LASER

COLOR COMPUTER

350 / 500 / 700

MAIN UNIT MANUAL

VTECH

PREFACE

The LASER 350/500/700 Main Unit Manual has been written to serve as a tutorial as well as reference source for the LASER 350/500/700 user. These three models of personal computer are built around the same CPU Z80A and they have similar structure. In fact, they share the same resident ROM BASIC Interpreter.

The manual is organized in three parts. Part I contains general information such as how to connect the system, how to care for the computer and specifications for the three models.

Part II (BASIC Programming Manual) is written for those who wish to learn BASIC programming on this series of computer. Moreover, it can also serve as a general introduction to programming in BASIC on any system. BASIC commands are presented in sequence, from the very simple "PRINT" command, to more complex Assembly Language Subroutine Calling.

Part III is written in particular for the more advanced users of computer and for the electronics hobbyists who wish to probe the internal structure of this series of computer. Hardware design details, memory organization and software aspects are discussed. Novice users may skip this part.

Finally, the appendix at the end of this manual serves as a constant reference tool while you progress with your computer.

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

TABLE OF CONTENT

PART 1 USER'S MANUAL

PART 2 BASIC REFERENCE MANUAL

INDUCTION

CHAPTER 1

CHAPTER 2

CHAPTER 5

APPENDIX

PART I

USER'S MANUAL

Congratulations on buying the LASER 350/500/700!

You have bought one of the finest home computers available today.

Before starting to operate your computer, we suggest that you read this part of the manual carefully.

Moreover, you will need this manual in the future. So you should not throw it away. Rather you should have it at hand as you progress along the computing highway.

WARNING: To prevant fire or electric shock, do not expose this computer to rain or moisture.

No user servicable parts inside. Do not open the cabinet of the computer unless you are instructed to do so, such as when installing an expansion card.

IMPORTANT INFORMATION

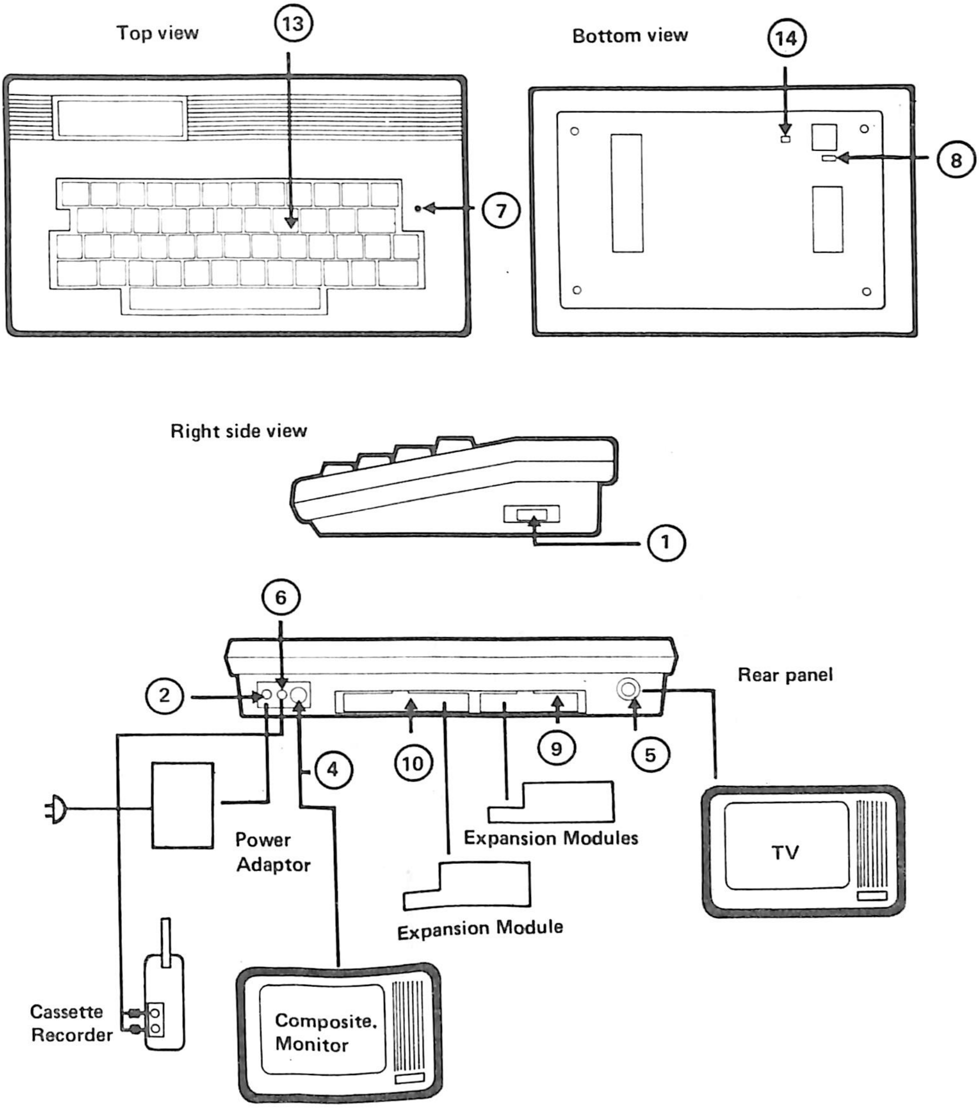
The following message is valid for NTSC FCC versions only, so it may or may not apply to your model.

This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict, accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:
Reorient the receiving antenna
Relocate the computer with respect to the receiver
Move the computer away from the receiver
Plug the computer into a different outlet so that computer and receiver are on different branch circuits.
If necessary, the user should consult the dealer or an experienced radio/ television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful:
'How to Identify and Resolve Radio-TV Interference Problems'.
This booklet is available from the U.S. Government Printing Office, Washington, DC20402, Stock No. 004-000-00345-4.

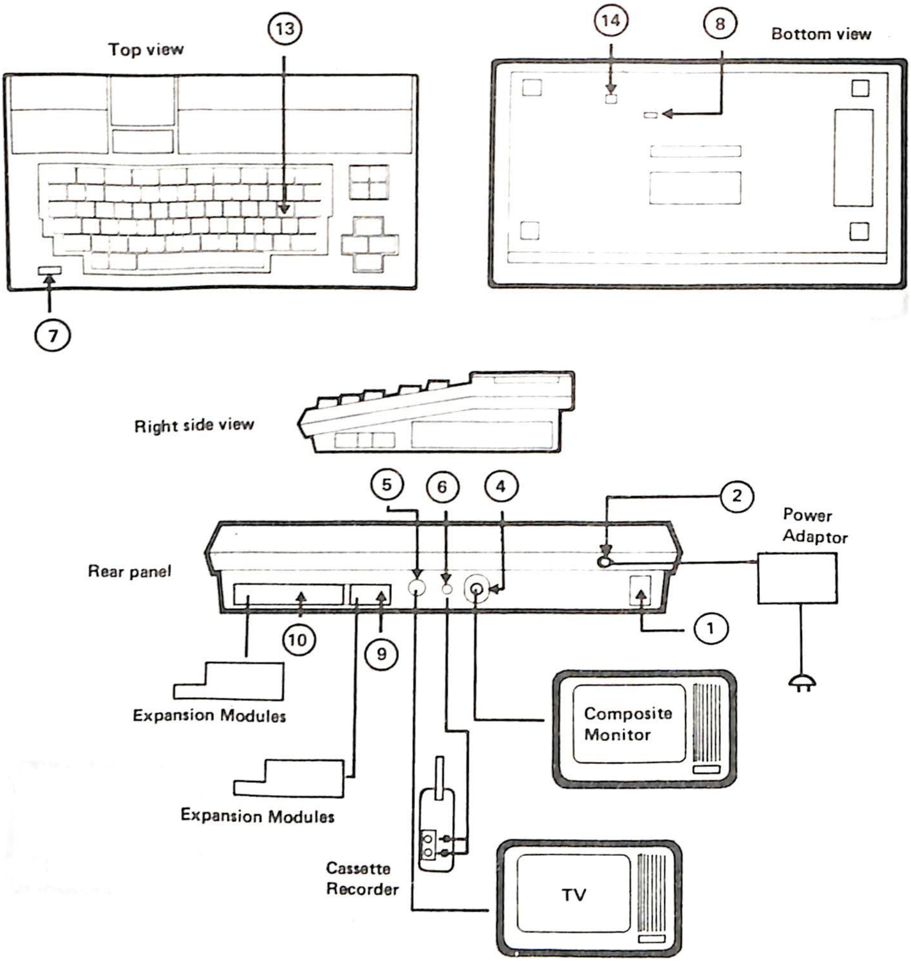
WHAT SHOULD BE IN THE BOX

1. This Manual.
2. Your Color Computer main unit.
3. A power adaptor (for LASER 350/500 only).
4. A cord to connect your computer to the TV Antenna Outlet.

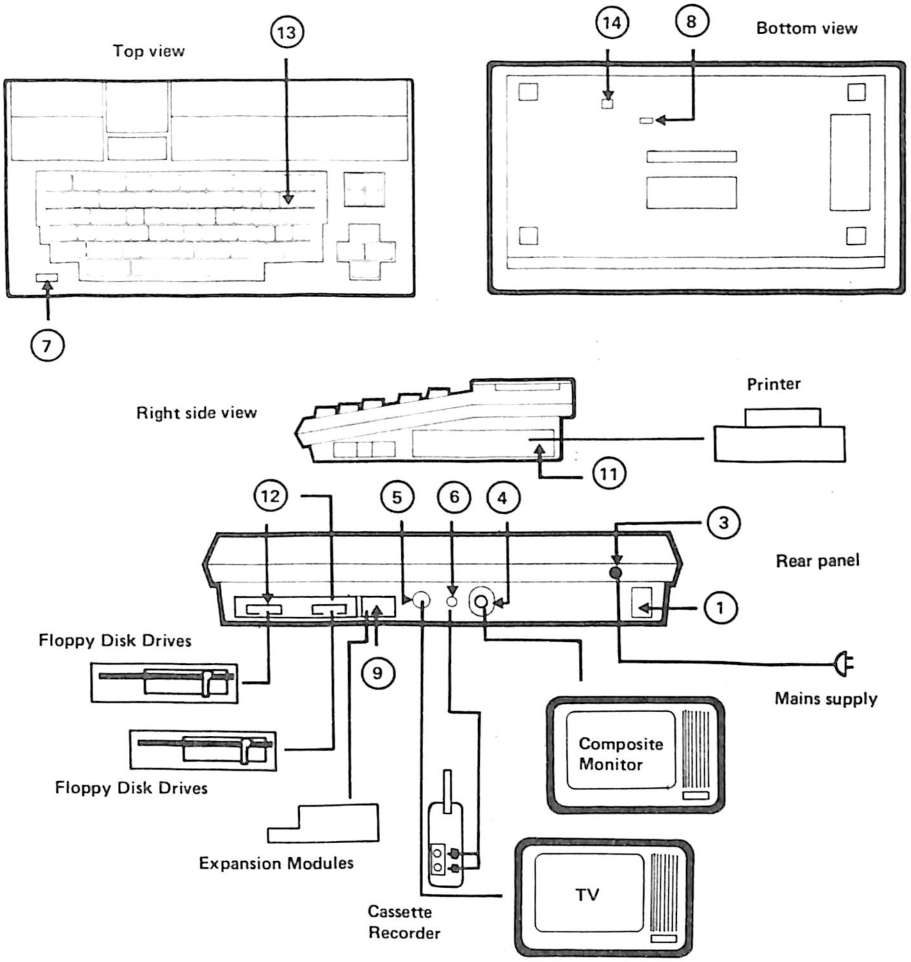
To get your compuer working you will need a color TV, a B/W TV or a composite green monitor.


Top view
13
Bottom view
14
7
8
Right side view
1
6
2
Rear panel
4
10
9
5
Power
Adaptor
Expansion Modules
Expansion Module
TV
Cassette
Recorder
Composite.
Monitor

LASER 350


Top view
13
7
14
8
Bottom view
Right side view
5
6
4
2
Power Adaptor
Rear panel
1
10
9
Expansion Modules
Composite Monitor
Expansion Modules
Cassette Recorder
TV

LASER 500


Top view
13
14
8
Bottom view
7
Right side view
Printer
11
12
5
6
4
3
Rear panel
1
Floppy Disk Drives
9
Mains supply
Composite Monitor
Floppy Disk Drives
Expansion Modules
TV
Cassette Recorder

LASER 700

Connection Diagrams and Switches

1. Power ON/OFF Switch.
2. Power Adaptor Socket (For LASER 350/500)
3. Main Power Socket (For LASER 700 only)
4. Composite Monitor Output Socket.
5. TV Output Socket.
6. Cassette Interface Jack.
7. Power Indicator light.
8. Color Defeat Switch (For NTSC & PAL versions only)
9. Expansion Bus (P 1)
10. Expansion Bus (P2) (For LASER 350/500)
11. Printer Connector (For LASER 700 only)
12. Floppy Disk Drive Connector Sockets (For LASER 700 only)
13. Keyboard.
14. Channel Select Switch (For NTSC & PAL versions only)

SETTING UP THE LASER 350/500/700.

IMPORTANT: PLEASE MAKE SURE THAT ALL COMPUTER PERIPHERALS ARE TURNED OFF BEFORE YOU MAKE ANY CONNECTIONS.

(1) Connect the composite monitor cable from composite video output socket of your computer to your composite color/green monitor. If you are using a domestic TV as a display, you should connect the TV cable from the TV output socket of your computer to the TV antenna outlet of your TV set.

(2) Connect the cassette cable from the cassette socket of your computer to your data cassette recorder/audio cassette player. On the recorder side of the cable, the black colored plug should be inserted into the Mic input while the red colored plug should be inserted into the Ear output. If you are not using a cassette for storage, then you may skip this step.

(3) If you decide to use any expansion modules such as Printer Interface, Joystick Interface or Floppy Disk Drive Interface, just remove the cover from the operate and insert the interface module. (BE SURE TO TURN OFF THE POWER BEFORE INSERTING OR REMOVING ANY EXPANSION MODULE)

(4) Connect the power cable (For LASER 700 only) to your mains supply. If you are using LASER 500 or LASER 350, you need to connect the DC power adaptor plug to your computer's DC power input and connect the other end of the power adaptor to your mains supply.

(5) Double check that everything is properly installed before turning on the POWER ON/OFF SWITCH of your computer.

(6) The indicator light should glow as you turn on power to your computer. The following message should be displayed on your TV/monitor screen:

```
VIDEO TECHNOLOGY
BASIC V3.0
READY
```

If this display does not appear, TURN OFF THE POWER IMMEDIATELY.
Double check everything and try again. If it still don't work, follow the instructions in the following "Trouble Shooting Checklist".

TROUBLE SHOOTING CHECKLIST

SYMPTOM

1. No Indicator Light
   - The Mains supply not turned on or badly connected.
   - The Power Adaptor not properly connected or badly contacted.
   - Power switch not turned on.
   - The power plug badly connected.

2. No Screen Display
   - Aerial cord not properly connected or badly contacted.
   - The TV set not properly turned.
   - Misconnection between the TV set and the Monitor socket.
   - Misconnection between the Monitor and TV socket.

3. Screen Display without the READY message
   - Improper set-up procedure. Switch off the power for a while and switch on aqain.
   - For LASER 700 user without disk drive, the RESET key need to be pressed to enter BASIC after power up.

4. Abnormal Performance
   - Improper set-up procedure. Switch off the power for a while and switch on again.

5. Cassette Loading and Saving Not Working
   - Cassette Interface Cord not properly connected or badly contacted.
   - The tone and volume level of the cassette recorder not set at proper range.(Please refer to Basic Programming Manual)

If none of the above work, please contact your nearest Dealer.

HOW TO CARE FOR YOUR COMPUTER

1. Do not attempt to probe the inside of your computer. Dangerous high voltage is used inside. Contact your dealer for service if necessary.

2. Keep the computer main unit, expansion modules and TV/monitor display away from excessive heat, humidity, dust and any form of moisture.

3. Do not cover the ventilation holes on the top and bottom of the cabinet during operation. Otherwise, excessive heat might develop inside your computer.

4. Do not use thinner or any oil or petroleum based cleaner for the cabinet or the keyboard. It may damage the surface finish. Use only mild soap or detergent solution for cleaning. Ensure that the POWER is switched off during cleaning.

5. Do not drop the main unit. Handle it with care.

DIFFERENCES BETWEEN LASER 350/LASER 500/LASER 700

This manual is written for the three models LASER 350, LASER 500 and LASER 700. These three models share the same 32K bytes resident ROM (Read-Only-Memory) which contains an Extended Microsoft BASIC Interpreter.

Software features and hardware circuitry are basically the same for all threee models except in the following areas:

(1) Keyboard and Cabinet.
The cabinet of LASER 500 and LASER 700 are the same size, whereas the LASER 350 is slightly smaller.

All three models are equipped with sculptured, typewritter style, alphanumeric, full stroke keyboard. In addition, the LASER 500 and the LASER 700 have CURSOR KEYS, RESET, TAB, GRAPH, CAP LOCK, BACK SPACE, ESCAPE and Editing keys along with ten Programmable Function Keys.

Most of these special key functions can be generated on the LASER 350 by CONTROL key combinations.

(2) Built-in RAM (RANDOM ACCESS MEMORY)
The LASER 350 comes with 16K Bytes of on board RAM, expandable to 144K Bytes RAM with optional RAM Expandion Modules.

The LASER 500 comes with 64K Bytes of on board RAM with optional RAM Expansion Modules.

In the LASER 700, two versions of built-in RAM are offered; the 64K Byte version and 128K Byte version. Both are expandable to 144K Byte with optional RAM Expansion Modules.

(3) Graphics Modes
The BASIC Interpreter of the three models, as well as the hardware circuitry on-board, is capable of supporting two text modes (40 column and 80 column Text) and six graphics modes (GR0, GR1, GR2, GR3, GR4 and GR5).

However, some of the graphic modes require large amount of display memory (RAM). So you may not be able to invoke some graphics modes if your model do not possess enough memory.

Graphics modes GR3, GR4, GR5 are not available to LASER 350 users, unless they have installed an optional 64K RAM Expansion Module. Both 40 column and 80 column text modes are available to LASER 350 users, however, even without any RAM Expansion.
All graphics modes and text modes are avialable to LASER 500/ LASER 700 users, who have at least 64K RAM on-board.

(4) Power Supply
The LASER 350 and the LASER 500 comes with an external Power Adaptor Module as standarad accessory.

LASER 700 has a power supply built into the main unit.

(5) Printer Interface.
The LASER 700 has a built-in Printer Interface for Centronics Printers.

For the LASER 350 and the LASER 500, optional Printer Interface modules are available.

(6) Floppy Disk Drive Interface
The LASER 700 has a built-in Floppy Disk Drive Interface which supports two 5¼" Floppy Disk Drives. There are provisions for two Drive connectors on the back panel of the LASER 700.

For the LASER 350 and the LASER 500, optional Floppy Disk Drive Interface Modules are available, which support two 5¼" Floppy Disk Drives.

BASIC statements and programs are compatible on all three models provided that memory requirements do not exceed the limits available. For instance, a program on a LASER 500, which utilizes graphics mode GR5, will not be able to RUN on a LASER 350 unless a 64K RAM Expansion Module is installed on the LASER 350. But other than this memory limitation, all program are 100% compatible.

SPECIFICATIONS

LASER 350

CPU	:	Z80A Running at 3.7MHz	
Built-in ROM	:	32K Bytes Extended Microsoft BASIC	
Built-in RAM	:	16K Bytes	
RAM Expandable to	:	144K Bytes (optional expansion module)	
Built-in text modes	:	40 columns X 24 rows	16 colors
		80 columns X 24 rows	2 out of 16 colors
		upper/lower case letters	
Built-in graphics modes	:	320H X 192V	2 out of 16 colors
		160H X 192V	16 colors
		160H X 96V	16 colors
		*640H X 192V	2 out of 16 colors
		*320H X 192V	16 colors
		*160 X 192V	16 colors

*64K RAM Expandion Module required

Operating system supported : DOS (disk drive required)
CP/M (disk drive required 64K RAM Expansion Required)

Keyboard : 49 keys Sculptured, Typewriter Style, Full-stroke Keyboard with Space Bar, Programmable Beep/No Beep Key Entry, All keys Auto-Repeat.

Other Built-in Features : Programmable Foreground/Background/Backdrop color, Full On-screen Editing, Single key command entry, Graphics Character, Single Channel Sound Output, Cassette Interface at 600 Baud, Composite Video Output, RF Output for Domestic TV, DC Power Adaptor.

Expansion (options) : 64K/128K RAM Memory Expansion Module, Centronics Printer Interface, Joystick Interface, Floppy Disk Drive Interface. Light Pen Interface, Fast Cassette Interface, RS232 Interface, Chinese Card. AD/DA Interface.

LASER 500

CPU : Z80A Running at 3.7MHz
Built-in ROM : 32K Bytes Extended Microsoft BASIC
Built-in RAM : 64K Bytes
RAM Expandable to : 144K Bytes (optional expansion module)
Built-in text modes : 40 columns X 24 rows 16 colors
80 columns X 24 rows 2 out of 16 colors
upper/lower case letters
Built-in graphics modes : 320H X 192V 2 out of 16 colors
160H X 192V 16 colors
160H X 96V 16 colors
640H X 192V 2 out of 16 colors
320H X 192V 16 colors
160H X 192V 16 colors
Operating System Supported : DOS (disk drive required)
CP/M (disk drive required)

Keyboard :77 keys Sculptured, Typewritter Style, Full-stroke keyboard with Space Bar, Cursor Keys, Screen Editing Keys, Tab, Cap Lock, Reset and 10 Programmable Function Keys, Programmable Beep/No Beep Key Entry, All Keys Auto-Repeat.

Other Built-in Features : Programmable Foreground/Background/Backdrop color, Single Key Command Entry, Graphics Characters, Single Channel Sound Output, Cassette Interface at 600 Baud, Composite Video Output, RF Output for Domestic TV, DC Power Adaptor.

Expansion (Options) :64K/128K RAM Memory Expansion Module, Centronics Printer Interface, Joystick Interface, Light Pen Interface, Fast Cassette Interface, RS232 Interface, AD/DA Interface, Chinese Card, Floppy Disk Drive Interface.

