing in desired format. ISBN# 0-916439-16-X $39.95

INVENTORY FILE

Item Number___ Description___

Onhand ____ Price ____

Location ____

Reord. Pt.____ Reord. Qty.___

Cost ____

Other titles available. For FREE CATALOG and name of nearest dealer, write or call (616) 241-5510. For postage and handling, include $4.00 ($6.00 foreign) per order. Money Order and checks in U.S. dollars only. Mastercard, VISA and American Express accepted. Michigan residents include 4% sales tax.

CANADA: Book Center, Montreal (514) 332-4154

FREE PEEKS & POKES POSTER WITH SOFTWARE

For name & address of your nearest dealer call (616) 241-5510

You Can Count On

Abacus Software

P.O. Box 7211 Grand Rapids, MI 49510 - Telex 709-101 - Phone 616/241-5510

PRINTER BOOK FOR THE COMMODORE-64 AND VIC-20

This book gives comprehensive coverage of using printers with the Commodore 64 (and VIC-20). You'll learn how to print text as well as high resolution screen dumps. The Commodore printers — 1525, MPS801 and 1526 and explained in detail. Expanded coverage of the 1520 plotter is also explained. You'll also learn how to work with other printers like the Epson and compatibles. Over 50 example programs are included that demonstrate screen dumps, word processors, 3-D graphics and much more.

ISBN 0-916439-08-9

YOU CAN COUNT ON Abacus Software

P.O. BOX 7211 GRAND RAPIDS, MICH. 49510 PHONE 616-241-5510