LASER 700

CPU	:	Z80A Running at 3.7MHz
Built-in ROM	:	32K Bytes Extended Microsoft BASIC
Built-in RAM	:	64K/128K Bytes
RAM Expandable to	:	144K Bytes (optional expansion module)
Built-in text modes	:	40 columns X 24 rows 16 colors 80 columns X 24 rows 2 out of 16 colors upper/lower case letters
Built-in graphics modes	:	320H X 192V 2 out of 16 colors 160H X 192V 16 colors 160H X 96V 16 colors 640H X 192V 2 out of 16 colors 320H X 192V 16 colors 160H X 192V 16 colors
Operating System Supported	:	DOS (disk drive required) CP/M (disk drive required)
Keyboard	:	77 keys Sculptured, Typewritter Style, Full-stroke, Keyboard with Space Bar, Cursor Keys, Tab, Screen Editing Keys, Cap Lock, Reset and 10 Programmable Function Keys, Programmable Beep/No Beep Key Entry, All Keys Auto-Repeat.
Power Supply	:	Built-in Power Supply Unit for Main Unit and Floppy Disk Drive
Printer Interface	:	Built-in Centronics Printer Interface
Floppy Disk Interface	:	Built-in Floppy Disk Interface for two external 5¼″ Floppy Disk Drives
Other Built-in Features	:	Programmable Foreground/Background/Backdrop color, Full on-screen Editing, Single Key Command Entry, Graphics Characters, Single Channel Sound Output, Cassette Interface at 600 Baud, Composite Video Output, RF Output for Domestic TV, RGB Monitor Interface
Expansion (options)	:	64K RAM Memory Expansion Module, Joystick Interface, Light Pen Interface, RS232 Interface, Chinese Card

PART II

BASIC PROGRAMMING MANUAL

INTRODUCTION

This part of the manual is intended for people who want to learn to program in BASIC on the Color Computer. With a little time and effort you will very soon discover that there is nothing very difficult about learning how to program your computer. You will be introduced to the fundamentals of BASIC and to the procedures of programming. Things are explained one at a time and step by step. All you have to do is to start at the beginning and make sure you try everything as it comes up. Take your time. Understand one step befoe going to the next.

The key to success is to try everything. It is not enough to read about it. You must do it. You don't learn to play the piano, type or swim by reading a book. You learn by doing. Don't worry about making mistake, just correct the mistake and continue. The computer doesn't worry about it, why should you? There is nothing that can be done from the keyboard that can damage your computer. Cautions are included in the text when statements that might DESTROY data files are introduced. Everywhere else, feel free to try things out with your computer at every stage of learning.

In general, you should follow the SEQUENCE of presentation given in the text. However, Chapter 17 which discusses the use of tape storage may be read at anytime when you wish to save a program on the cassette tape. While this manual is written for one who wishes to learn to program in BASIC on this computer, it can also serve as a general introduction to programming in BASIC on any system. Just remember that the BASIC language has many forms. There are often slight differences between one implementation of BASIC and another.

Finally, this manual will not only help you to understand BASIC but it will also help you to understand the fundamentals of computer programming in general.

Have fun with your color computer!

CHAPTER 1

THE COMPUTER

- What is a computer?
- What makes up a computer
- What is a program?
- Computer languages
- BASIC

WHAT IS A COMPUTER?

A computer is a device which performs various operations based on instructions given by the person who uses it. The computer cannot tell the user how to solve a problem. It has to be told what to do. The computer cannot think, Yet it is a very effective tool in the hands of a competent and experienced user.

A computer system consists of a number of machines or devices where operations are coordinated by a central control unit. These machines when working together are able to perform simple logical and arithmetic processes such as comparing two numbers. They can also read in information, store this information and give out results in a form understandable by us.

WHAT MAKES UP A COMPUTER SYSTEM?

Generally a computer system consists of the following units:

i) Central Processor Unit (CPU)
This can be considered the brain of the computer system. It performs operations specified in the instructions, such as arithmetic and logical operations.

ii) Memory Unit — Information and instructions given by the user or generated by the computer are stored here. This unit is inside the computer. The CPU gets information from it directly.

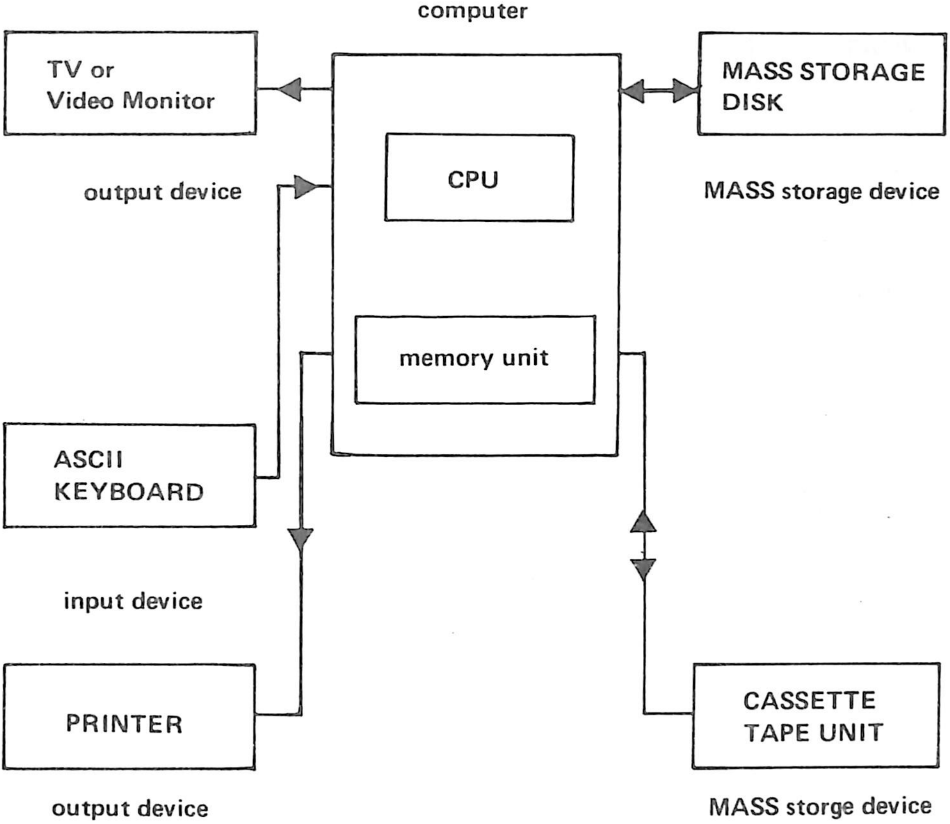
iii) Mass Storage Unit — This unit is outside the computer. It stores instructions and information given by the user or generated by the computer. The tape storage unit and the floppy disk units are examples of mass storage units. Information stored in these units has to be transferred to the internal memory unit before the CPU can process it.

iv) Input Device — As the name suggests this allows the user to enter instructions or information to the computer. The keyboard is an example of an input device.

v) Output Device — This receives information or results sent from the computer. Examples are the printer and the TV screen.

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

Although computer systems vary in size, all practical computer systems require the above mentioned units.



Configuration of a Computer System in general

WHAT IS A PROGRAM?

A program is a set of instructions. The process of specifying a set of instructions for a computer is called programming. The individual preparing a program is called a programmer. The programmer feeds in or 'inputs' a series of instructions (program) which tells the computer the steps to take to complete the task required of it.

COMPUTER LANGUAGES

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

There are many programming languages in use today. Some are designed for very specialised applications. Others are designed for more general use. BASIC is a language in the latter category.

BASIC

BASIC, an acronym of Beginners' All-Purpose Symbolic Instruction Code, is a powerful programming language. BASIC has a simple English vocabulary, few grammatical rules and it resembles ordinary mathematical notation. To instruct your computer you must know BASIC. It will be introduced gradually and explained at each step.

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

The BASIC described in this manual is common for the three color computers LASER 350/500/700. Most of the BASIC commands and functions are identical in the three different models except that some graphic mode commands are not available for the model with less memory. For details please refer to the chapter on graphic commands.

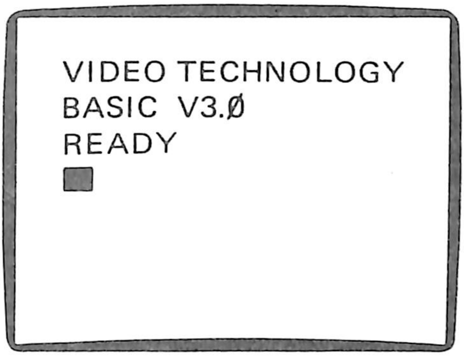
CHAPTER 2

HOW TO USE YOUR COLOR COMPUTER

- To start
- How to operate the keyboard
- The PRINT Command
- SYNTAX ERRORS
- EDITING
- INSERT
- A look ahead

TO START

When you have set up your computer and switched it on, your TV screen should look like this:



```
VIDEO TECHNOLOGY
BASIC V3.Ø
READY
■
```

READY is a prompt message telling you that the computer is waiting and, as the word suggests, is ready to receive your instructions. The flashing square, the CURSOR, tells you exactly where you are typing on the screen. That is, where any information you feed in from the keyboard will appear on the screen.

Your compuer has 2 different character display modes. One is the 40 columns X 24 rows characters display, as you can see on the screen above. The other is the 80 column X 24 rows characters display.

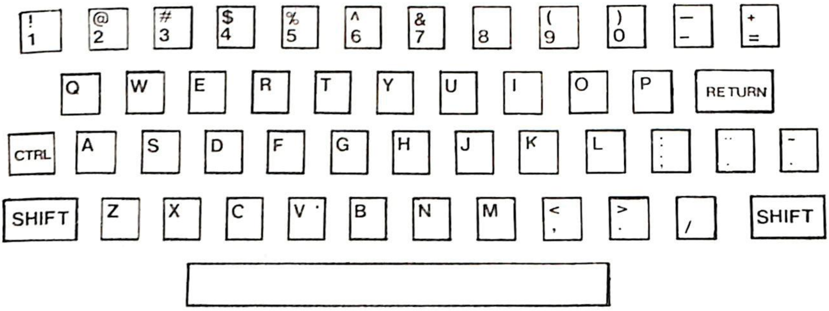
There are some "tricks" to select 40 or 80 column display at power up. To try it now, switch off your color computer and then keep holding the CTRL key when you switch on the computer again. Then you can see the smaller character display mode on the screen.

Of course, you can use BASIC commands to switch between 40 and 80 column display. You will come to more detail on these so called "TEXT" modes in later chapters.

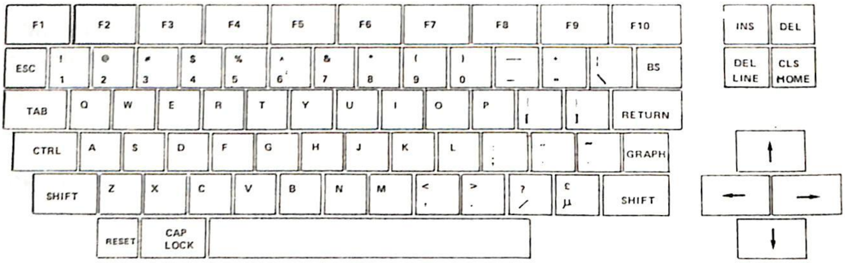
HOW TO OPERATE THE KEYBOARD

The keyboard layout of the LASER 350 and LASER 500/700 are quite different from each other. Some keys found on LASER 500 are not found on the LASER 350. However, their operations are similar. The following pages describes the keyboard of all three models in general. You may skip those parts not avialable on your model. Please note that some keys not found on LASER 350 keyboard can be replaced by CONTROL key combinations.

LASER 350 KEYBOARD



LASER 500/700 KEYBOARD


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10
INS DEL
DEL LINE CLS HOME
ESC ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 — - BS
TAB Q W E R T Y U I O P [ ] RETURN
CTRL A S D F G H J K L GRAPH
SHIFT Z X C V B N M < , > . ? / £ µ SHIFT
RESET CAP LOCK

The keyboard layout of LASER 350 and LASER 500/700 are shown above.

You will notice that these keyboard panels resemble a typewriter's, except for a number of special keys.

Although each key is labelled with only one, or not greater than 2 symbols, most of the keys are dual-function or even triple function. That is, different symbols can be obtained, or different functions can be performed by pressing that key alone, pressing the key with the SHIFT key, or pressing the key with the CTRL key.

To illustrate this, press the key 1 alone. The number "1" will be generated on the screen. If you press the same key 1 while holding the SHIFT key down, a symbol "!" will be generated instead. This rule applys to all other keys with two different symbols on the keytop.

All keys on the LASER 350/500/700 have auto-repeat feature. If you press a key and hold it down for about a second, it will repeat itself at about 10 characters per second until you release it.

Here are more detailed descriptions of the special keys.

RETURN

The RETURN key implies "Return control to the computer". After hitting this key, the computer will start interpreting whatever you have typed in before RETURN . If the computer does not understand you at all, it will mildly show its annoyance by giving you an error message.

Hitting RETURN has another effect; it performs a 'carriage return' on the screen, i.e., it will put the little flashing cursor at the beginning of the next line down, or it will cause the screen to scroll up if the display has already reached the bottom of the screen.

SHIFT

On either side of the keyboard there are two keys labelled SHIFT . They perform similar functions as shift keys on an ordinary typewriter.

These SHIFT keys do not generate any character when used on their own. When pressed with another key, a different character code will be generated. For instance, pressing SHIFT and 4 will produce a '$'.

CTRL

The CTRL key, meaning CONTROL, is located right above the SHIFT key on the left side of the keyboard. This key does not generate a keycode of its own, but only alters the codes of other keys. Some of the keys, when pressed with the CTRL key, will generate a series of codes that corresponds to a BASIC statement or perform special functions as listed in table:

CTRL key pressed with	BASIC statement code generated
& 7	{
* 8	}
(9	[
) 0	]
— –	\
+ =	¦
Q	FOR
W	TO
E	NEXT
R	RETURN
T	THEN
Y	ELSE
U	IF
I	INPUT
O	LET
P	PRINT

CTRL key pressed with	BASIC statement code generated
A	AUTO
S	STEP
D	DIM
F	GOSUB
G	GOTO
H	CLS
J	REM
K	RUN
L	LIST
Z	PEEK (
X	POKE
V	LPRINT
B	LLIST
N	NEW
M	DELETE

CTRL key pressed with	Special Function Performed
! 1	CAP LOCK Toggling
@ 2	TAB
# 3	Clear to End of Line
$ 4	Home Cursor
% 5	Clear Screen
∧ 6	Beep Buzzer
: ;	Insert
" '	Rubout
~ `	Graphic Character Enable
C	Break BASIC Program excution
< ,	← Cursor left
> .	→ Cursor right
? /	↑ Cursor up
SPACE BAR	↓ Cursor down

NOTE: When a key is pressed together with the SHIFT key and the CTRL key, the function performed or key code generated is the same as that by pressing that key with the SHIFT key only.

ESC (LASER 500/700 only)

Pressing the ESC key on the left of the keyboard changes the meaning of all keys that you press thereafter (known as ESCAPE SEQUENCE).

For instance, pressing B after you have pressed the ESC key will set the character to be displayed in inverse. That is, the Foreground/Background color for the text will be reversed.

Besides typing ESC directly from the keyboard, you may also invoke these Escape Sequence by means of the PRINT command.

Example:
```
PRINT CHR$(27); CHR$(66)
```

The screen is set to inverse display as mentioned above.

For more details of these ESCAPE SEQUENCES, Please refer to Appendix–D.

CAP LOCK (LASER 500/700 only)

From the keyboard, you can enter either upper or lower case letters. The CAP LOCK key, with a small indicating light on it, is like an electronic locking switch. It toggles between capital and lower case letter entry. When your LASER 500/700 is first turned on you may notice that the red light on the CAP LOCK key is on, indicating that all letters typed will be in capital.

Pressing the key once will turn off the light and changes the letters all to lower case. Pressing it once again will return to all capital letters.

LASER 350 users should note that pressing the 1! key while holding the CTRL key down has same function as the CAP LOCK key on the LASER 500/700.

GRAPH (LASER 500/700 only)

Your computer is designed to display characters as well as some predefined graphic symbols in TEXT mode. The GRAPH key must be pressed to use these "Graphic Characters".

Hitting the GRAPH key once puts you into Graphic Charater Entry Mode. You will stay in this mode and keep on entering Graphic Characters until you press the GRAPH key again, or the RETURN key.

For instance, the graphic symbol "♠" is entered by pressing GRAPH, then pressing the I key.

For a more detailed description of these graphic characters please refer to Appendix – G.

LASER 350 users should note that pressing the ~ key while holding the CTRL key down has the same effect as the GRAPH key on LASER 500/700.

TAB (LASER 500/700 only)

The TAB key is located above the CTRL key on the left side of the keyboard. Hitting the TAB key will move the cursor to the next pre-set TAB column to the right of the cursor.

The pre-set TAB COLUMNS are 9, 17, 25, 33 in 40 column display. While in 80 column display, they are 9, 17, 25, 33, 41, 49, 57, 65, 73. So, if the cursor is positioned in column 20, for instance, pressing TAB will move it to column 25.

LASER 350 users should note that pressing the 2 @ key while holding the CTRL key down has the same function as the TAB key on LASER 500/700.

BS (LASER 500/700 only)

BS means "Back Space". Pressing this key will cause the cursor move one place to the left, without erasing anything. You may use this key to go back and retype the characters that you want to alter.

You will soon recognize that the BS key is functionally equivalent to the left arrow key ← on the LASER 500/700 keyboard.

LASER 350 users should note that pressing the <, key while holding the CTRL key down has same function as pressing the BS , or ← key on LASER 500/700.

CURSOR KEYS: ↑ ↓ ← → (LASER 500/700 only)

These CURSOR KEYS (or ARROW KEYS) located at the lower right-hand corner, perform cursor movements in the direction their arrows indicate.

These cursor keys are used for on-screen editing supported by the built-in BASIC V3.0, to edit your programs.

A more detailed description of BASIC Program Editing can be obtained in later Chapters of this manual.

LASER 350 users should note that pressing the <, , >. , ?/ or the SPACE BAR together with the CTRL key has a similar effect to pressing the ← , → , ↑ , ↓ keys on the LASER 500/700.

SCREEN EDITING KEYS: INS DEL DEL LINE CLS HOME

(LASER 500/700 only)

These keys, located at the right side of the keyboard, are grouped as Screen Editing Keys because they allow the user to perform editing functions by pressing one single key.

INS This key stands for "Inserting" characters, while the DEL key stands for "Deleting" characters.

DEL LINE actually menas"Delete To End of the Line". This key allows quick deleting of the characters on the rest of the line.

CLS HOME is a dual function key. Pressing it alone will bring the cursor to its "HOME" position, ie., at the top left hand corner of the screen. The remaining parts of the screen is not changed. Pressing it with the SHIFT key will clear the Text Screen and place the cursor to its "HOME" position.

FUNCTION KEYS

F1 F2 F3 F4 F5
F6 F7 F8 F9 F10

(LASER 500/700 only)

The function keys, located on the top row of the keyboard, allow the user to enter a BASIC statement, a line of characters or a special function with one single keystroke.

The meaning of each Function key is defined at power up as their default functions. However, users are allowed to redefine the meaning of these function keys with the BASIC command—KEY.

For details of the default Function key configurations and how to redefine them, please refer to later Chapters and to the Appendix.

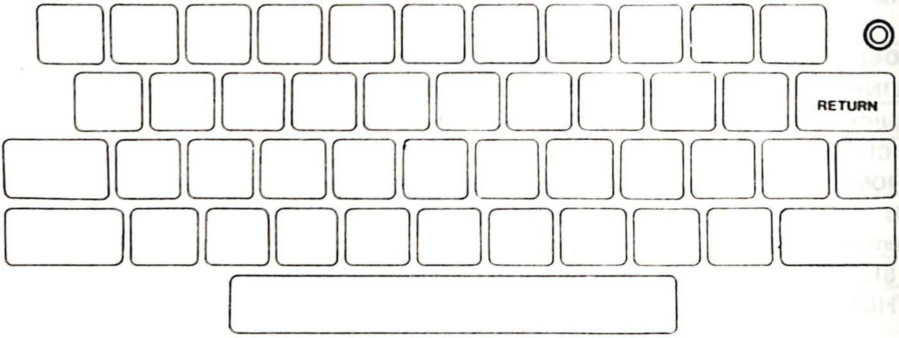
RESET (LASER 500/700 only)

Pressing the **RESET** key performs a very special action on the whole system. It does not generate any key code but instead, it forces any program in execution to stop. Usually you will be returned to BASIC', as shown by the prompt message "READY".

Normally you should not touch this key except when you get locked up in your program which goes to an endless loop somehow.

THE PRINT COMMAND

You instruct your computer using the language of BASIC. The computer can obey at once or it can store the instructions and run them later as a program. Let us now instruct the computer to act at once. To do this, we need our first word from the BASIC language: PRINT



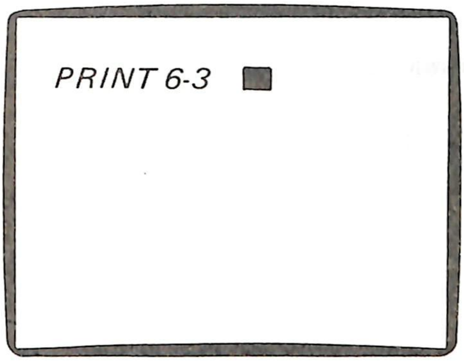
Note: In looking at the operation of PRINT as our first BASIC command, we will be using the RETURN key near the top right-hand corner of the keyboard. This key is used in BASIC to let the computer know when we have finished feeding in a command, so that it can go ahead and either carry out the command, or store it away in its memory to run later. Do not confuse this RETURN key with the BASIC command RETURN which we will meet later.

When using PRINT or any other statements you can type out each letter, e.g. P, R, I, N, T or you can just press the appropriate combination of keys, in this case CTRL and P (CTRL-P) to get the keyword. Try it out on your computer.

Note: (CTRL-P) means that you press the CTRL and P , keys simultaneously.

Whichever way you type in PRINT the computer knows that it has to print what follows on the screen. For simplicity, you can just type '?' to represent the command PRINT.

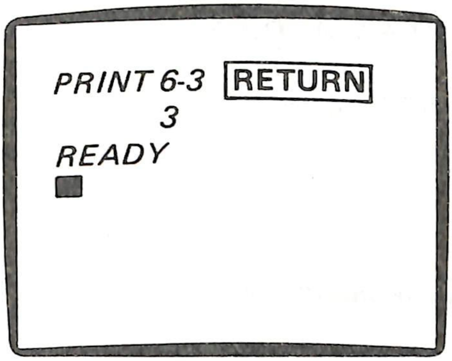
For example, type PRINT 6-3 and the screen should look like this.



```
PRINT 6-3 ■
```

Notice that when you press a key there is a (BEEP) sound. This tells you that the key has registered and is helpful in that you do not constantly have to check the screen.
Moreover, if you keep holding a key (or keys), the same character (or characters) is sent to the computer until you leave that key (or keys).

Now press RETURN and your screen should look like this.

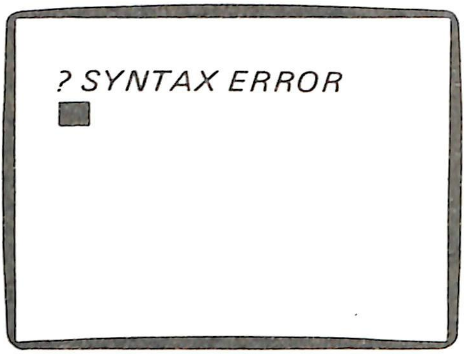


```
PRINT 6-3 RETURN
       3
READY
■
```

By pressing the RETURN you have told the computer that the message is completed and you want the line executed. Remember then to press RETURN after each completed message.

SYNTAX ERRORS

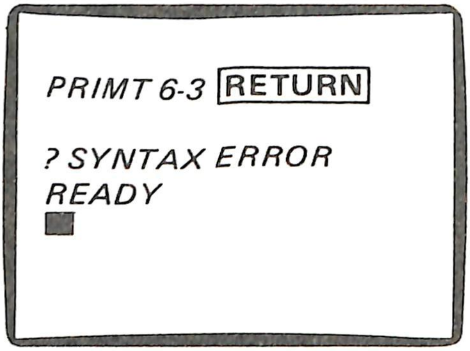
You may find the following appearing on your screen.



```
? SYNTAX ERROR
■
```

This means SYNTAX ERROR. A syntax error is usually due to incorrect punctuation or a typing error.

Suppose you type in PRIMT 6-3 and then press the RETURN key your screen will look like this.



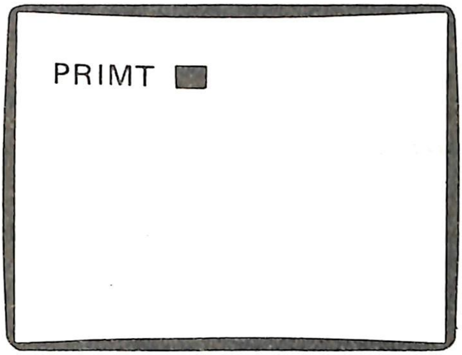
```
PRIMT 6-3 RETURN

? SYNTAX ERROR
READY
■
```

In addition to SYNTAX error there are a number of other errors that may occur. The various error types are listed in an appendix. These error messages tell you the reasons why your programs go wrong. If you are familiar with these messages, you can make use of them to correct your programs quickly and successfully.

EDITING

If you make a mistake while you are entering a program statement, you can use the CTRL key and the appropriate keys to move the CURSOR back to the wrong entry and make a correction.

For example:

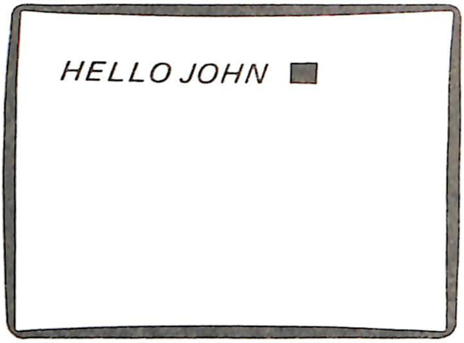


```
PRIMT ■
```

First move the CURSOR over 'M' by pressing ← , or BS , or CTRL ≤
You will observe that the letter M is changed to flashing between 'M' and the "Inversed M". Now press 'N' followed by 'T' to get 'PRINT'. The CURSOR automatically move to the right of 'T'.

More generally, you can make corrections to your program lines using any cursor movement keys, Insert key, Delete key, Delete to End of Line key, or the BASIC command DELETE. LASER 350 users might use the equivalent CONTROL key combinations.

Look at another example. If you have typed in a line, then the CURSOR is at the right side of the screen. Let us suppose you have made a mistake at the beginning of the line. You want to delete this line. Press ← (, or BS , or CTRL ≤) Keep pressed until the CURSOR has moved back to where you want it, in this case, the character 'H' . Then press the DEL , (or CTRL M) and keep pressed until the line is erased.



```
HELLO JOHN ■
```

The above operation could have been done much simpler by moving the CURSOR to the letter 'H' on the word "HELLO", then press DEL LINE (or CTRL 3#) for 'clear to End of Line'.)
Suppose that after having HELLO JOHN you decide to change the name. However this time suppose the CURSOR is on a lower line. Well all you do is to press ↑ (or CTRL ?/).

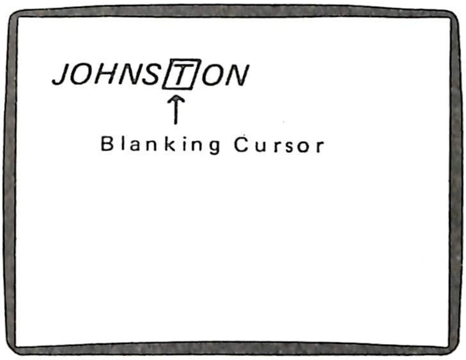
Each time you press ↑ (CTRL ?/) the CURSOR will move up one line. Once you have reached the line you wish to edit, you can just carry on the same way as described in the example above.

To familiarize yourself with editing you need to experiment with it. Here, as elsewhere, the old adage holds true: Practice Makes Perfect.

INSERT

This allows you to insert characters starting at the position the CURSOR is in without changing what is already there. For example: you wish to insert S in JOHNTON between the N and the T.

Well you move the CURSOR to the T. Press INS and the letters 'TON' would be shifted by 1 position to the right leaving one space. Then type S. Your display should now look like this:



JOHNS$\boxed{T}$ON

Be sure to press RETURN after you finish editing. This will update the current line where the CURSOR is located. This is particularly important with numbered program lines. If you forget to do so, the original line is still kept in the program.

CLS

If you want to clear the whole text screen, press SHIFT CLS HOME (or CTRL 5).

This will clear the screen, 'HOME' the cursor to the top left-hand position, but it will not clear the memory. The program will only be wiped from memory if you press NEW (or CTRL N). It will also be wiped out if you disconnect the power from the computer.

A LOOK AHEAD

At this stage you are probably very eager to jump ahead and see what your computer is capable of. So using your newly acquired ability try typing in these programs.

Be careful to type in everything. Do not worry about understanding the commands at the moment.

1) To get all the characters on the screen type this

Example

```
10      TEXT 40 : CLS  RETURN
20      FOR I = 33 to 159 STEP 6   RETURN
30      FOR J = Ø TO 5 RETURN
40      PRINT TAB (7*J); CHR$(I+J) ;   RETURN
50      NEXT J  RETURN
60      NEXT I  RETURN
70      GOTO 70  RETURN
RUN  RETURN
```

To stop this program, press CTRL C

2) To see some of the color possibilities try this one.

Example:

```
1Ø   TEXT 4Ø : GR 1 : D = Ø    RETURN
2Ø   FOR C = Ø TO 15           RETURN
3Ø   COLOR C, Ø, Ø             RETURN
4Ø   FOR Y = Ø TO 11           RETURN
5Ø   FOR X = Ø TO 159          RETURN
6Ø   SET (X, Y + D)            RETURN
7Ø   NEXT X                    RETURN
8Ø   NEXT Y                    RETURN
9Ø   D = D + 12                RETURN
1ØØ  NEXT C                    RETURN
11Ø  GOTO 11Ø                  RETURN
RUN  RETURN
```

To stop this program, press CTRL C

CHAPTER 3

USING YOUR COLOR COMPUTER AS A SIMPLE CALCULATOR

- Simple Instructions
- Order of Numeric Operations
- Brackets.

SIMPLE INSTRUCTIONS

To use the computer as a calculator simply type PRINT followed by the problem and then press RETURN . Your computer, of course, cannot only add, using +, but it can also subtract using –, multiply using*, divide using/and raise one number to the power of another using ↑ , +, –, *, /, ↑ are called operations, and they operate on numbers called operands.

Example:

```
PRINT 3 ↑ 2          RETURN
```

and the answer 9 will appear.

Module arithmetic is denoted by MOD.
It gives the integer value that is the remainder of an integer division.

For instance: 10.4 MOD 4 = 2
25.68 MOD 6.99 = 5

Example:

```
PRINT                1Ø.4 MOD 4
  2
```

ORDER OF NUMERIC OPERATIONS

When operations are combined, care must be taken to note the order in which the computer carries out the operations. The order is as follows:

1) Minus sign – used to indicate negative numbers.
2) Exponentiation starting at the left and moving right.
3) Multiplication and division (which are given the same order of precedence). Here too the computer moves from left to right.
4) Subtraction and addition moving from left to right.

A)

Example:

```
PRINT 3 ↑ 2 ↑ 2 ↑ 2        RETURN
6561
```

This is done by squaring 3 to get 9. Then squaring 9 to get 81, and then squaring 81 to get 6561.

B)

Example:

```
PRINT 6 * 2 + 3    RETURN
```

Here the computer first multiplies 6 X 2 and then adds 3.

C) .

Example:

```
PRINT 6 + 3 * 4 + 6/3     RETURN
2Ø
```

First the computer carries out the multiplication and division and then adds to give 6 + 12 + 2.

BRACKETS

All operations within brackets will be carried out first before the other operations.

Example:

```
PRINT 18/(3 + 3 )         RETURN
3
```

Where brackets are placed within brackets the innermost brackets are calculated first.

Example:

```
PRINT 2Ø/(1 + (3 ↑ 2))          RETURN
2
```

CHAPTER 4

CONSTANTS AND VARIABLES

- Constants
- Variables
- LET
- Semi-colons and commas
- Colons

CONSTANTS

In the previous chapters we have been using constants. A constant is, of course, something which does not change and can be either positive or negative. The number 6.32 is a constant. Moving onto 6.33 just gives a different constant.

The range of a number in the computer is $-10^{38} \leqslant X \leqslant 10^{38}$
The lowest positive number is 10^{-38}.

VARIABLES

A variable, as you might surmise, is something which changes. In Y = X + 3 both X and Y are variables as they have many possible values. Variable names may be any length and up to 40 characters are significant. The first character must be a letter. For example, A, AB are valid names of variable.

A variable name cannot be or include any of the command words like LET or PRINT. There are totally 3 types of variables namely:

a) real : – can have decimal value, e.g. A = 3.5, or A = 1E3.
b) integer : – cannot have decimal value, e.g. A% = 3.
c) string : –contain a string, e.g. A$ = "ABCD".

LET

The command **LET** can be used to assign a value to a variable. If a variable is not assigned a value it is assumed to be equal to zero. The variable will keep its assigned value until another **LET, READ** or an **INPUT** command is used to change the value.

Example:

```
LET A = 7      RETURN
LET B = 9      RETURN
PRINT A + B    RETURN
16
```

In BASIC the = sign does not mean the same as it usually does. Here it tells the computer to give the variable on the left hand side the same value as the right hand side.

The left hand side of the statement must always be a variable. Have a look at the next example.

Example:

```
LET A = 2        RETURN
LET B = 3        RETURN
LET C = 2Ø       RETURN
LET A = 2 + A       RETURN
LET D = 3 + D        RETURN
PRINT A; B; C; D   RETURN
4    3    2Ø    3
```

In the fourth line we see that A is assigned a new value of 2 plus the old value of A, which was also 2, giving a total of 4.

In the fifth line we see that D is given the value 3 + D, as zero is the value given to any variable without an assigned value. However, it is a good practice to predefine all variables.

Note that in your computer it is not strictly necessary to use **LET** to assign a value to variable, and *A = 7* will carry out the same function as *LET A = 7.*

SEMI-COLONS, COMMAS

If more than one item is included in a **PRINT** statement the items should be separated by either a (,) or (;). Note the use of the semi-colon in the **PRINT** statement above. This causes the results to be printed immediately after each other with a space is left for the sign of that number. When we use the semi-colon with string there is no space. Typing one or more spaces between expressions has the same effect as typing a semi-colon.

A comma causes the result to be printed as follows. Think of the lines on screen divided into print zones of 14 spaces each. A comma causes the next value to be printed at the beginning of the next zone.

COLONS

If you have more than one statement on a line you must separate them by using colons.

Example:

```
10 FOR I = 1 TO 5 : PRINT I;: NEXT     RETURN
RUN   RETURN
1   2   3   4   5
```

LISTS OF PRINT STATEMENTS

Example:

```
10   PRINT 4    RETURN
20   PRINT 6    RETURN
30   PRINT 8    RETURN
RUN       RETURN
```

Your screen will show

Example:

```
4
6
8
```

SEMI COLONS AT THE END OF PRINT STATEMENTS

Example:

```
10 PRINT 4;    RETURN
20 PRINT 5;    RETURN
30 PRINT 6;    RETURN
RUN       RETURN
```

Your screen will show

Example:

```
4   5   6
```

In general, a semi-colon placed at the end of a **PRINT** statement tells the computer not to go to a new line after printing.

CHAPTER 5

INTRODUCTION TO BASIC PROGRAMMING

- Immediate mode/Deferred execution
- REM
- INPUT
- NEW
- RUN
- LIST
- Pause in listing
- Delete a line

PROGRAMMING

OK, so now let's try some simple programming. In the last few chapters we have been dealing with "immediate execution" – with the computer obeying immediately. We now want the computer to store statements so that they can be executed later on – "deferred execution".

Have a look at this program.

Example:

```
1Ø REM RAISE TO THE POWER OF 3    RETURN
2Ø INPUT A   RETURN
3Ø PRINT A; A ↑ 3   RETURN
```

Notice that each line begins with a number. These numbers tell the computer not to obey immediately but to store the lines away. The line number governs the order in which the line will appear on the screen. It is useful to write the numbers in tens as new lines can be later fitted into any part of the program by giving them a value of say 15 or 25. The range of possible line numbers is from Ø to 65529.

REM

The REM in line 1Ø is simply there to remind you later on of the purpose of the program. The computer will ignore any line which starts with REM. However REM lines use memory space so if you are short of space you can delete REM lines.

INPUT

The INPUT in line 2Ø asks you to assign a value to the variable A. When you run this program, a question mark "?" will display. The computer will wait until you type in a value and give this value to the variable A.

NEW

So how do we feed the program into the computer? Well first feed in the **NEW** command and press RETURN . This will wipe out any old programs and variables. Remember the **NEW** command clears the memory of the computer.

Now type

Example:

```
1Ø REM RAISE TO THE POWER OF 3   RETURN
2Ø INPUT A   RETURN
3Ø PRINT A; A ↑ 3   RETURN
```

You can now run the program by typing **RUN** and press RETURN . The sign '?' will now appear under **RUN** . This is the result of the INPUT statement and the computer is now waiting for you to give a value to the variable A. This value should be typed next to '?'.

Let's type 2 and RETURN

The screen will look like this.

Example:

```
1Ø REM RAISE TO THE POWER OF 3.     RETURN
2Ø INPUT A     RETURN
3Ø PRINT A; A ↑ 3    RETURN
RUN    RETURN
?  2   RETURN
2  8
```

RUN

By typing **RUN** and pressing RETURN the whole of your stored program will be executed, starting at the line with the smallest line number. If however you press **RUN** and then a line number before RETURN , the program will be executed starting from that line.

Have a look at this program

Example:

```
1Ø INPUT A, B      RETURN
2Ø PRINT A + B     RETURN
RUN      RETURN
```

What will happen here is, you will get a '?' to ask for the values of A and B. You have to type in the value of A, followed by a comma, and then the value of B. So carry on with the program above ....

Example:

```
1Ø INPUT A, B       RETURN
2Ø PRINT A + B       RETURN
RUN     RETURN
? 3, 6      RETURN
9
```

The INPUT command requires you to type in the numbers in exactly the same sequence as listed on the INPUT line, with each number seperated by a comma.

LIST

If you want the whole program to be displayed in an ascending line number order then just type **LIST** and press RETURN .

Example:

```
1Ø INPUT A       RETURN
2Ø INPUT B        RETURN
3Ø PRINT A; B; C; A + B + C     RETURN
25 INPUT C  RETURN
```

Example:

```
LIST    RETURN
1Ø INPUT A
2Ø INPUT B
25 INPUT C
3Ø PRINT A; B; C; A + B + C
```

If you only want one line to be displayed then type

Example:

```
LIST (Line number)      RETURN
```

Example:

```
LIST 2Ø  RETURN
2Ø INPUT B
```

To list part of a program say, lines 2Ø–3Ø, type

Example:

```
LIST 2Ø–3Ø   RETURN
2Ø INPUT B
25 INPUT C
3Ø PRINT A; B; C; A + B + C
```

If you type *LIST – 3Ø* you will get the program listed up to line 3Ø from the start.

If you type *LIST 3Ø – you* will get the program listed from line 3Ø to the end.

PAUSE IN LISTING

If you have a very long program you might wish to have a look at a particular line while it is being listed. To do this just press the SPACE bar when you wish the listing program to stop.

Press the same bar again to continue.

DELETE A LINE

To get rid of any program line just type the line number and press RETURN .

Your computer also allows you to perform the above function in another way.

Just type:

```
DELETE 4Ø
```

and the program line will be deleted.

Furthermore, to delete all program lines from 4Ø to 1ØØ inclusively, type:

```
DELETE 4Ø–1ØØ
```

CHAPTER 6

COMPUTER PROGRAMMING REVISITED

- GOTO
- PAUSE
- CONT
- STOP
- END
- CLEAR

Here are some more commands to help you write more interesting programs.

GOTO

This command tells the computer to go backwards or forwards to the line number following the **GOTO** statement and then to carry on executing the program from that line number.

Example:

```
1Ø INPUT A    RETURN
2Ø PRINT A, A ↑ 3    RETURN
3Ø GOTO 1Ø    RETURN
   RUN    RETURN
   ?
```

If you give the value say 2 to A the computer will return the results 2 and 8. However a question mark will again appear on the screen asking you to give A another value. This procedure is the result of the GOTO statement telling the computer not to end at line 3Ø but to go back to line 1Ø and start again.

PAUSE

When you get tired of putting in different values of A you can press CTRL C . The computer will stop the program at the end of the executing statement and will output on to the screen a message:

BREAK IN 10

It should be noted that the word BREAK is not a command. You can break any continuous BASIC program by pressing CTRL C .

Pressing the RESET key on LASER 500/700 served similar purpose. It will stop any executing program and return you to BASIC command level.

CONT

If however, after stopping the program execution you feel there are still some values of A you would like to try you can type CONT, and the computer will start to execute the program once more.

CONT causes a program to continue after stopping in response to CTRL C or STOP, without resetting any variables. It might be possible to continue execution after an error.

STOP

A useful statement in programming is STOP. This causes the program to stop at the line printed after the STOP statment and can help you to examine the results of the variables at various stages in the program. It is also extremely useful when it comes to locating mistakes (debugging). A liberal supply of STOP statements throughout a program is therefore a good idea, until you are sure that it is working properly.

You can restart the program by typing CONT. The program will carry on from the next line after the STOP.

END

The **END** statement is used to terminate execution. But unlike the **STOP** statement, execution cannot be continued after an END statement.

Example:

```
10 INPUT A    RETURN
20 IF A > Ø THEN PRINT "A IS POSITIVE":
   END  RETURN
30 IF A < Ø THEN PRINT "A IS NEGATIVE":
   END  RETURN
40 PRINT "A IS ZERO"     RETURN
5Ø END  RETURN
```

NOTICE the **STOP** statement will give you the line number when it is executed. This will not happen with the END statement.

CLEAR

The CLEAR statement sets all numeric variables to zero, all strings to null, and optionally sets the end of user memory and the amount of stack space.

The format of `CLEAR, M, N`

in which M, N are expressions representing numbers.

The value of M sets the highest memory for use by the BASIC program. The value of N set aside stack space for BASIC. Default is 512 bytes of memory.

For LASER 350 without any RAM expansion, the user can increase his available RAM by typing in:

```
CLEAR    , & HB7FD
```

However, no graphic mode can be used after that.

CHAPTER 7

NUMERIC FUNCTIONS

- What is a Function
- ABS
- SGN
- SQR
- LOG
- EXP
- INT
- FIX
- RND
- SIN
- COS
- TAN
- ATN
- User defined Functions

WHAT IS A FUNCTION?

A function is a 'law' which when applied to a certain value will give a new value. We call the first value the argument and the new value the result.

SQR is the square root function. So if we type

Example:

```
PRINT SQR (9)    RETURN
```

we will get the answer 3.

In this example 9 is the argument, SQR is the function and 3 is the result.

Below we give a list of the numeric functions and a brief explanation. Any function which we consider to be new to the reader will be explained in more detail afterwards. The functions will appear later on in programs so we don't give a sample program for each one here.

A LIST OF NUMERIC FUNCTIONS

Function	What it does
ABS(X)	Returns the absolute (positive) value of X
SGN(X)	Returns the sign of the argument X negative returns – 1 X positive returns + 1 X zero returns Ø
SQR(X)	Returns the square root of X. X cannot be negative.
LOG(X)	Gives the natural logarithm of X, i.e., the logarithm to the base e (=2.71828). The value of the argument must be greater than zero.
EXP(X)	Gives you the value e^X – i.e., the natural antilogrithm of X.
FIX(X)	Returns truncated integer part of X.
RND(X)	Gives a random number between Ø and 1. The value of X affects the next random number generated.

SIN(X)
COS(X)
TAN(X)
	The argument of the trigonometrical functions is taken to be in radians (1 radian = 360/2π = 57.296 degrees). The range of X is –9999999<X<9999999.

ATN(X)	This gives the result of ARC TANGENT in radians.

A FURTHER LOOK AT ABS SGN INT RND

ABS(X)

This gives the absolute (positive) value of the argument. So ABS(—7)=7.

Example:

```
PRINT ABS (7 – 2 * 4)    RETURN
1
```

SGN(X)

This function will give the value of +1 if X is positive, Ø if X is zero, and —1 if X is negative. So *SGN (4.3) = 1;SGN (Ø) = Ø;SGN(—.276) =—1.*

Example:

```
A = –6    RETURN
PRINT SGN (A); SGN (A – A)    RETURN
–1    Ø    RETURN
```

INT(X)

This converts arguments which are not whole into the largest whole number below the argument. So *INT (5.9) = 5;* also *INT (—5.9) = —6.* Note that with negative arguments, the absolute value of the result returned by INT will be greater than that of the argument.

Example:

```
PRINT INT (-6.7)    RETURN
–7
```

RND(X)

This will produce a random number between Ø and 1. The same sequence of random numbers is generated each time the program is RUN unless the RANDOMIZE command is executed at the program start.

X< 0 always restarts the same sequence foı any X.

X > 0 or X omitted generates the next random number in the same sequence.

X = 0 repeats the last number generated.

Example:

```
10 FOR I = 1 TO 5
20 PRINT INT (RND * 100);
30 NEXT
RUN
24    30    31    51    5
READY
RUN
24    30    31    51    5
READY
```

The number displayed are always the same.

RANDOMIZE

RANDOMIZE is not by itself a numeric function. Its purpose is to reseed the random number generator, which you may look upon as a small machine that throws out random numbers as the RND(X) function informs it to do so.

If the random number generator is not reseeded, the RND function returns the same sequence of random numbers each time the program is RUN.

To change the sequence of random numbers every time the program is RUN, place a RANDOMIZE statement at the beginning of the program.

Example:

```
5 RANDOMIZE
10 FOR I = 1 TO 5
20 PRINT INT (RND  * 100);
30 NEXT
RUN
   21    38    43    13    80
READY
RUN
   87    55    43    57    34
READY
```

Different results are now obtained each time.

USER— DEFINED FUNCTIONS

The BASIC in your computer is equipped with powerful User—Defined Functions capability. An example of User—Defined Function is:

```
    10 DEF FNAB (X, Y) = X + Y
```

In this BASIC statement line, the function name AB, with parameters or arguments X and Y, is defined as the sum of X and Y.

After the function is defined, it can be called as a variable by its name. For instance, another program line calling the above defined function might be:

```
    20 T = FN AB (I, J)
```

This line would assign the value of variables I plug J to the variable T.

CHAPTER 8

STRING

- String variables
- String functions
- LEN
- STR$
- VAL
- LEFT$
- RIGHT$
- MID$
- ASC
- CHR$
- OCT$
- HEX$
- STRING$
- INKEY$
- INSTR
- INPUT A$
- LINE INPUT A$
- INPUT$
- STRING COMPARISONS

STRINGS

Note: We assume that you are now familiar with the use of the RETURN key so we will not keep reminding you of it.

A string is any combination of CHARACTERS that is treated as a unit.

String constants must be enclosed in inverted COMMAS.

Example:

```
"HELP"
```

When using the PRINT statement a semi-colon between strings will not cause a space to appear between the results. They will appear immediately next to each other.

STRING VARIABLES

Any letter of the alphabet or letter followed by a number digit can be used as a string variable but must be followed by a $ sign. The computer accepts these characters as the variable name.

Example:

```
A$ = "ONE DOZEN EGGS"
```

You can add strings to each other. The is called concatenation. You cannot subtract, divide or multiply strings.

Example:

```
1Ø A$ = "I   A"
2Ø B$ = "M 15 YEA"
3Ø C$ = "RS OLD"
4Ø PRINT A$ + B$ + C$
   RUN
   I AM 15 YEARS OLD
```

Notice the spacing of the string characters here.

STRING FUNCTIONS

We can also use functions to act on strings. Have a look at the following:

LEN

This function works out the length of the string argument, which must be in brackets. So if you type *PRINT LEN ("JOHN")* the computer will return the result 4. This is telling you that there are 4 characters irт the string "JOHN". Blank spaces have the value of a character. Thus if you put in spaces 'J O H N' it comes out as 7 characters.

STR$

The **STR$** function changes a number argument into a string. Let us take a look at the following example and see how it works.

Example:

```
A$ = STR$ (73)
```

This is the same as saying

Example:

```
A$ = "73"
```

Here is an example program

Example:

```
1Ø A$ = STR$ (7 * 3)
2Ø B$ = A$ + "BIG"
3Ø PRINT B$
     RUN
       21BIG
```

VAL

VAL works like **STR$** but in reverse. It changes a string argument into a number. It only works on numbers not on operators or other characters.

Look at the following short program

Example:

```
1Ø A$ = "33"
2Ø B$ = "2Ø"
3Ø C = VAL (A$ + B$)
4Ø PRINT C; C + 1ØØ
RUN
332Ø          342Ø
```

SUBSTRINGS

It is also possible to get substrings of strings. A substring is as you might guess a part of a string. For example: "ABC" is a substring of "ABCDE".

LEFT$(A$, N)

This will return the substring from the leftmost of string A$ – the first character – to the Nth character.

Example:

```
10 A$ = "ABCDE"
20 B$ = LEFT$ (A$ + "FGH", 6)
30 PRINT B$
    RUN
     ABCDEF
```

RIGHT$(A$, N)

This will return a substring as in the above example but starting from the Nth character from the end and running to the last one – the right most character in the string A$.

Example:

```
10 A$ = "WHY"
20 B$ = RIGHT$ (A$ + "ME", 4)
30 PRINT B$
     RUN
      HYME
```

MID$(A$, M, N)

This function returns a substring of the string A$ starting from the Mth character with a length of N characters.

Example:

```
10 A$ = "ABCDEFGH"
20 B$ = MID$ (A$, 2, 3)
30 PRINT B$
    RUN
    BCD
```

ASC(A$)

The **ASC** statement which is written as **ASC (A$)** where A$ is a variable string expression, will return the ASCII code (in decimal) for the FIRST character of the specified string. Brackets must enclose the string specified. Refer to the appendix for the ASCII code. For example the ASCII decimal value of "X" is 88. If *A$ = "XAB", then ASC (A$) = 88.*

Example:

```
1Ø X = ASC ("ROY")
2Ø PRINT X
   RUN
    82
```

CHR$(N)

This statement works the opposite way around to the ASC statement. The CHR$ statement will return the string character which corresponds to the given ASCII code. The argument may be any number from 0 to 255 or any variable expression with a value within that range. Brackets must be put around the argument.

Example:

```
3Ø PRINT CHR$(68)
   RUN
    D
```

OCT$(X)

This will return a string which represent the octal (Base 8) value of the decimal argument X. X is rounded to an integer before OCT$ (X) is evaluated.

Example:

```
1Ø PRINT OCT$ (24)
RUN
3Ø
```

HEX$(X)

This function, similar to OCT$(X), will return a string which represents the hexadecimal value of the decimal argument. X is also rounded to integer before HEX$(X) is evaluated.

Example:

```
10 INPUT X
20 A$ = HEX$(X)
30 PRINT X "DECIMAL IS"; A$; "HEXADECIMAL"
RUN
? 32 (user typed in 32)
32 DECIMAL IS 20 HEXADECIMAL
```

STRING$(I, J) OR STRING$(I, X$)

The above two formats will return a string of length I whose characters all have ASCII code J or the first character of X$.

Example:

```
10 X$ = STRING$(10, 45)
20 PRINT X$ "MONTHLY REPORT" X$
RUN
  ----------MONTHLY REPORT----------
```

Where ASCII for 45 is the symbol "–".

INKEY$

INKEY$ returns either a one-character string containing a character read from the keyboard or a null string (i.e., an empty string, with no characters) if no key has been pressed at the keyboard. All characters are passed through to the program except for CTRL C, which terminates the program.

Example:

```
10 A$ = INKEY$
20 PRINT A$;
30 GOTO 10
```

To stop this program, press CTRL C

INSTR (I, X$, Y$)

This string function searches for the first occurrence of string Y$ in X$ and returns the position at which the match is found. I represents an optional offset position for starting the search, where 1 ⩽ I ⩽ 255.

If I > LEN(X$) or if X$ is null or if Y$ cannot be found, INSTR(I, X$, Y$) returns a Ø value.

If Y$ is null, INSTR returns I or 1.

Example:

```
1Ø X$ = "ABCDEB"
2Ø Y$ = "B"
3Ø PRINT INSTR (X$, Y$) ; INSTR (4, X$, Y$)
RUN
2 6
```

INPUT A$

Your have come across INPUT a numerical variable in eariler chapters. In fact, your computer allows for assiging a string which you typed in while your program is running, to a string variable.

The above statement, INPUT A$ when executed, will print a '?' sign to ask you to type in characters from the keyboard, until you terminate your input by pressing RETURN.

You should not embed ' , ' (comma) in the string you enter. Otherwise, a 'Redo from Start' error message will be issued. Then the computer prints another '?' on the next line to let you re-enter your string.

Example:

```
1Ø INPUT A$
2Ø PRINT A$
RUN
? ABC, DEF RETURN (you typed in)
? Redo from Start
? ABCDEF RETURN  (you typed in)
ABCDEF
```

LINE INPUT A$

The statement works similar to INPUT A$, in which a line of characters that you typed from the keyboard is assigned to the string variable A$. It differs from merely INPUT A$ by not printing a '?' sign, and it allows you to embed ' , ' (comma) in your string input.

Example:

```
1Ø LINE INPUT A$
2Ø PRINT A$
RUN
ABCDEF, GHIJK [RETURN] (you typed in)
ABCDEF, GHIJK
```

INPUT$

The function INPUT$(X) is different from the command INPUT A$ mentioned in the previous chapters.

This function returns a string of X characters. The characters input from keyboard are not echoed to the screen.

Example:

```
1ØØ PRINT "TYPE P TO PROCEED OR S TO STOP"
11Ø X$ = INPUT $ (1)
12Ø IF X$ = "P" THEN 5ØØ
13Ø IF X$ = "S" THEN 7ØØ ELSE 1ØØ
```

STRING COMPARISONS

Relational operators can be applied to string expressions to compare the strings for equality or alphabetic precedence. As far as equality is concerned all the characters (and any blanks) must be identical and in the same order.

Example:

```
1Ø A$ = "AA"
2Ø B$ = "BA"
3Ø IF A$ = B$ then PRINT 2Ø
4Ø IF A$ < B$ then PRINT 3Ø
5Ø IF A$ > B$ then PRINT 4Ø
    RUN
    3Ø
```

The comparisons are done by taking the ASCII value of the string characters from the table in the appendix and then comparing these values. The table gives us the value for 'A' as 65 and 'B' as 66. The program above is therefore asking for confirmation that 65 is less than 66.

If the first two CHARACTERS of a string are equal the computer will search for the third CHARACTER and do the comparison on this.

Example:

```
1Ø A$ = "ABC"
2Ø B$ = "ABD"
3Ø IF B$ > A$ then PRINT 4Ø
   RUN
   4Ø
```

Here the critical comparison is between the characters C and D. The ASCII table value of C is 67 and the table value of D is 68. B$ is therefore greater than A$.

CHAPTER 9

CONDITIONS

- IF . . . THEN . . . ELSE
- Conditional Branching
- Logical Operators
- TRUTH TABLES
- ON. . . GOTO/ON . . . GOSUB
- ON ERROR GOTO
- RESUME

CHAPTER 9

CONDITIONS

- IF . . . THEN . . . ELSE
- Conditional Branching
- Logical Operators
- TRUTH TABLES
- ON. . . GOTO/ON . . . GOSUB
- ON ERROR GOTO
- RESUME

IF . . . THEN . . . ELSE

As we make our way through BASIC, we find that we gain more control over the computer, that is, we are able to do more with the computer. In this chapter we are going to take a look at the **"IF . . . THEN . . . ELSE"** statement. This is, perhaps, one of the two most important programming concepts in BASIC. The other one is **"FOR . . . NEXT"**. We will look at this in the following chapter.

Let us look at this example.

Example:

```
6Ø IF A$ > B$ THEN PRINT A$ ELSE PRINT B$
```

This tells the computer that if the expression A$ is greater than B$ to carry out the statement *PRINT A$;* otherwise it should carry out the statement *PRINT B$.*

CONDITIONAL BRANCHING

In general terms, the IF . . . THEN . . . statement is used for conditional branching. It uses the general form "IF (condition) THEN (action clause)." A condition is made up of: an expression, a relation and an expression.

Any BASIC expressions may be used but both expressions must be of the same type, that is either both numeric or both string expressions.

Relations or comparisons used in the IF . . . THEN statement are the following:

= Equal to
<= Less than or equal to
<> Not equal to
>= Greater than or equal to
< Less than
> Greater than

Here are some more examples of how we can use conditionals.

IF . . . THEN A = B
IF . . . THEN GOTO
IF . . . THEN GOSUB
IF . . . THEN PRINT
IF . . . THEN INPUT

IF . . . THEN . . . ELSE

As we make our way through BASIC, we find that we gain more control over the computer, that is, we are able to do more with the computer. In this chapter we are going to take a look at the "**IF . . . THEN . . . ELSE**" statement. This is, perhaps, one of the two most important programming concepts in BASIC. The other one is "**FOR . . . NEXT**". We will look at this in the following chapter.

Let us look at this example.

Example:

```
6Ø IF A$ >B$ THEN PRINT A$ ELSE PRINT B$
```

This tells the computer that if the expression A$ is greater than B$ to carry out the statement *PRINT A$;* otherwise it should carry out the statement *PRINT B$.*

CONDITIONAL BRANCHING

In general terms, the IF . . . THEN . . . statement is used for conditional branching. It uses the general form "IF (condition) THEN (action clause)." A condition is made up of: an expression, a relation and an expression.

Any BASIC expressions may be used but both expressions must be of the same type, that is either both numeric or both string expressions.

Relations or comparisons used in the IF . . . THEN statement are the following:

= Equal to
<= Less than or equal to
<> Not equal to
>= Greater than or equal to
< Less than
> Greater than

Here are some more examples of how we can use conditionals.

IF . . . THEN A = B
IF . . . THEN GOTO
IF . . . THEN GOSUB
IF . . . THEN PRINT
IF . . . THEN INPUT

Example:

```
3Ø IF X > 25 THEN 6Ø
```

Here if the condition X > 25 is true, the computer is told to jump to line 60 (Note: the GOTO is optional after THEN).

If the condition is not true, that is, if X is not greater than 25 then the computer simply carries on with the normal line number order in the program. Notice that it is not necessary to use the ELSE part of the command here as this is optional.

Example:

```
1Ø INPUT A, B
2Ø IF A > B THEN 5Ø
3Ø IF A < B THEN 6Ø
4Ø IF A = B THEN 7Ø
5Ø PRINT A; "IS GREATER THAN"; B: END
6Ø PRINT A; "IS LESS THAN"; B: END
7Ø PRINTA; "IS EQUAL TO ";B
8Ø END
   RUN
   ?    7, 3
   7 IS GREATER THAN 3
```

Example:

```
4Ø IF P = 6 THEN PRINT "TRUE" ELSE PRINT
"FALSE"
```

In this example if P = 6 the computer will print TRUE. Any other value will produce a FALSE. In either case the computer will carry on to the next line.

It is possible for more than one statement to follow the THEN or ELSE command.

Example:

```
5Ø IF A = 5 THEN PRINT "TRUE" : S = S – 3:
GO TO 9Ø ELSE PRINT "FALSE": K= K + 8
```

So if A equals 5 the computer will print TRUE, substract 3 from the variable S and go to line 9Ø. If A does not equal 5 the computer will print FALSE, add 8 to the variable K and then carry on with the next normal line.

LOGICAL OPERATORS

Logical operators are used in IF . . . THEN . . . ELSE and such statements where a condition is used to determine subsequent operations within the user program. The logical operators are: AND, OR, NOT, XOR, IMP, EQV.

For purposes of this discussion A and B are relational expressions having only TRUE and FALSE. Logical operations are performed after arithmetical and relational operations.

OPERATOR	EXAMPLE	MEANING
NOT	NOT A	If A is true, NOT A is false.
AND	A AND B	A AND B has the value true only if A and B are both true. A AND B has the value false if either A or B is false.
OR	A OR B	A OR B has the value true if either A or B or both are true. It has the value false if both are false.
XOR	A XOR B	A XOR B has the value true only if A and B are different. A XOR B has the value false if A and B are the same.
IMP	A IMP B	A IMP B has the value false, only if A is true, and B is false. Otherwise, A IMP B has the value true.
EQV	A EQV B	A EQV B has the value true only if A, B are the same. A EQV B has the value false if A, B are different.

TRUTH TABLES

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

TRUTH TABLE FOR "NOT" FUNCTION

A	NOT A
T	F
F	T

TRUTH TABLE FOR "AND" FUNCTION

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

TRUTH TABLE FOR "OR" FUNCTION

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

TRUTH TABLE FOR "XOR" FUNCTION

A	B	A XOR B
T	T	F
T	F	T
F	T	T
F	F	F

TRUTH TABLE FOR "IMP" FUNCTION

A	B	A IMP B
T	T	T
T	F	F
F	T	T
F	F	T

TRUTH TABLE FOR "EQV" FUNCTION

A	B	A EQV B
T	T	T
T	F	F
F	T	F
F	F	T

Note that T = TRUE and F = FALSE.

Example:

```
10 INPUT A, B, C
20 IF A = B AND B = C THEN PRINT "A = B = C"
30 IF (NOT A = B) OR (NOT B = C) THEN 5Ø
40 END
5Ø PRINT "A = B = C IS FALSE"
6Ø END
   RUN
   ?  1Ø, 5, 7
   A = B = C IS FALSE
```

Moreover AND, OR, NOT can be used to manipulate numerical values. These operations are based on binary numbers with 1 and Ø representing TRUE and FALSE respectively. For example:

i) NOT 1 = –2 [1=binary ØØØØØØØ1 and –2=binary 1111111Ø, so it just changes the 1 to Ø and Ø to 1. In other words, TRUE (1) changed to FALSE (Ø) and FALSE (Ø) is changed to TRUE (1).]

ii) 6 OR 13 = 15 [6=binary ØØØØØ11Ø and 13=binary ØØØØ11Ø1, so with reference to the OR truth table, 6 OR 13 = 15 = binary ØØØØ1111]

iii) 6 AND 13 = 4 [6=binary ØØØØØ11Ø and 13=binary ØØØØ11Ø1, so with reference to the AND truth table, 6 AND 13=4 binary ØØØØØ1ØØ]

ON . . . GOTO /ON . . . GOSUB

The ON . . . GOTO or ON . . . GOSUB statement allows the program to direct its flow depending on the value of an expression.

For example:

```
1ØØ   ON L–1 GOTO 15Ø, 3ØØ, 32Ø
```

OR

```
1ØØ   ON L–1 GOSUB 15Ø, 3ØØ, 32Ø
```

The expression (L–1) is evalvated. Its value determines which line number in the list will be used for branching or subroutine calling. If the value is 3, for instance, the third line number (320) would be the destination.

If the value of (L–1) evaluated is Ø or greater than the number of listed line numbers, the BASIC program will continue with the next executable statement.

ON ERROR GOTO

Using this statement avoid halting the program when an error is encountered. The error is being trapped, by directing the program to a routine (error handling routine).

Example:

```
10   ON ERROR GOTO 1ØØØ
```

Executing the above line will cause subsequent errors, including immediate mode errors (such as SYNTAX ERRORS) to jump to the specified error handling routine at 1ØØØ.

The ON ERROR GOTO statement may be located anywhere in the program, but it is good practice to have it as early as possible, as this statement must be executed before the occurance of an error to avoid program interruption.

RESUME

The RESUME statement is to continue a program exectuion that has been halted due to an error. It is mainly used at the end of an error handling routine, and it causes the program to restart execution in one of these forms:

1. RESUME OR RESUME Ø — Program execution resumes at the line which caused the error

2. RESUME NEXT — Program execution resumes at the line immediately following the one which caused the error

3. RESUME line number — Program resumes at the specified line number

FOR . . .

Often we
process
ments
For exa
we do
like this
Example

From
divide
stateme
on

You
each
statem
STEP
to use
Exam

CHAPTER 10

LOOPING

- FOR . . . TO
- NEXT
- STEP
- WHILE . . . . WEND

FOR . . . TO . . . NEXT . . . STEP

Often we need the computer to perform repetitive tasks. The looping process enables us to do just this without having to type in similar statements many times.

For example if we want many numbers cubed and then divided by 3 we don't have to type in various values; all we have to do is something like this:

Example:

```
10     FOR X = 1 TO 1Ø
20     PRINT X; (X ↑ 3)/3
3Ø     NEXT X
4Ø     END
       RUN
       1     .333333
       2     2.66667
       3     9
       4     21.3333
       5     41.6667
       6     72
       7     114.333
       8     17Ø.667
       9     243
       1Ø    333.333
```

From the above example we can see that the computer has cubed and divided by 3 the numbers between 1 and 10. The FOR . . . TO . . . statement, therefore, stipulates the range of numbers you wish to act on.

You will notice that the number were incremented (increased) by one each time. The increment size can be changed by using the STEP statement and a positive number. If a negative number follows the STEP statement we will get a decrement (decrease). It is also possible to use a decimal, an expression or a variable.

Example:

```
1Ø     FOR X = 1 TO 1Ø STEP 2
2Ø     PRINT X;(X ↑ 3)/3
3Ø     NEXT X
4Ø     END
       RUN
```

This will act on all the odd numbers between 1 and 10 and your screen will show the following numbers:

Example:

```
1     .333333
3     9
5     41.6667
7     114.333
9     243
```

You will also notice that each FOR loop must be closed with a NEXT statement. The variable name of the NEXT statement must be the same as the variable name of the FOR statement –in this case X. On your computer the variable name following the NEXT statement can be omitted if you wish. However it is good programming practice to put it in. This can avoid unexpected results (and errors) when you have a number of FOR-NEXT loops – particularly if they are "nested" (explained shortly).

Loops are very useful for writing tables as you will see from the following example.

Example:

```
10      REM TO PRINT A SINE AND COSINE TABLE
20      PRINT "SIN(X)", "COS(X)"
30      FOR X = Ø TO 2 STEP Ø.5
40      PRINT SIN(X), COS(X)
50      NEXT X
60      END
        RUN
SIN(X)                  COS(X)
Ø                       1
.479426                 .877583
.841471                 .540302
.997495                 .0707371
.909298                 -.416147
```

If you do use a variable name following your NEXT statement and you use 2 loops you must be careful not to cross your loops.

Example:

```
   10      ............
 ┌-20      FOR X = Ø TO 1Ø
 │ 30      ...........
┌┼-40      FOR Y = Ø TO 5
││ 50      ............
│└-60      NEXT X
└--70      NEXT Y
```

This causes crossed loops and will result in a NEXT WITHOUT FOR error message. The correct way is as follows:

```
  1Ø........
┌-2Ø        FOR X = Ø TO 1Ø
│ 3Ø......
│┌4Ø        FOR Y = Ø OT 5
││5Ø.........
│└6Ø     NEXT Y
└-7Ø      NEXT X
```

This gives you the loop for Y "nested" or fully inside the loop for X.

With your computer, the number of level of nesting for FOR—NEXT loops depends on the memory size of the computer.

WHILE . . . WEND

The combination of WHILE, WEND statement is another way to execute a series of BASIC statements in a loop as long as a given condition is true.

An example of WHILE/WEND is:

```
10                  I = Ø
20       WHILE      SQ < 99
30                  I = I + 1
40                  SQ = I ↑ 2
50                  PRINT I,
60                  PRINT SQ
70       WEND

RUN
1                   1
2                   4
3                   9
4                   16
5                   25
6                   36
7                   49
8                   64
9                   81
1Ø                  1ØØ
```

The purpose of this program is to calculate all the numbers whose square is just less than 99.

The statement loop consists of line numbers 30 to 60. The expression to be checked by WHILE statement is SQ < 99. When the value of SQ, computed from the loop, is greater than or equal to 99, the loop is not entered and program ends.

WHILE/WEND loops may be nested, i.e., place one loop inside another. But you should make sure that each WEND will match the most recent WHILE.

CHAPTER 11

SUBROUTINES

- GOSUB
- RETURN

GOSUB – RETURN

A program has a beginning and an end. It has a structure. This structure is made up of smaller building blocks. You may need some of these blocks or sections of the program many times in various places in the overall program. To help us deal with these similar smaller parts of the program we can use subroutines. The statements we use are **GOSUB** and **RETURN.**

```
1Ø          PRINT "HELLO"
2Ø          GOSUB 5Ø
3Ø          PRINT "GOODBYE"
4Ø          END
5Ø          PRINT "HOW ARE YOU?"
6Ø          GOSUB 8Ø
7Ø          RETURN
8Ø          PRINT "SEE YOU"
9Ø          RETURN
            RUN
            HELLO
            HOW ARE YOU?
            SEE YOU
            GOODBYE
```

The lines are executed in this order 1Ø, 2Ø, 5Ø, 6Ø, 8Ø, 9Ø, 7Ø, 3Ø, 4Ø. Hopefully you can see from this example how the GOSUB statement works.

The **GOSUB** statement tells the computer to move on to the line number indicated, that is to the line number following **GOSUB.** However unlike the **GOTO** command the GOSUB command makes the computer "remember" where it is jumping from, so it can come back again to the statement that immediately follows the GOSUB statement.

The computer will carry on with the subroutine until the **RETURN** statement is met. It is the **RETURN** statement that makes the computer go back to the statement following the **GOSUB** statement.

For another example:

2Ø GOSUB 6Ø tells the computer to jump to line 6Ø. The **RETURN** statement. On meeting the **RETURN** statement the computer will go back and start executing at the statement following the **GOSUB** statement in line 2Ø.

Have a look at this example and see if you can work out what's happening.

```
1Ø        FOR X = 1 TO 5
2Ø        GOSUB 6Ø
3Ø        PRINT X; S
4Ø        NEXT X
5Ø        END
6Ø        S = Ø
7Ø        FOR J = 1 TO X
8Ø        S = S + J
9Ø        NEXT J
1ØØ       RETURN
          RUN

          1    1
          2    3
          3    6
          4    1Ø
          5    15
```

Get the idea? Here the subroutine is the section from lines 6Ø to 1ØØ inclusive, and we are "calling it" (i.e. using it) a total of 5 times.

CHAPTER 12

LIST AND TABLES

- ARRAYS
- DIM
- OPTION
- ERASE

ARRAYS AND DIM

There are two types of variables – Simple Variables and Array Variables. Up to now we have been dealing with simple variables. Let us now take a look at the array type.

An array is an organised list of values which provides an efficient way of handling large amounts of information. The values can be either numbers or strings. To set up an array you first have to give the array a name and a size. The name can be either a letter or a string e.g. A$ (5).

It is easy to distinguish an array variable from a simple variable. The array variable is always followed by brackets containing a number. e.g. A(2), B(7), G5$(7). This number in the brackets is called a subscript.

WHY USE ARRAYS?

Let us suppose you have a number of books at home – say 100 books – and you want to index all your books. If we assign a variable to each book name for example:

Example:

```
1Ø      A1$ = "GONE WITH THE WIND"
2Ø      A2$ = "OLIVER TWIST"
        .
        .
        .
        .
        .
        .
1ØØØ    L1$ = "BASIC PROGRAMMING"
```

This would be very inefficient and time consuming. A better way of dealing with this list is to use an ARRAY. The variable A$ will stand for the list of books. Let us look at the following example:

Example:

```
1Ø      REM BOOK NAMES
2Ø      DIM A$(99)
3Ø      FOR X = Ø TO 99
4Ø      INPUT "BOOK NAME"; A$ (X)
50      NEXT X
        RUN
        BOOK NAME?
```

After you give a name after to the question mark, 'BOOK NAME?' will appear again underneath. This will carry on until your list is completed.

Before you can use an ARRAY it is necessary to use the DIM statement. Above we have DIM A$(99). This tells the computer to reserve space for the array called A$ and that array has 1ØØ subscript variables (from A$(Ø) to A$(99) inclusive). It is possible at a later time to sort, rearrange or print out this set of data. This type of array is called an one dimensional array and it deals with lists. DIM stands for dimension.

It is also possible to have a two dimensional array where we have two subscripts and we are dealing with numbers in the matrix form.

Let us suppose we have 5 students doing 3 exams. The results of the exams look like this.

	EXAM(1)	EXAM(2)	EXAM(3)
STUDENT 1	50%	70%	90%
STUDENT 2	63	42	36
STUDENT 3	20	62	50
STUDENT 4	70	75	84
STUDENT 5	93	82	68

These results can be recorded on the computer using a two dimensional array. We would have to start with the statement: DIM A (4, 2). Here 4 is one less than the number of students and 2 is one less than the number of columns and in this case exams. So A(3, Ø) will be 7Ø. This is the score of the fourth student in the first exam.

It is possible to have up to a three dimensional ARRAY – DIM A(3, 6, 2). The size of each dimension is limited by the memory size of the computer so remember; A(X, Y) is a two dimensional array variable, and A(X, Y, Z) is a three dimensional one.

Note that if you do not use the DIM statement the subscripts Ø – 1Ø are allowed for each dimension of each array used by default.

OPTION

The BASIC statement having the format:

```
OPTION BASE  Ø
```

or

```
OPTION BASE  1
```

is used to set the minimum value for an array subscript.

If you do not specify, the computer assumes that your program is OPTION BASE Ø.

For instance, OPTION BASE 1 tell the computer that the lowest value an array subscript may have is one.

ERASE

After you have set up some arrays at the start of your program, you may want to eliminate them to free up your memory.

The BASIC statement used to do this is:

```
ERASE       A, B, C, .......
```

Arrays having the same name may be redimensioned after they are ERASEd, or the previously allocated array space in memory may be used for other purposes.

Example:

```
1ØØ        DIM A (2Ø), B(2Ø)
.
.
.
4ØØ        ERASE A, B
41Ø        DIM A (1Ø), B(1Ø)
.
.
.
```

CHAPTER 13

READ, DATA, RESTORE

- READ
- DATA
- RESTORE

READ AND DATA

When it is necessary to enter a lot of information or data into the computer, using the INPUT statement can be very time consuming. To help us out here we can use the READ and DATA commands.

Example:

```
1Ø    DATA 1Ø, 6Ø, 7Ø, 8Ø, 9Ø
2Ø    READ A, B, C, D, E
3Ø    PRINT A; B; C; D; E
      RUN
      1Ø    6Ø    7Ø    8Ø    9Ø
```

The READ statement consists of a list of variable names with commas between each variable.

The DATA statement consists of a list of expressions separated by commas. These expressions can be either numeric or strings. The READ statement makes the computer look up the value of its variables form the DATA statement. When the computer goes to READ first it will assign the first expression from the DATA list. The next time it goes to READ it will assign the second value – and so on. If the READ runs out of DATA you will get '? OUT OF DATA ERROR'.

RESTORE

If you want to use the same data later on in the program you can do so by using the RESTORE statement.

Example:

```
1Ø    DATA 1, 3, 8, 9
2Ø    READ A, B, D
3Ø    RESTORE
4Ø    READ X, Y
5Ø    PRINT A; B
6Ø    PRINT X; Y
7Ø    END
      RUN
      1  3
      1  3
```

The RESTORE command makes subsequent READ statement get their values form the start of the first DATA statement.

Now see if you can work out what is happening here.

Example:

```
1Ø    REM FIND  AVERAGE
2Ø    DATA Ø.125, 3, Ø.6, 7
3Ø    DATA 23, 9.3, 25.2, 8

4Ø    S = Ø
5Ø    FOR I = 1 TO 8
6Ø    READ N
7Ø    S = S + N
8Ø    NEXT
9Ø    A = S/8
1ØØ   PRINT A
      RUN
      9.52813
```

Now using our student's examinations results form the chapter on arrays see how the READ and DATA commands can be used.

Example:

```
1Ø    CLS: DIM A (4, 2)
2Ø    PRINT "RESULT" : PRINT
3Ø    PRINT TAB(8); "EX(1), EX(2), EX(3)"
4Ø    PRINT
5Ø    FOR J = Ø TO 4
6Ø    PRINT "STUDENT"; J + 1;
7Ø    FOR I = Ø TO 2 : READ A(J, I): PRINT A (J,I);:
      NEXT : PRINT
8Ø    NEXT
9Ø    END
1ØØ   DATA 50, 7Ø, 9Ø, 63, 42, 36, 2Ø, 62, 5Ø, 7Ø, 75
11Ø   DATA 84, 93, 82, 68
      RUN
```

CHAPTER 14

PEEK AND POKE

- PEEK
- POKE

PEEK (address)

The PEEK function will return the value stored at the specified address in the memory of the computer. The value will be displayed in the form of a decimal number whose value is in range of Ø–255.

Example:

```
3Ø    A = PEEK (28672)
4Ø    PRINT A
```

This returns the value the program has at 28672 and gives this value to A. It should be noted that the address need not be a value; it can be an expression.

POKE address value, expression

The POKE function complements the PEEK function. It sends a value to the stated address location. You need therefore an address and value, and again the value has to be between Ø and 255.

Example:

```
1Ø    A = 1
2Ø    POKE 4096Ø, A
3Ø    B = PEEK (4096Ø)
4Ø    PRINT B
      RUN
      1
```

When using this command you must be very careful as it can destroy your program. It is wise to save the program before you execute POKE. It is not recommended for newcomers without prior knowledge of what it does. You can only POKE to the Random Access Memory (RAM), that is to the place where the computer stores the information it wants to keep like your BASIC program, variables, the picture for the television and the various musical notes. So maybe the address you indicate may not be in the memory of the computer. The possible address is in the range of –32768 to +65535. (but not all of this range is occupied by RAM).

CHAPTER 15

GRAPHICS

- GRAPHICS MODE VS TEXT MODE
- COLOR
- SET
- RES
- POINT
- DRAW
- MOVE
- GRAPHIC CHARACTERS

GRAPHICS MODE VS TEXT MODE

So far we have been creating only characters on the screen display. In fact, your computer is capable of two different categories of display. Once the power is turned on the computer is in the normal text mode, with maximum of 24 rows multiplied by 40 columns. You may also have tried pressing the CTRL key at power up so as to enter another text mode which can display a larger number of smaller characters. If you haven't, try it now. Double the amount of information, (24 rows by 80 columns), can be put on one screen in this mode.

You can also switch to a number of graphics modes that your computer offers for programming and games purposes.

To get back to text mode for character display, your program should execute a TEXT command. This BASIC command takes on a few forms, with slightly different effects.

COMMAND	FUNCTION
TEXT	Selects text mode display.
TEXT 4Ø	Selects 40 columns X 24 row text mode. If you are already in 40 column mode, no particular effect occurs. Otherwise, it clears the text screen.
TEXT 8Ø	Selects 80 column X 24 row text mode. If already in 80 column mode, no particular effect occurs. Otherwise, it clears the text screen.

The following is a brief description of the commands used for entering different graphics modes. One thing to keep in mind is that after invoking graphics mode commands. Your program should be executing continuously or staying in a loop. Otherwise when it stops execution and the control is passed back to the BASIC Interpreter, the screen display will automatically flip back to text mode.

In addition, LASER 350 users should note that some graphics modes are not available to them unless a 64K RAM Memory Expansion Module optional is installed. An "OUT OF MEMORY" error message will be issued to remind the user.

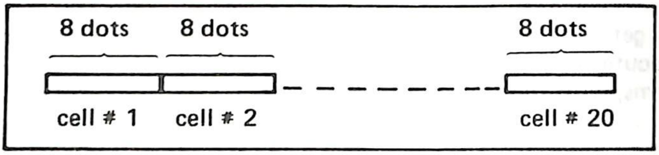
GR/GRØ — GR, which is same as GR Ø, will set the screen to a graphics display of 160H X 96V, with 16 different colors available.

NOTE: The expression 160H X 96V means there are 160 different pixels on a horizontal line and a total of 96 such lines in the vertical direction. These figures indicates the resolution of the graphics mode. The higher the figures, the greater the resoltuion and the more detailed the graphics will be.

GR 1

GR1 will set the screen to 160H X 192V. There may be 16 different colors on the whole screen, but every set of 8 horizontal dots, counting from left to right, can only have 2 different colors out of 16 colors.

To visualize this limitation, you may partition one horizontal line of 160 pixels in to 20 cells, with each cell containing 8 horizontal dots.



Then the 8 dots in each cell may only have 2 different colors out of 16. For instance, if cell = 1 is RED and GREEN, then the color CYAN cannot be present in this cell.

GR 2

GR2 will set the screen to 320H X 192V graphics mode. Any two out of 16 different colors are available on the whole screen.

GR 3

(Only for LASER 500 and LASER 700. Available for LASER 350 only with optional 64K RAM expansion card.)

GR3 will set the screen to 160H X 192V graphics mode. 16 different colors are available for every dot.

GR 4

(Only for LASER 500 and LASER 700. Available for LASER 350 only with optional 64K RAM expansion card.)

GR4 will set the screen to 320H X 192V graphics mode. Color limitation is similar to GR 1. That is, 16 different colors are possible on the whole screen, but every set of 8 horizontal dots, counting from left to right, can only have 2 different colors out of 16 colors.

GR 5 (Only for LASER 500 and LASER 700. Available for LASER 350 only with optional 64K RAM expansion)

GR5 will set the screen to 640H X 192V graphics mode. Any two out of 16 different colors are available for the whole screen.

The following table summarizes the graphics modes available:

Graphic Mode	Memory Requirement	Resolution (H X V)	Color
0	LASER 350 LASER 500 LASER 700	160 X 96	16
1	LASER 350 LASER 500 LASER 700	160 X 192	16 (8 dots 2 colors out of 16 colors)
2	LASER 350 LASER 500 LASER 700	160 X 192	2
3	LASER 350 with optional 64K RAM M/E LASER 500 LASER 700	160 X 192	16
4	LASER 350 with optional 64K RAM M/E LASER 500 LASER 700	320 X 192	16 (8 dots 2 colors out of 16 colors)
5	LASER 350 with optional 64K RAM M/E LASER 500 LASER 700	640 X 192	2

Table 15-1 Graphics Modes

COLOR

The colors, mentioned above, only apply to the 'Foreground' color and 'Background' color of the display screen. In fact, your computer is capable of altering the 'Backdrop' color through BASIC commands.

Backdrop color always referes to the color of the area between the border of your monitor/TV screen and the active display area, i.e., area in which you can put text or graphics on. On the other hand, exact meaning of Foreground and Background color are different in each display mode.

In 40 column text mode, each individual character has its own programmable Foreground/Background color. The meaning of these colors are illustrated in Fig.15–1.

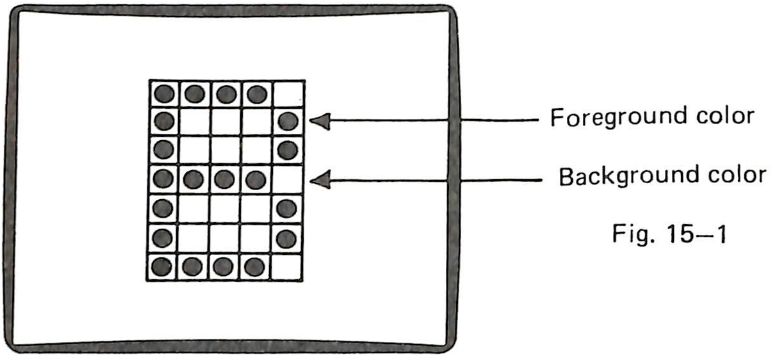


Fig. 15–1

In 80 column text mode, there can be only 2 colors out of 16 possible colors on the whole active display area. The color of the dots on the dot matrix is referred to as the Foreground color, whereas the remaining color of the active display area is the Background color. The actual meaning are illustrated in Fig.15–2.

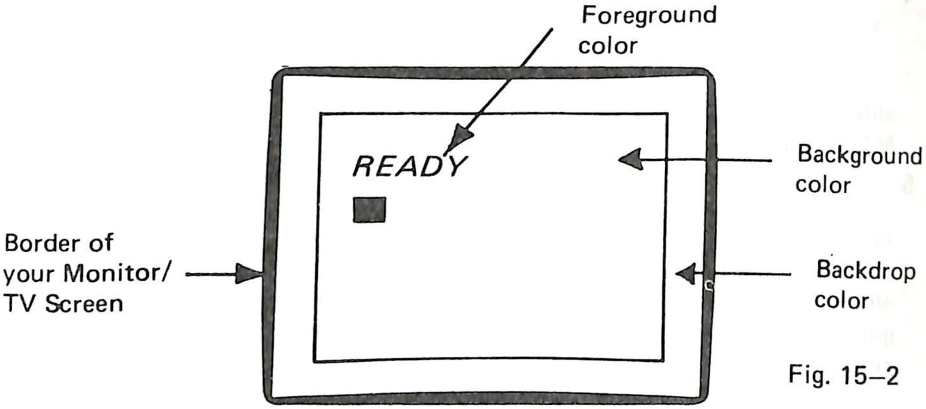


Fig. 15–2

In graphics mode GR Ø and GR 3, where each individual pixel can has its own color out of 16 possible colors, but always has a Background color of black. It is not possible to change the Background color by software. The Foreground color is the color of the dots being plotted onto the active display area, as illustrated below. You may assume that a diagonal line is plotted on these particular graphics modes.

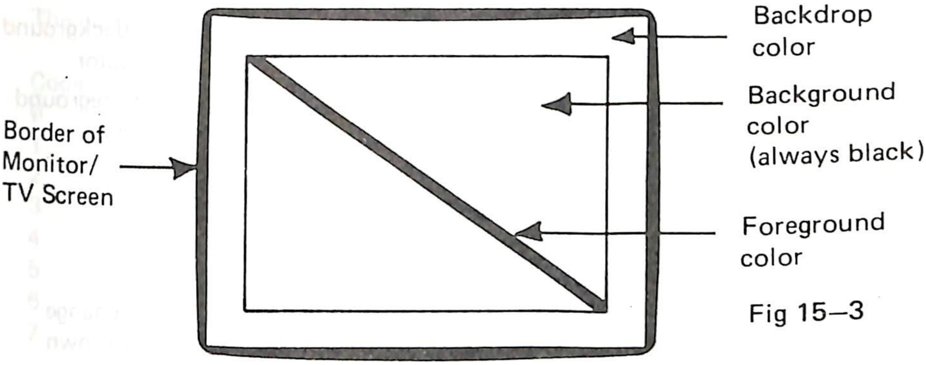


Fig 15–3

In graphic mode GR1 and GR4, each set of 8 consecutive dots, counting from the left, can only have 2 out of the 16 possible colors. Fig. 15–4 is an enlarged view showing a typical set of 8 consecutive dots. Each such set has its own Foreground color.

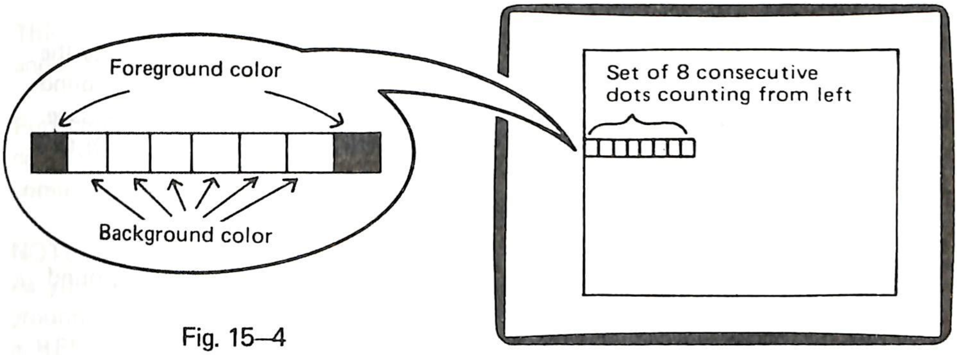


Fig. 15–4

Any position on the set of 8-dots being "turned on" by the SET, or the DRAW command, is in its foreground color. Those positions being "turned off" or untouched are in their Background color. Commands to turn on and turn off a dot will be covered in the following pages.

In graphics mode GR 2 and GR 5, the whole active display area can only have 2 colors out of the 16 possible colors. The meaning of Foreground and Background color are similar to those in 80 column text mode. Fig. 15–5 Illustrates a typical display with a diagonal line plotted.

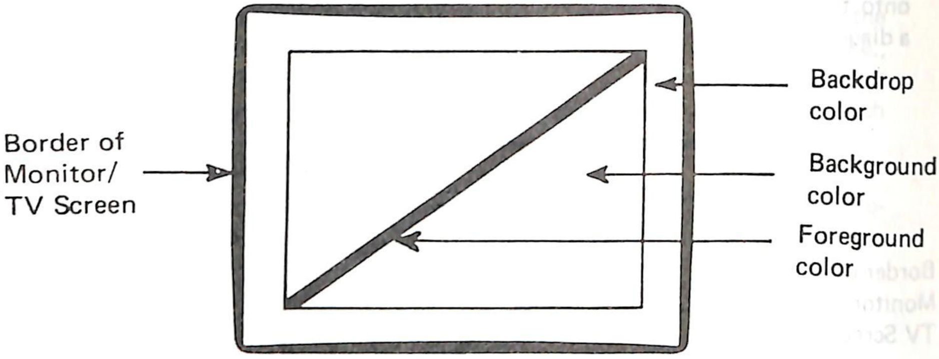


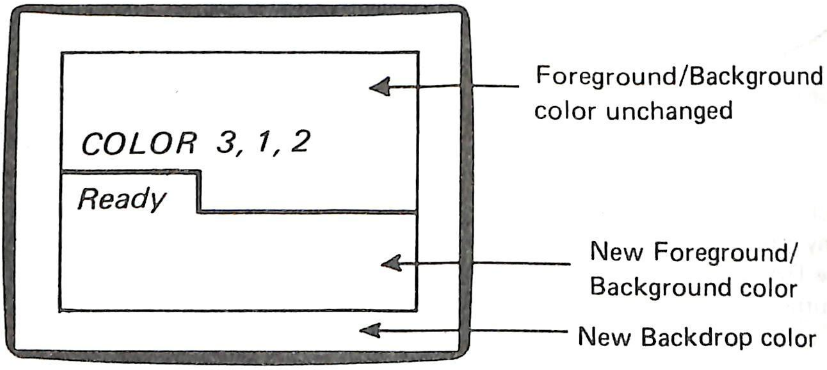
FIG. 15–5

The BASIC Interpreter of your computer allows you to set or change the Foreground, Background, and even the Backdrop color at your own will.

For example, if you would like to have a colourful 40 column TEXT display with green Backdrop color, blue background color and cyan foreground text, you should type:

```
COLOR 3, 1, 2
```

The Backdrop color would change to green as soon as you press the RETURN key. However, the blue background and cyan foreground affects only the "Ready" message and the lines below that message. Lines above will remain unchanged. More general form of the COLOR statement is:


COLOR 3, 1, 2
Ready
Foreground/Background color unchanged
New Foreground/ Background color
New Backdrop color

More general form of the COLOR statement is:

```
COLOR I, J, K
```

Where I, J, K are numbers representing color codes of Foreground color, Background color, and Backdrop color respectively.

The color code and the color represented are summarized as follows:

Code	Color	Code	Color
Ø	black	8	grey—2
1	blue	9	light blue
2	green	1Ø	light green
3	cyan	11	light cyan
4	red	12	light red
5	magenta	13	light magenta
6	yellow	14	light yellow
7	grey—1	15	white

You may omit one or two of I, J, K if you do not want to change it. But must remember to place commas in the command if I, OR J is to be omitted.

Example:

```
COLOR , , K
```

This will change the Backdrop color to that represented by code K and keep the Foreground/Background colors unchanged.

Furthermore, the latest COLOR command issued will determine the color of the graphics that you are going to plot, as described in the coming pages.

NOTE: There is one point which the user of a color TV should know. As you might have observed, some combinations of Foreground/Background colors result in very blurred appearance. One such combination is RED Foreground text on GREEN Background. This is the result of a color TV standard that was internationally adopted. Color TV sets are limited in their ability to place certain colors next to each other on the same line. Therefore, some colors work excellently, while other combinations are to be avoided.

For LASER 700 users using RGB monitor, only 8 colors are available. Color code 8—15 are same as 0—7.

GRAPHICS COMMANDS

The following commands are for creating graphics in the various graphics modes.

SET (X, Y)

This command plots a dot at a specified location on the graphic screen, which is determined by the values of X and Y. The value of X and Y must be in the range that the current graphics mode allows.

The color of the dot being plotted is determined by the latest Foreground color used. So you should have issued a COLOR I, J, K before the SET (X, Y) command.

When this command is executed, the dot given by (X, Y) would be set to the latest foreground color. The Foreground color of the set of 8 consecutive dots in GR 1 & GR 4 would also be changed, making color changes in all 8 dots of the set.

RES (X, Y)

This BASIC statement is complementary to the SET (X, Y) statement. This is used to wipe out a point switched on by the SET command. Similary, X, Y specify the location of the point and must be in the range that the current graphics mode allows.

When this command is executed, the dot given by X, Y will be made the same color as the background.

Note: Don't confuse RES command with the RESET key on LASER 500/700 keyboard.

Example:

```
10 COLOR 15, Ø : GR 1
20 FOR X = Ø TO 159
30 Y = INT (6/5 * X)
40 SET (X, Y)
50 NEXT X
6Ø FOR I = 1 TO 2ØØØ : NEXT I
70 FOR X = Ø TO 159
8Ø Y = INT (6/5 * Y)
9Ø RES (X, Y)
1ØØ NEXT X
11Ø TEXT 4Ø

RUN
```

This example will plot a diagonal line on the GR 1 graphic screen, and then rub this line out. When this program is compieted, the computer will return to text mode.

Note: If you want to keep the computer in graphics mode, replace line 110 by:

```
110 GOTO 1Ø
```

Press CTRL C to stop this program.

POINT (X, Y)

This tells you if a specific point has been plotted by the SET command. If the specified point has been plotted, POINT (X, Y) will return its color code. Usually it is used with the IF – THEN – ELSE statement.

Example:

```
8Ø   SET (4Ø, 2Ø)
9Ø   IF POINT (4Ø, 2Ø) THEN
     PRINT "YES" ELSE PRINT "NO"
```

DRAW (X , Y)

This will draw a straight line from the last drawn coordinates, say (XO, YO), to the new co-ordinate (X, Y) by "turning on" all the dots along the line.

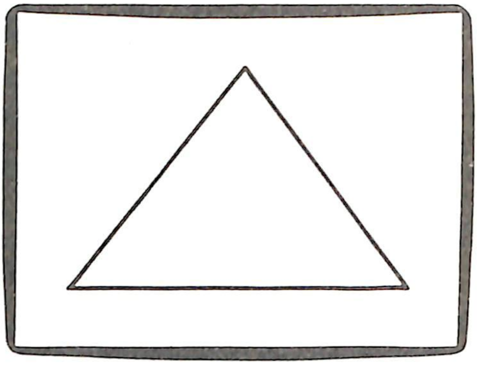
The next time you use the DRAW command, another straight line will be drawn from the previous (X, Y) to the new (X, Y). The initial value of (X, Y) will be taken as (Ø, Ø). The color of the line is determined by the latest issued COLOR command.

You may use a series of such DRAW statements to create your own polygon shape figure.

Example:

```
1Ø   COLOR 15, Ø, Ø : GR 1
2Ø   MOVE (79, 15)
3Ø   DRAW (149, 1ØØ) : DRAW (9, 1ØØ) : DRAW (79, 15)
4Ø   GOTO 2Ø
RUN
```

A triangular shape would be drawn as follows:



Press CTRL C to stop.

MOVE (X, Y)

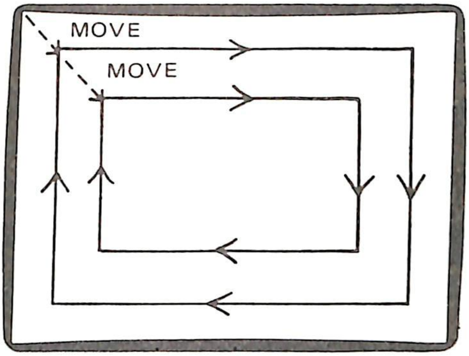
This statement moves the (X, Y) coordinates of last DRAW or MOVE command to a new X, Y position. Although the command by itself gives no visible effect on the graphic screen, it will affect the next DRAW statement.

This statment can be used in conjunction with the DRAW (X, Y) command to draw shapes composed of straight lines, while at the same time, some segments are to be skipped.

The following example shows a simple way to use the MOVE and DRAW statements to produce two rectangles, one inside the other.

Example:

```
10   GR 1
20   MOVE (9, 11) : DRAW (149, 11) : DRAW (149, 180) :
     DRAW (9, 180) : DRAW (9, 11)
30   MOVE (29, 31) : DRAW (129, 31) : DRAW (129, 160) :
     DRAW (29, 160) : DRAW (29, 31)
40   GOTO 20
RUN
```



Press CTRL C to stop.

GRAPHICS CHARACTERS

Apart from plotting points and drawing shapes on graphic modes in the previous pages, your computer is also capable of displaying "Graphic Characters" onto the text screen. Of course these are not full graphics in the sense that you cannot change every point on the screen. You may only choose from a predefined set of graphic character patterns.

There are totally 32 graphic characters available in the current version of your computer. Details of their patterns may be found in Appendix.

In order to produce these graphic character on the screen, you have to press the key GRAPH on LASER 500/700 or CTRL ~ on the 350. Once you press this key, you will stay in the graphic character entry mode and you are free to type in more graphic characters. In this mode, some of the keys on the keyboard will have different meanings from those printed on top of the keys.

For example, after pressing GRAPH , the key A when pressed, will generate a ' T ' symbol on the screen. If you want to leave the graphic character entry mode, just press the GRAPH (CTRL ~ for LASER 350 users) key or simply the RETURN key.

Here is a simple program showing you how to use these graphics characters.

Example:

```
10  CLS
20  FOR I = 1 TO 40
30  PRINT "♠";
40  NEXT
RUN
```

```
♠♠♠· · · · · · · · · · · · · · · ·♠
READY
```

Note: While entering line 30 of this program, you have to press [GRAPH] (or [CTRL] [≂] for LASER 350), then press the '[I]' key to produce the symbol [♠] , and finally press [GRAPH] (or [CTRL] [≂]) to get back into normal character entry.

SOUND

Anothe
sound

Examp

This
variabl
note

It is
durat
diffe

By
selec

DURA

Code

CHAPTER 16

SOUND AND MUSIC

- SOUND
- MUSIC

SOUND

Another interesting feature of the computer is its ability to produce sound. Here is an example program.

Example:

```
1Ø  FOR I = 1 TO 8
2Ø  READ X
3Ø  SOUND X, 7
4Ø  NEXT I
5Ø  DATA 16, 18, 2Ø, 21, 23, 25, 27, 28
    RUN
```

This will produce 8 notes going up the scale. In this program the variable X is the frequency and the constant 7 is the duration of the note.

It is possible to get 31 different frequencies and 9 different note durations. The tables below show how which codes produce the different frequencies and duration.

By varying the notes and duration therefore, and using the tables below, it is possible to produce tunes of your choice.

DURATiON

Code	Note	Note length
1	𝅘𝅥𝅰	$\frac{1}{8}$
2	𝅘𝅥𝅯	$\frac{1}{4}$
3	𝅘𝅥𝅯.	$\frac{3}{8}$
4	𝅘𝅥𝅮	$\frac{1}{2}$
5	𝅘𝅥𝅮.	$\frac{3}{4}$
6	𝅘𝅥	1
7	𝅘𝅥.	$1\frac{1}{2}$
8	𝅗𝅥	2
9	𝅗𝅥.	3

FREQUENCY

Code	Pitch	Code	Pitch
Ø	rest	16	C4
1	A2	17	C#4
2	A # 2	18	D4
3	B2	19	D # 4
4	C3	20	E4
5	C # 3	21	F4
6	D3	22	F # 4
7	D # 3	23	G4
8	E3	24	G # 4
9	F	25	A4
1Ø	F # 3	26	A # 4
11	G3	27	B4
12	G # 3	28	C5
13	A3	29	C # 5
14	A # 3	30	D5
15	B3	31	D # 5

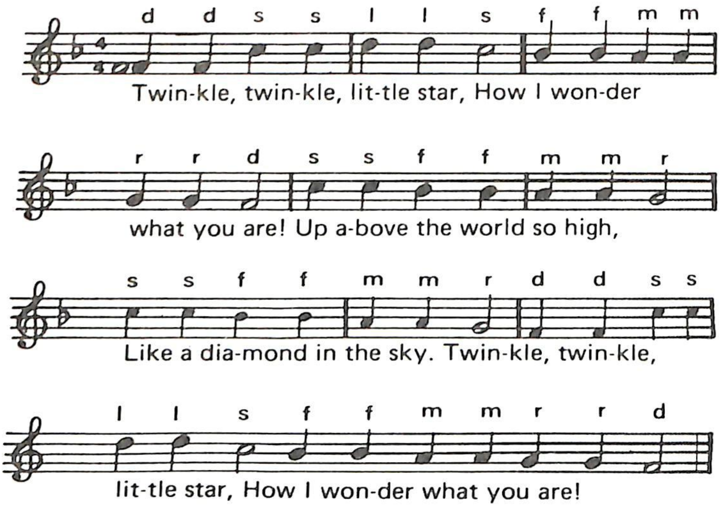
MUSIC

Below you can see how a musical score is transposed into Data for the computer.

TWINKLE, TWINKLE, LITTLE STAR

Nursery Rhyme

Key F


d d s s l l s f f m m
Twin-kle, twin-kle, lit-tle star, How I won-der
r r d s s f f m m r
what you are! Up a-bove the world so high,
s s f f m m r d d s s
Like a dia-mond in the sky. Twin-kle, twin-kle,
l l s f f m m r r d
lit-tle star, How I won-der what you are!

TWINKLE, TWINKLE, LITTLE STAR

Example:

```
2 DATA 21, 4, 21, 4, 28, 4, 28, 4, 3Ø, 4, 3Ø, 4, 28, 6, 26
        4, 26, 4, 25, 4
4 DATA 25, 4, 23, 4, 23, 4, 21, 6, 28,4, 28, 4,26, 4, 26
        4, 25, 4, 25, 4, 23, 6
6 DATA 28, 4, 28, 4, 26, 4, 26, 4, 25, 4, 25, 4,28, 6,
        21, 4, 21, 4, 28, 4, 28, 4
8 DATA 3Ø, 4, 3Ø, 4, 28, 6, 26, 4, 26, 4, 25, 4, 25, 4
        23, 4, 23, 4, 21, 6
10 FOR I = 1 TO 42: READ F, D: SOUND F, D:
NEXT: END
```

CHAPTER 17

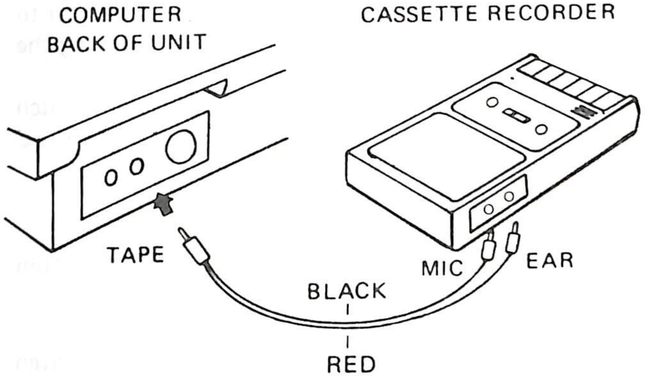
STORING PROGRAM ON TAPE

- SETTING IT UP
- CLOAD
- CSAVE
- VERIFY
- CRUN
- PRINT #
- INPUT #

You may have developed some programs which you want to retain. It is too much trouble to type in a program, especially if it is long, every time you want to use it. This problem is easily solved. You can store your programs on tape and whenever you need them, load them into memory.

SETTING IT UP

To do this you need an ordinary cassette tape recorder, a cassette tape and interconnecting cords. Connect the recorder as shown in the picture.



You need to be familiar with three commands, namely, **CSAVE, CLOAD** and **VERIFY.**

There is a file name for each program on the tape. A file name is a "must" for saving a program but not absolutly necessary for loading and verifying a program.

The file name can be one to sixteen characters in length. The first character must be a letter; the rest can be any character. For our purpose saving a program means transferring a program that you have typed into the computer, to a tape.

Verifying a program means checking to make sure that the program on the tape is the same as the program in the computer.

Loading a program means transferring a program from the tape to the computer.

CLOAD "FILE NAME"

The procedure for loading a program from the tape to the computer is as follows:

1. Load the tape containing the required program into the recorder.
2. Rewind the tape to the start of the required program.
3. Type the COMMAND *CLOAD "FILE NAME".*
   BE SURE NOT TO PRESS THE RETURN KEY
4. Press the play button on the recorder.
5. Press the RETURN key.
6. If the computer finds no program on the tape, the statement **WAITING** is displayed on the screen. If you want the computer to come out of WAITING, PRESS CTRL C before stopping the cassette recorder.
7. If the incoming program has the file name which does not match with the specified one, then the statement 'FOUND *T: FILE NAME'* appears and the program is skipped.
8. The desired program is loaded with the statement *'LOADING T : FILE NAME'* appears.
9. When the statement READY is displayed, press the STOP button on the recorder.

Let us suppose there are three programs on the tape and you have given them file names: PROGRAM 1, PROGRAM 2, PROGRAM 3. You want PROGRAM 3 and it is at the end of the tape. You can type: *CLOAD "PROGRAM 3".* Then your screen will show:

Example:

```
CLOAD "PROGRAM 3"
WAITING
FOUND T: PROGRAM 1
FOUND T:PROGRAM 2
LOADING T: PROGRAM 3
READY
```

If you know the location of PROGRAM 3 and you set the tape at the beginning of this program the screen will show:

Example:

```
CLOAD "PROGRAM 3"
WAITING
LOADING T: PROGRAM 3
READY
```

NOTE T: stands for TEXT file.

CSAVE "FILE NAME"

If you wish to save a program, make sure that you use a good quality tape. The quality of the tape can affect the quality of your recording.

It is important to set the volume of the cassette recorder within a proper range. This range will vary from one recorder to another. The tone should be set to MAXIMUM level.

The procedure for storing/saving a program is as follows:

1. Type in the complete program. It is advisable to use a short program at the start. Longer programs can be saved once you have achieved success with a short program.

2. Type in the COMMAND *CSAVE "FILE NAME"*
   Remember a File Name is a "MUST" for saving a program.
   BE SURE NOT TO PRESS THE RETURN KEY

3. Load the recorder with a good quality tape.

4. Press the PLAY and RECORD buttons on the recorder.

5. Press the RETURN key.

   The flashing CURSOR will disappear and the storing begins.

6. When the flashing CURSOR reappears, the storing is completed.

7. Press the STOP button on the recorder.
   The program that you typed in is now stored on a tape. To make sure that it is stored the user may verify this for himself.

VERIFY "FILE NAME"

To verify a program on tape (ususlly just after you have **CSAVE'd** it). The procedure is as follows:

1. List the program in the computer to make sure that the program still exists.

2. Type the COMMAND *VERIFY "FILE NAME"*
   BE SURE NOT TO PRESS THE RETURN KEY

3. Press the PLAY button on the recorder.

4. Press the RETURN key. The flashing CUROSR will disappear and the verifying begins.

Example:

```
VERIFY "PROGRAM 2"
WAITING
FOUND T: PROGRAM 1
LOADING T: PROGRAM 2
VERIFY OK
READY
```

5. The **'OK'** statement tells us that the programs on the tape are the same as the program in the computer.

6. If the verifying reveals an error, the statement **'VERIFY ERROR'** will be displayed on the screen. This statement shows that the program on the tape is different from the one in the computer. In this case the user should **CSAVE** the program and verify it once again.

You can verify that there is a program on the tape by listening. If there is a program on the tape and you run it on the cassette recorder, the recorder will give out a distinctive sound.

CRUN "FILE NAME"

One more powerful command **'CRUN'** can be used. This command is similar to **'CLOAD'** except that the loaded program will start execution automatically after the loading is completed.

The four cassette interface commands, **CSAVE, VERIFY, CLOAD** and **CRUN** help the user to save his programs, verify them, and get them back into the computer to execute. The user should pay attention to the volume level of the recorder. Cassette interface is a means of inexpensive MASS STORAGE.

There are two more cassette type commands that the user should become familiar with. They are:

PRINT #

PRINT # "FILENAME", item list.
Sends the values of the specified variables or data onto a cassette tape. It is understood that the recorder must be properly connected and set in record mode when this statment is executed.

INPUT

INPUT # "FILE NAME", item list.
Inputs the specified number of values stored on cassette and assigns them to the specified variable names.

Example:

```
10 PRINT # "KAM", 1, 2, 3, 4, 5
   RUN
```

The data constants 1 to 5 are saved in the data file "KAM". BE SURE to put your cassette recorder in RECORD mode BEFORE execution.

Example:

```
1Ø INPUT # "KAM", A, B, C, D, E
2Ø PRINT A;B;C;D;E;
   RUN
   FOUND D:KAM
    1    2    3    4    5
```

The data in the data file "KAM" are assigned to the variables A to E. BE SURE to put your cassette recorder in PLAY mode BEFORE execution.

NOTE: D stands for DATA file.

NOTE: If you are still unsuccessful in trying to **CSAVE, CLOAD, CRUN** or **VERIFY** your tape programs after several attempts, you may try to use another brand or model of cassette recorder to test the result. This is because not all of the recorders are suitable for data recording.

CHAPTER 18

PRINT FORMATTING COMMANDS

- WIDTH
- TAB
- NULL
- SPC
- SPACE$
- PRINT USING
- WRITE

WIDTH LPRINT integer expression

This statement sets the printed line width in number of characters for the text screen or line printer.

If the LPRINT option is omitted, the line width is set at the text screen. If LPRINT is included, line width is set at the line printer.

The 'integer expression' must be within 15 to 255 with initial default value of 255.

Example:

```
10 PRINT "ABCDEFGHIJKLMNOPQR"
READY
RUN
ABCDEFGHIJKLMNOPQR
READY
WIDTH 16
READY
RUN
ABCDEFGHIJKLMNOP
QR
READY
```

TAB (I)

This command works in very much the same way as the TAB on a typewriter. In this case it moves the cursor to a particular position of a line.

If the current print position is already beyond space I, TAB goes to that position on the next line. I must be in the range 1 to 255 inclusive. TAB may only be used in the PRINT or LPRINT statements.

Example:

```
40 PRINT TAB(6); 1; TAB(20);1
RUN
   1         1
```

NULL n

This command is used to set the number of nulls to be printed at the end of each line.

Example:

```
READY
NULL   2
READY
100  INPUT X
200  IF X < 5Ø  GOTO 8ØØ
 :
```

Two null characters will be printed after each line.

SPC (I)

The SPC (I) statement prints I blanks on the screen. It may be inserted in your program to seperate two printed items by a specified number of spaces, without tediously typing a large number fo spaces in entering your program.

The value I must be in the range Ø to 255. A '; ' is assumed to follow the SPC (I) command, so that the next item printed will follow immediately after the spaces.

Example:

```
10 PRINT "OVER" SPC(15) "THERE"
RUN
OVER • • • • • • • • • • • • THERE
```

SPACE$ (X)

This statement is itself a string function which returns a string of spaces of length X. The expression X is rounded to integer and must be in the range Ø to 255.

When used with the PRINT command, neatly laid out results will be printed by a program.

Example:

```
1Ø FOR I = 1 TO 5
2Ø X$ = SPACE $ (I)
3Ø PRINT X$; I
4Ø NEXT I
READY
RUN
1
     2
        3
            4
               5
```

PRINT USING

This command is useful in allowing you to state how you want your lines printed. It is designed particularly for reports and tables.

It takes the form of:

```
PRINT USING.string; value or string
```

This string or value can be either a variable or constant. What will happen is that the string or value to the right of the semi-colon will be inserted as specified by the field specifiers,the preceding string.

A) "!" This specifies that only the first character in the given string is to be printed.

```
1Ø  A$ = "ASDF"
2Ø  PRINT USING "!";A$
RUN
A
```

B) "#" A number sign is used to represent each digit position in a numeric field. Digit positions are always filled. If the number to be printed has fewer digits than the positions specified, the number will be right justified (preceded by spaces) in the field.

" . " A decimal point may be inserted at any position in the field. If the format string specifies that a digit is to precede the decimal point, the digit will always be printed (as Ø if necessary). Numbers are rounded as necessary.

Example:

```
PRINT USING "# #. # #"; .78
Ø.78
```

C) " +" A plus sign at the beginning or end of the format string will cause the sign of the number (plus or minus) to be printed before or after the number.

" – " A minus sign at the end of the format field will cause negative numbers to be printed with a trailing minus sign.

Example:

```
PRINT USING "+ # #. # #";–68.95
–68.95
PRINT USING "# #.# # –";–68.95
68.95–
```

D) " **" A double asterisk at the beginning of the format string causes leading spaces in the numeric field to be filled with asterisks. The ** also specifies positions for two more digits.

Example:

```
PRINT USING " * * #. #";–Ø.9
*–Ø.9
```

E) " $$ " A double dollar sign causes a dollar sign to be printed to the immediate left of the formatted number. The $$ specifies two more digit positions, one of which is the dollar sign. The exponential format cannot be used with $$. Negative numbers cannot be used unless the minus sign trails to the right.

Example:

```
PRINT USING "$$ # # #.# #"; 456.78
$456.78
```

F) " **$ " The **$ at the beginning of a format string combines the effects of the above two symbols. Leading spaces will be asterisk filled and a dollar sign will be printed before the number. **$ specifies three more digit positions, one of which is the dollar sign.

G) " , " A comma that is to the left of the decimal point in a formatting string causes a comma to be printed to the left of every third digit to the left of the decimal point. A comma that is at the end of the format string is printed as part of the string. A comma specifies another digit position.

Example:

```
PRINT USING "# # # # , # #"; 1234.5
1,234.5Ø
```

H) " % " If the number to be printed is larger than the specified numeric field, a percent sign is printed in front of the number. If rounding causes the number to exceed the field, a percent sign will be printed in front of the rounded number.

Example:

```
PRINT USING "# # . # # "; 111.22
% 111.22
PRINT USING ". # #";.999
%1.ØØ
```

WRITE

The WRITE command is similar to the PRINT command, except that commas and double quotation marks are generated along with the printed text.

The format of WRITE is

```
WRITE expression, expression, ...........
```

Where the expressions must be separated by commas.

The output produced from WRITE also contains commas for numerical values and double quotation marks for strings.

Example:

```
1Ø   A = 8Ø:B=9Ø : C$="THAT'S ALL"
2Ø   WRITE A, B, C$
RUN
8Ø, 9Ø,"THAT'S ALL"
READY
```

CHAPTER 19

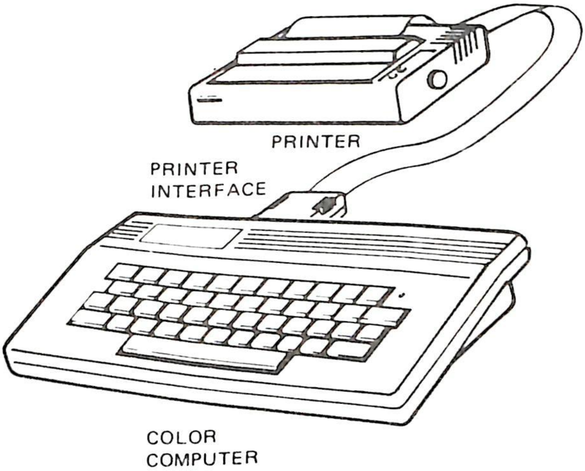
USING A PRINTER

- LLIST
- LPRINT

THE PRINTER (OPTIONAL)

To further expand the capabilities of your computer you can acquire a PRINTER. This can be attached to your computer by means of a PRINTER INTERFACE. If you decide to acquire a PRINTER, you will receive a separate leaflet containing detailed operating instructions.

SETTING UP THE PRINTER



To operate the computer successfully you need to familiar with the following commands **LLIST, LPRINT**

LLIST

This performs a similar function in relation to the PRINTER as **LIST** does in relation to the TV screen. **LLIST** outputs to the PRINTER, to give you a "hard copy" of your programs.

LPRINT

This command (and statements) is similar to **PRINT,** except that **LPRINT** is used with the PRINTER.

CHAPTER 20

BASIC PROGRAMMING UTILITY COMMANDS AND FUNCTIONS

- AUTO
- RENUM
- FREE
- SWAP
- POS
- LPOS
- TRON/TROFF
- ERR/ERL VARIABLES
- ERROR
- INP
- OUT
- WAIT
- KEY
- JOY

AUTO LINE NUMBER, INCREMENT

This statement is a handy tool which generates a line number automatically after every carriage return. It frees you from the tedious job of typing line numbers while entering long programs. Moreover, it avoids the danger of typing the same line number, which might destroy your existing program lines.

AUTO begins numbering at your specified line number and numbers each subsequent line by the specified increment. If you omit the optional parameters, the computer assumes the increment to be the last AUTO statement specified, or the pre-set value of 1Ø.

Example:

AUTO 1ØØ, 5Ø	*Generates line number 1ØØ, 15Ø, 2ØØ ....*
AUTO	*Generates line number 1Ø, 2Ø, 3Ø, 4Ø ....*

If AUTO produces a line number that is already in use, then the line will be listed. Any input will replace the existing line. Typing a carriage return immediately will save that line and generate the next line number.

However, if a new line number generated is terminated by a carriage return with no input typed in, then an "undefined line number" error will be printed.

To terminate AUTO, type CTRL C . The line in which Control—C is typed is not saved, so remember to make it separate line.

RENUM new number, old number, increment

This is another handly tool for re-arranging your program line numbers. It makes the programme more readable and more comprehensive and makes it easier to insert between two existing lines.

In the above expression, new number is the first line number to be used in the new sequence, old number is the line number in the current program where renumbering is to begin, increment is the space between line numbers. If you don't specify New Number and Increment, both will be assumed to be 10. If Old Number is omitted, the entire program will be renumbered.

Moreover, RENUM also changes all line number references following GOTO, GOSUB, THEN, ON . . . GOTO and ON . . . GOSUB . . . etc. to reflect new line numbers.

Example:

RENUM	Renumber the entire program. The first new line number will be 1Ø, with subsequent lines each increasing by an increment of 1Ø.
RENUM 3ØØ,, 5Ø	Renumber the entire program. The first new line number will be 3ØØ, with subsequent lines each increased by an increment of 5Ø.
RENUM 1ØØØ, 9ØØ, 2Ø	Renumber the lines from 900 up so they start with line number 1ØØØ and by an increment of 2Ø.

FRE (Ø)/FRE (X$)

The expressions following FRE are dummy arguments. This statement returns the number of bytes in memory not being used by your BASIC program.

Strings in BASIC can have different lengths, but need to be manipulated. This frequently causes the internal memory become very fragmented. Using this statement with a dummy argument can force BASIC to gather up all the loose fragments into single piles. This is known as "garbage collection".

Appropriate use of this function frees up areas of memory, and can often give you a surprising amount more memory space to work with.

Example:

```
PRINT FRE (Ø)
14542
READY
```

SWAP variable 1, variable 2

This statement allows the program to interchange the values, or content of two variables. The two variables may be of any type such as integer, string, floating point . . . etc. But the two variables must be of the same type in order to be interchangeable.

Example:

```
10    A = 10:B= 20
20    PRINT A,B
30    SWAP A, B
40    PRINT A, B
READY
RUN
10    20
20    10
READY
```

POS (X)

This statement returns the current cursor position. The leftmost position is 1. X is dummy argument which does not affect the result.

Example:

```
10  IF POS(X) >60 THEN PRINT CHR$(13)
```

LPOS (X)

This statement returns the current position of the line printer print head within the line printer buffer. It does not necessary give the physical position of the print head.

X is also a dummy argument here.

Example:

```
100  IF LPOS (X) >60 THEN LPRINT CHR$(13)
```

TRON/TROFF

TRON/TROFF commands are useful features for determining where a program is going wrong (debugging). After TRON is executed in either immediate mode or deferred mode, each line number of the program is printed as it is executed. These line numbers apppear enclosed in square brackets. This trace mode is turned off by the TROFF statement or when a NEW command is executed.

Example:

```
TRON
READY
LIST
1Ø    K = 1Ø
2Ø    FOR J = 1 TO 2
3Ø    L = K + 1Ø
4Ø    PRINT J ; K ; L
5Ø    K = K + 1Ø
6Ø    NEXT
7Ø    END
READY
RUN
[1Ø]   [2Ø]   [3Ø]   [4Ø]   1    1Ø    2Ø
[5Ø]   [6Ø]   [3Ø]   [4Ø]   2    2Ø    3Ø
[5Ø]   [6Ø]   [7Ø]
READY
TROFF
READY
```

ERR/ERL VARIABLES

When an error handling subroutine is entered after executing the ON ERROR GOTO statement, the variable ERR contains the error code for the error condition, and the variable ERL contains the line number of the line in which the error was detected. The ERR and ERL variables are usually used in IF . . . THEN statements to direct program flow in the error trap routine.

Usually you would use the following statement in your error trapping routine:

Example:

```
1Ø     ON ERROR GOTO      1ØØØ
⋮
1ØØØ IF ERR = error code     THEN
1Ø1Ø IF ERL = linenumber      THEN
⋮
```

If the statement that caused the error was an immediate mode statement, ERL will contain the value 65535.

Details of error codes are listed in the Appendix.

ERROR integer expression

This statement means different things compared with the ON ERROR GOTO statement mentioned in earlier chapters. This error statement serves two purposes:

1) To similate the occurrence of BASIC error.
2) To allow error codes to be defined by the user.

FUNCTION 1

The value of the integer expression must be between Ø and 255. If the value of this expression matches with an error code already used by the BASIC Interpreter, than the ERROR statement will similate that error and the corresponding error message will be printed.

Example:

```
LIST
1Ø    S = 1Ø
2Ø    T = 5
3Ø    ERROR S + T
4Ø    END
READY
RUN
String too long in line 3Ø
READY
```

FUNCTION 2

To define your own error code, use a value that is greater than any of those used by the BASIC Interpreter. This user-defined error code may then be conveniently handled in an error trap routine.

Example:

```
11Ø  ON ERROR GOTO 4ØØ
12Ø  INPUT "WHAT IS YOUR BET" ;B
13Ø  IF B > 5ØØØØ THEN ERROR 21Ø
 .
 .
 .
4ØØ  IF ERR = 21Ø THEN PRINT "HOUSE  LIMIT  IS
     $5ØØØØ"
41Ø  IF ERL = 13Ø THE RESUME 12Ø
 .
 .
```

In this example, error code 21Ø is newly defined by the programmer to trap the condition of variable B when it is greater than 5ØØØ. If such an error was trapped, the program would resume at line number 12Ø and prompt the user to input a value for B again.

INP (I)

Before going into this function, you will come across the more technical concept of INPUT/OUTPUT ports. These so called 'Ports' are something similar to memory (RAM, ROM etc.) but not exactly the same. You have learned to POKE into and PEEK from memory, and now you can do similar operation on the INPUT/OUTPUT Ports.

The instruction INP (I) returns the value read from INPUT Port I. I must be in the range Ø to 255. This value of I is limited by the Central Processing Unit of your computer.

Example:

```
1ØØ   A = INP (255)
```

The actual value returned will be between Ø and 255.

OUT I, J

OUT is another instruction related to the INPUT/OUTPUT Ports. This sends a byte to a machine output port. I and J are integer expressions in the range Ø to 255. I is the port number and J is the data to be transmitted.

Example:

```
1ØØ   OUT 32, 1ØØ
```

WARNING: As similar to the POKE statement, improper use of the OUT statement can crash your system. So you should understand the internal structure of the computer before you use this statement.

WAIT port number, I, J

This WAIT instruction is used to suspend program execution while monitoring the status of a machine input port.

The bit pattern developed at the INPUT Port is checked to see if it matches a specific value while the program is suspended. The data read at the Port is exclusive ORed (EOR) with the integer expression J, and then ANDed (AND) with value I. If the result is zero, the BASIC program loops back to read the port and check it again. If the result is non zero, program execution continues with the next line number. If J is omitted, it is assumed to be zero.

Example:

```
1ØØ   WAIT 32, 2
```

It may be possible for the WAIT statement execution to cause an infinite loop. In which case it will be necessary to manually reset the machine or power up again.

KEY (for LASER 500, LASER 700 only)

The command KEY, which takes a number of forms is another user-friendly feature of your computer. It affects the ten function keys located on the keyboard of your LASER 500/700. At power up, these function keys are assigned their pre-set values. You may examine, print or change these function key definition with this KEY command.

Format 1:

```
KEY N, STRING
```

This form assigns a string to the function key N.

Example:

```
KEY 3, "LIST"
```

This will assign the string "LIST" to F3, so that, each time you press F3 , 'LIST' will be output to the screen just as if you had typed it from the keyboard.

Although there are only physically 1Ø Function keys found on the keyboard, you have 3Ø different function keys, F1–F3Ø, available. Function key numbers 11–2Ø are generated by pressing the Function key labelled F1 – F10 together with the SHIFT key. Function key number 21–3Ø are generated by pressing the function key together with the CTRL key.

Moreover you may add the "Carriage Return" code as part of your function key.

Example:

```
KEY 18, "CLS : LIST" + CHR$(13)
```

Where CHR$(13) is the ASCII code for "Carriage Return"

After executing the above line, pressing F8 while holding the SHIFT key down will clear the display screen and list your BASIC program.

You should note that if "+ CHR$(13)" was omitted in the above example, then pressing SHIFT F8 will not clear the screen and list the program until the RETURN key is pressed.

Format 2:

```
KEY LIST
```

This will list all the definitions of Function keys F1–F3Ø on the screen.

Format 3:

```
KEY LLIST
```

This will print out all the definitions of Function keys F1–F3Ø. You should have installed your printer before using this command.

At power up, Function keys F1–F1Ø are pre-defined as follows:

F1	Clear the screen and list the program
F2	RUN the BASIC program
F3	Set 40 column text mode
F4	Set 80 column text mode
F5	Set white Foreground and black Background
F6	Set blue Foreground, light blue Background and blue Backdrop
F7	Set normal display mode
F8	Set inverse display mode
F9	Set keyboard silent
F1Ø	Set keyboard beep

Format 4:

```
KEY B
```

This will set the keyboard to beep on every key you press. This is also the power up pre-set condition.

Format 5:

```
KEY S
```

This is complementary to the KEY B function. It sets the keyboard to silent running, i.e., no beep for any key you press.

Format 6:

```
KEY  D, n
```

This will preset the auto-repeating key delay, i.e., the amount of time that you hold a key down before it begins auto-repeating. The value of n is between 1 and 255 inclusive. The actual amount of delay time is n X 20ms. (1ms=one thousandth of a second)

Example:

```
KEYD, 2ØØ
```

This results in a 4 second delay before auto-repeat starts.

Format 7:

```
KEY P, n
```

This presets the auto-repeat period, i.e., the interval between which two codes are generated after holding down a key for enough time. Value of n is between 1 and 255 inclusive. The actual auto-repeat period is n X 20ms. (1 ms – one thousandth of a second)

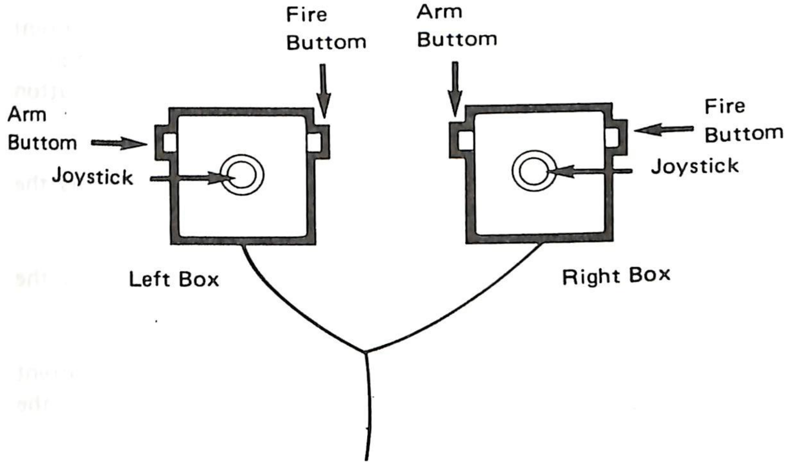
Example:

```
KEY P, 5
```

This results in 100ms (-0.1 second) period for auto-repeating. In other words, when a key is pressed down for enough time, ten codes would be generated each second.

JOY

The JOY function is valid only when you have installed an optional Joystick Expansion Module. This hardware accessory consist 2 boxes. Each box is equipped with one movable "stick" at the top and two push bottoms, one on each side of the box as shown belows:

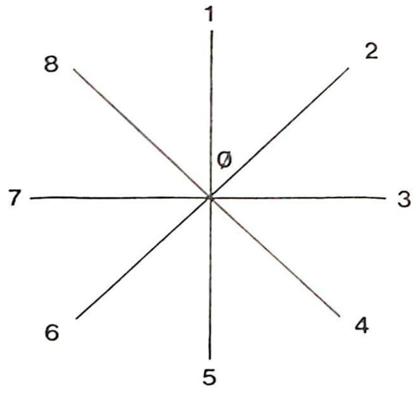


Format of JOY Function is : JOY (X)

Where X is integer between Ø and 5 inclusive.

JOY (Ø) returns an integer value from Ø to 8 which represents the current position of the left Joystick.

The following diagram indicates the relationship between the Joystick position and the JOY (Ø) value.



The centre position is assigned a value of Ø.

JOY (1) returns a binary value (Ø or 1), according to the current status of the left 'Arm Button', i.e. the arm button on the left box. Pressing the button results in a value of 1, and releasing it results in a value of Ø. This rule also applies to other functions related to the button.

JOY (2) returns a binary value (Ø ro 1) to reflect the current status of the left "Fire Button". i.e., the fire button on the left box. The value returned is 1 if the button is pressed, and Ø if the button is released.

JOY (3) returns an integer value (Ø to 8) which represents the current position of the right Joystick.

Notation and meaning of the returned integer is the same as in JOY (Ø).

JOY (4) returns a binary value (Ø or 1) to reflect the current status of the right arm button. The value is 1 if the button is pressed, and Ø if the button is released.

JOY (5) returns a binary value (Ø or 1) to reflect the current status of the right fire button. The value is 1 if the button is pressed, and Ø if the button is released.

Example:

```
1Ø  CLS
2Ø  A = JOY (Ø) : B = JOY (1):C = JOY (2)
3Ø  D = JOY (3) : E = JOY (4):F = JOY (5)
4Ø  PRINT A, B, C, D, E, F
5Ø  GOTO 2Ø
RUN
```

Move the Joystick around, press the buttons and see what happens to the numbers shown on the screen.

Press CTRL C to stop.

CHAPTER 21

SPECIAL DATA TYPE & CONVERSION BETWEEN TYPES

- DATA TYPES
- DEF INT/SNG/DBL/STR
- CDBL
- CINT
- CSNG
- CVI/CVS/CVD
- MKI$/MKS$/MKD$

DATA TYPE (This Chapter is for Advanced BASIC Programmers)

In.the previous chapters we have been manipulating numerical constant, variables, integers, floating points numbers, strings and arrays without rigorous understanding of data type. You might not have aware that your computer has been treating all these matters with a very clear defination of DATA Type.

The information, or data, that your computer's BASIC Interpreter uses can be boardly divided into CONSTANTS and VARIABLES.

Constants

Constants are the actual values which the BASIC Interpreter uses during execution. There are two types: String and Numeric.

String constants are sequences of up to 255 alphanumeric characters enclosed in double quotation marks.

Numeric constants are positive or negative numbers of the following five types:

1. Integer constants – whose numbers between –32768 and +32767.

2. Fixed Point constants – positive or negative real numbers that contain a decimal point.

3. Floating Point Constants – positive or negative numbers represented in exponential form, consisting of a signed fixed point number followed by letter E and a signed integer exponent. The allowable range for a floating point constant is 10^{-38} to 10^{+38} . (Double Precision floating point constants use letter D instead)

4. Hex Constants – hexadecimal numbers with prefix & H.

5. Octal Constants – octal numbers with prefix & O or &.

In general, numeric constants may be either single precision or double precision numbers. With double precision, the numbers are stored with 16 digits of precision, and printed up to 16 digits.

To summarize, a single precision constant is any numeric constant that has
(i) seven or fewer digits, or
(ii) exponential form using E, or
(iii) a trailing exclamation point (!)

A double precision constant is any numeric constant that has:

(i) eight or more digits, or
(ii) exponential form using D, or
(iii) a trailing number sign (#)

VARIABLES

Variables are names used to represent the values used in a BASIC program. These values may be assigned by the programmer, or occur as the result of calculation. Before a variable is assigned a value, its value is assumed to be zero.

A variable name must not be a reserved word, i.e., that of special meaning to the BASIC Interpreter.

For example, 'GOTO' is reserved.

Up to 40 characters can be used for the variable names in this BASIC Interpreter.

Variables may represent either a numeric value, a string or an array. String variable names are written with a dollar sign ($). Numeric variables may be declared as integer, single or double precision by attaching declaration characters as the last character of the variable name.

Example:

LIMIT%	declares an integer variable
MINIMUM!	declares a single precision value
PI #	declares a double precision value

An array variable name has as many subscripts as there are dimensions in the array. The maximum number of dimensions for an array is 255, and the maximum number of elements per dimension is 32767.

The following table summarizes the storage space requirements of the different types of variables:

		BYTES
VARIABLES:	INTEGER	2
	SINGLE PRECISION	4
	DOUBLE PRECISION	8
ARRAYS:	INTEGER	2 per element
	SINGLE PRECISION	4 per element
	DOUBLE PRECISION	8 per element

STRINGS: 3 bytes overhead plus the present contents of the string.

Under certain circumstances, the BASIC Interpreter will convert a constant from one type to another. Moreover, programmers are allowed to perform conversion between data types or declare their constant or variable to be of a particular type. The coming pages will describe the BASIC statements which perform these tasks.

DEF INT/SNG/DBL/STR

There are four statements used to declare variable types: integer, single precision, double precision, or string.

A DEF (type) statement declares that the variable names beginning with the letter (s) specified will be that type of variable.

Example:

```
10 DEFDBL  L–P
```

All variables beginning with the letters L, M, N, O and P will be double precision variables.

```
1Ø DEFSTR  A
```

All variables beginning with the letter A will be string variables.

```
1Ø DEFINT I–N, W–Z
```

All variables beginning with the letters I, J, K, L, M, N, W, X, Y, Z will be integer variables.

If no such type-declaration statements are specified, the BASIC Interpreter assumes that variable names without declaration characters are single precision variables.

Finally, you should be aware that a type declaration character (such as %, !, #) always takes precedence over a DEF (type) statement.

CDBL (X)

This function returns the number X in double precision number form.

Example:

```
1Ø  A = 454.67
2Ø  PRINT A;CDBL(A)
READY
RUN
454.67  454.67ØØ13427344
READY
```

CINT (X)

This function returns X in integer form by rounding the fractional portion. X must be in the range –32768 to +32767 otherwise an "Overflow" error occurs.

Example:

```
1Ø  PRINT CINT (45.67)
READY
RUN
46
READY
```

CSNG (X)

This function returns X in single precision number form.

Example:

```
1Ø  A = 975.3421 #
2Ø  PRINT A # ; CSNG (A #)
READY
RUN
975.3421   975.342
READY
```

NOTE: The CVI, CVS, CVD, MKI$, MKS$ and MKD$ function may seem a little strange. Actually they are intended for use in data storage when the disk operating system is available.

CVI/CVS/CVD (string)

These functions convert string values to numeric values.

Formats:

```
CVI  (2 byte string)
CVS (4 byte string)
CVD (8byte string)
```

CVI converts a 2-byte string to an integer.
CVS converts a 4-byte string to a single precision number.
CVD converts an 8-byte string to a double precision number.

Example:

```
1Ø N$ = "TESTING STATEMENT"
2Ø PRINT CVI (N$)
3Ø PRINT CVS (N$)
4Ø PRINT CVD (N$)
RUN
17748                          (Integer)
4.69115E – 14                  (Single Precision)
9.826534929158341D – 3Ø        (Double Precision)
READY
```

MKI$/MKS$/MKD$ (expression)

These functions convert numeric values to string values.

Formats:

```
MKI$ (integer expression)
MKS$ (single precision expression)
MKD$ (double precision expression)
```

MKI$ converts an integer to a 2-byte string.
MKS$ converts a single precision number to a 4-byte string.
MKD$ converts a double precision number to an 8-byte string.

Example:

```
1Ø  N$ = "LASER 5ØØ"
2Ø  PRINT CVI (N$) : A = CVI (N$)
3Ø  PRINT CVS (N$) : B = CVS (N$)
4Ø  PRINT CVD (N$) : C = CVD (N$)
5Ø  PRINT MKI$ (A) : PRINT MKS$ (B) : PRINT
    MKD$ (C #)
RUN
16716                           (Integer)
1.43152E–18                     (Single Precision)
5.8525Ø495516296D–25            (Double Precision)
LA
LASE
LASER 5Ø
READY
```

ASSEMB

Assemb
compu
Interpre
Languag

This ch
ambitio
compu

Assem
structio
you to
astrou

DEF U

This co
languag

digit
the U
DEF U

The v
routin

You a
BASIC
these

Exampl

CHAPTER 22

USING ASSEMBLY LANGUAGE SUBROUTINE

- DEF USR
- USR
- CALL
- VARPTR

ASSEMBLY LANGUAGE SUBROUTINE

Assembly Language instructions are the lowest level program that your computer can run. In fact, your BASIC program, and the BASIC Interpreter resident in your computer, are resting upon Assembly Language programs designed by our software engineers.

This chapter is written for the advanced BASIC programmers and those ambitious readers who would like to probe into the hidden side of computing beyond BASIC programs.

Assembly Language subroutines could be made by POKEing instruction into memory locations or taken from tape. It is advised that you take great care when using Assembly Language as it can have disastrous effects on stored programs.

DEF USR digit = integer expression

This statement is used to specify the starting address of an assembly language subroutine to be called later by the user.

'digit' may be any digit from Ø to 9. It corresponds to the number of the USR routine whose address is being specified. If 'digit' is omitted, DEF USRØ is assumed.

The value of the integer expression is the starting address of the USR routine.

You are free to use any number of DEF USR statements in your BASIC program to redefine subroutine starting address and to access these subroutines.

Example:

```
•
•
•
2ØØ DEF USRØ = 24ØØØ
21Ø X = USRØ (Y ↑ Z/2.89)
•
•
•
```

USR digit (X)

This statement calls the user's Assembly Language subroutine with argument X.
'digit' is in the range Ø to 9 and corresponds to the digit supplied with the DEF USR statement for that routine.
USRØ is assumed if 'digit' is omitted.

Example:

```
40  B = T * SIN (Y)
50  C = USR (B/2)
6Ø  D = USR (B/3)
 .
 .
 .
```

When the USR function call is made, register A contains a value that specifies the type of argument that was given. The value in A may be one of the following:

Value in A	**Type of Argument**
2	Two-byte integer (two's complement)
3	String
4	Single precision floating point number
8	Double precision floating point number

If the argument is a number, the [H, L] register pair points to the Floating Point Accumulator (FAC) where the argument is stored.

If the argument is an integer:
FAC–3 contains the lower 8 bits of the argument and
FAC–2 contains the upper 8 bits of the argument.
If the argument is a single precision floating point number:
FAC–3 contains the lowest 8 bits of mantissa and
FAC–2 contains the middle 8 bits of mantissa and
FAC–1 contains the highest 7 bits of mantissa with leading 1 suppressed (implied). Bit 7 is the sign of the number (0=positive, 1–negative)
FAC is the exponent minus 128, and the binary point is to the left of the most significant bit of the mantissa.

If the argument is a double precision floating point number:

FAC–7 throughFAC–4 contain four more bytes of mantissa (FAC–7 contains the lowest 8 bits).

If the argument is a string, the [D, E] register pair points to 3 bytes called the "string descriptor." Byte Ø of the string descriptor contains the length of the string (Ø to 255). Bytes 1 and 2, respectively, are the lower and upper 8 bits of the string starting address in string space.

CALL variable name (argument list)

The CALL statement is another way to transfer program flow to an external subroutine.

"variable name" contains an address that is the starting point in memory of the subroutine. It should not be an array variable name.

"argument list" contains the arguments that are passed to the external subroutine. It should contain only variables.

Example:

```
11Ø MYROUT = & HD ØØØ
12Ø CALL MYROUT (I, J, K)
:
```

The program assigns hexadecimal DØØØ to the variable name MYROUT and calls the subroutine at hexadecimal DØØØ, passing the variable I, J, K to it.

VARPTR (variable name)

This BASIC function returns the Memory Address of the first byte of data identified with 'variable name'. A suitable value must have been assigned to 'variable name. prior to the execution of VARPTR.

Any type of variable name may be used (numeric, string, array), and the address returned will be an integer in the range 32767 to –32768. If a negative address is returned, add it to 65535 to obtain the actual address.

VARPTR is usually used to obtain the address of a variable or array so it may be passed to an assembly language subroutine.

Example:

```
1ØØ X = USR (VARPTR (Y))
```

CHAPTER 23

HINTS ON MORE EFFECTIVE PROGRAMMING

There are several methods that you can use to improve your program. A program is effective means that this program occupies less memory space or executes much faster. The methods are explained as follows:

1) Use less LET
   The use of the command LET is optional for this computer. You may omit the LET command to save memory space for your program.

2) Use less REM
   REM command will allow you add remark to your program but it also occupies memory space. You may delete all unnecessary REM statements in your program.

3) Use Multiple-statements
   The use of multiple-statements will save memory space and also speed up your program.

4) Use integer variables
   Integer variables occupy less space than real variables. Whenever possible, you may substitute the real variables A, for example, by the integer variable A%.

5) Use subroutines
   You may use subroutines to save program space if the operation is called from different places on your program several times.

6) Use less parentheses
   You may eliminate the use of parentheses whenever possible to save memory space.

7) Use simple expression in functions.
   You may get the result from a function much faster if you simplify the argument of that function.

Example:

```
1Ø  A = 3^2 + 17 – 22/3 : A = INT (A)
```

will execute much faster than

Example:

```
1Ø  A = INT (3^2 + 17 – 22/3)
```

but occupy more space.

8) Estimate the size of array
The use of DIM occupies much memory space. Therefore it is better to assign just enough size for your array. Also use the zero subscripted elements as they are always available.

9) Use READ and DATA
Often you have to use the same command many times but with different parameters. In order to save space and repetitive typing, you may put these parameters into DATA statements and retrieve them by READ statements.

PART III

TECHNICAL REFERENCE MANUAL

INTRODUCTION

This part of the Manual covers the technical information of the three models LASER 350/LASER 500/LASER 700. Advanced users will find this part useful for probing into the internal workings of the computer. Beginners may skip this part.

The overall Memory Organization will be presented first. This includes the Bank — Switching technique used in this computer, Physical/ Logical Address Space and how to configure Memory Banks in user programs. Then, more detailed description of Memory Usage under the title of RAM, ROM and Memory mapped I/O will be given. I/O Ports usage summarizes the necessary information for controlling your computer via software.

The second chapter discusses the Display Modes available to your computer, the screen map and the color information of each display mode.

The third Chapter is dedicated to hardware details of the computer. The overall hardware structure is presented by functional block diagrams and further explained with simplified schematics. It is hoped that programmers can make use of these simplified functional blocks, together with the detailed circuit schematic diagrams in the Appendix, to understand the internal structure of this computer.

The fourth chapter discusses the software aspects of your computer. The power up sequence for the ROM cartridge and the automatic bootstrapping of the Floppy Disk are of interest to the advanced user, in particular, for designing their own ROM cartridges. Some useful Assembly Language entry points for user calling are also listed.

The last chapter describes the built-in System Monitor features. Each Monitor Command is explained in detail. These Monitor Command would be most valuable to Assembly Language Programmer who is debugging at the machine language level.

(Please note that in the technical reference manual hexadecimal numbers are used to represent memory address and data for convenience).

CHAPTER 1

MEMORY ORGANISATION

- BANK-SWITCHING
- MEMORY USAGE
  - ROM MEMORY
  - RAM MEMORY
  - MEMORY MAPPED I/O
- I/O PORTS USAGE
  - PORT 44H
  - PORT 45H

The heart of your computer is the Z-80A microprocessor referred to hereafter as the CPU (Central Processing Unit). The Z-80 microprocessor can directly reference a total of 65536 different memory locations by means of its 16 address lines. In computer terminology, the address space is 64K (1K = 1024 locations).

The VLSI Gate Array inside your computer further enhances the addressing capability to 256K (262144 locations) via a Windowing or Bank-Switching technique. The 256K memory address will be refered to as the Physical Address Space while the original 64K available to the Z-80 microprocessor will be refered to as the Logical Address Space.

The concept of Physical Address Space and Logical Address Space is illustrated is Fig. 1—1.

LASER 350 / LASER 500 / LASER 700
Memory organization

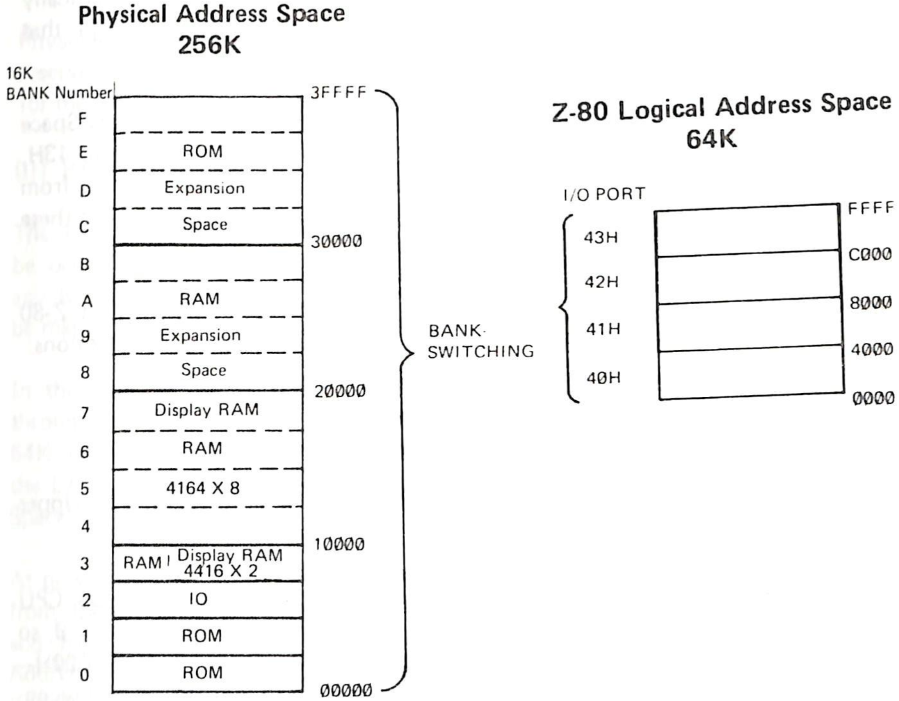


Fig. 1—1 Memory Organization LASER 350 / 500 / 700

BANK-SWITCHING

For convenience in discussion, the 64K logical Address Space of the Z-80 can be divided into four distinct areas each of 16 Kbytes in length. On the other hand, the 256K Physical Address Space of your computer is also divided into sixteen memory banks, each bank 16K bytes in length. These banks are numbered from Ø, 1, 2 ..... to FH, which corresponds to Physical Address ØØØØØH to 3FFFFH.
For example, bank Ø covers the area ØØØØØH to Ø3FFFH while bank 4 covers the area 1ØØØØH to 13FFFH.

The term Bank-Switching or Windowing refers to the process of mapping the 16 banks (Ø, 1, 2 .....FH) in the Physical Address Space into the 4 distinct areas of the Z-80 logical Address Space. To make things simple, the Z-80 CPU may be thought of as having four "SOCKET" in its 64K memory address space. Each "SOCKET" accepts one 16K Byte bank of memory. Any one 16K bank is available to the Z-80 CPU simply by "plugging" in the "SOCKET" with the bank number. Incidentally, a "SOCKET" being "plugged" in with a new bank number would imply that the old bank number is automatically "unplugged" and hence, not available to the CPU through that "SOCKET".

The actual bank configuration within the Z-80 logical Address Space is done by writing into hardware I/O Ports 4ØH, 41H, 42H and 43H, which determines the Bank assignments of the Z-80 CPU counting from lowest memory to highest memory. It is meaningless to read from these Ports because of their "write-only" nature.

For example, to assign Bank Ø to address ØØØØH–3FFFH in Z-80 logical Address Space, use the following machine language instructions:

```
LD A, ØØH
OUT (4ØH), A
```

Only the lower 4 bits of the data sent out are significant. The Upper 4 bits are neglected.

At power up, the Memory Bank configuration in the Z-80 CPU Address Space is forced by hardware to be all mapped to Bank Ø, so that the resident ROM is always enabled in logical Address ØØØØH–3FFFH. The CPU will then perform power up initialization and reconfigure these Memory Banks.

MEMORY USAGE

The LASER 350/LASER 500/LASER 700 computers are equipped with the same VLSI Gate Array Chip that handles the 256K Physical Address and their Bankswitching circuitry.

(I) ROM MEMORY:

All three models (LASER 350/LASER 500/LASER 700) have a built-in 32K bytes system ROM BASIC Interpreter. They are mapped into BANK Ø and BANK 1 of the 256K Physical Address Space. These two 16K banks Ø, 1 are in turn mapped to logical address ØØØØH–3FFFH and 4ØØØH–7FFFH respectively when BASIC is running. However, bank1 might be temporarily switched out or "unplugged" from 4ØØØH–7FFFH for writting into display RAM area or reading from the built-in memory mapped I/O area such as keyboard polling.

BASIC programmers should regard the Z-80 logical Address ØØØØH–7FFFH as ROM area. Therefore, they should bot POKE data into these addresses because ROM content cannot be changed by POKEing.

Physical Address BANKS 0CH through 0FH (totally 64K bytes) are reserved for ROM Expansion. Refer to the chapter on Software Aspects for more information about ROM cartridge design.

(II) RAM Memory

The 16K on-board RAM of the LASER 350 is defined by hardware to be occupying BANK3 of the 256K Physical Address Space. Without any RAM expansion, this one and only one 16K bank of RAM would be mapped into the Z-80 Logical Address Space 8ØØØH–BFFFH.

In the case of the LASER 500 without RAM Expansion, BANK 4 through BANK 7 of the 256K address space would be defined as the 64K on-board RAM. In fact, the 64K RAM Expansion Module of the LASER 350 also occupies these areas in the 256K Physical Address Space.

At power up, the Z-80 CPU checks for the presence of the 64K RAM from BANK 4 to BANK 7. If it is present, then Physical BANK 4 and BANK 5. would be mapped, or "plugged" into Z-80 Logical Address 8000H–BFFFH, and C000H–FFFFH respectively. Your BASIC program, data variables would be residing in these areas too.

Similary for the LASER 700 (64K RAM or 128K RAM built-in) and the LASER 350 with 64K or 128K RAM expansion, Physical BANK 4, 5 would be mapped into 8ØØØH–FFFFH in the Z-80 Logical Address Space. In fact, your computer's BASIC Interpreter does not distinguish between a LASER 350 with 64K RAM Expansion, or LASER 500, or a LASER 700 as far as software is concerned. The BASIC Interpreter will not utilize all of that "unplugged" RAM memory, Users would have to write a special program in order to access this RAM.

Physical Address BANK 3 and BANK 7 are of special meaning to your computer. They are defined via hardware to be the display RAM area. The 40 and 80 column text screen, graphic mode screens GR0 through GR 5 are using these RAM areas for memory mapped display. To change the display (text or graphics), simply write to these RAM locations. You are allowed to choose between BANK 3 or BANK 7 as display RAM (either BANK 3 or BANK 7). would be termporarily "plugged" into the Z-80 Logical Address Space, which "unpluggs" the BANK 1 ROM for a short while. This sometimes results in the same BANK being mapped into two different Logical Address areas at the same time. Nevertheless, this is allowed and is necessary for the LASER 350 with only 16K RAM.

The current version of BASIC Interpreter uses part of BANK 6 for the user-defined Function Key Definition text data. So you should be aware of this if your program is using BANK 6 for data storage.

If you have LASER 350 with 128K RAM Expansion, or LASER 500 with 64K RAM Expansion, or a 128K RAM LASER 700, or 64K RAM LASER 700 with 64K RAM Expansion, then BANK 8H through BANK BH are available to you as user RAM. You may perform BANK-SWITCHING in your program and use these areas for data storage.

(III) Memory Mapped I/O

BANK 2 of the 256K Memory Address Space in LASER 350/500/700 is reserved for on-board Memory Mapped I/O. The physical location ØA8ØØH–ØAFFFH of BANK 2 (totally 2K Bytes decoded address) is used in two different ways.

Firstly, the keyboard is scanned by reading these locations. The keyboard's keys are arranged in a matrix with a number of rows and columns. Each row of the matrix is mapped into a memory location. Each column is mapped into the corresponding bit position of that memory location. The key are scanned by an active low signal. If a key is pressed, the corresponding bit of the corresponding location will be reset. The actual locations corresponding to the 12 rows are shown as follows:

Address Read	Key Matrix row line
AFFEH	AØ
AFFDH	A1
AFFBH	A2
AFF7H	A3
AFEFH	A4
AFDFH	A5
AFBFH	A6
AF7FH	A7
A8FFH	A
A9FFH	B
AAFFH	C
ABFFH	D

For details of the keyboard matrix structure, refer to the correct section of keyboard.

The lines D, C, B, A are decoded outputs from a TTL decoder LS138. These four lines are only present in the LASER 500/700. They are not implemented on the LASER 350.

Secondly, when written, the entire area would perform output functions on each bit as shown in the table below:

Bit number	I/O Function	Remarks
Ø	Buzzer	Toggling bit Ø rings the Buzzer
2	Cassette Output	
3	Text/Graphic Mode Select	'1' for Graphics Mode 'Ø' for Text Mode
6	CAP LOCK Indicator	'1' for LED off 'Ø' for LED on

Table 1–1 I/O Area Functions

The following diagram summarizes the memory bank configuration used in LASER 350/LASER 500/LASER 700 and the presence of RAM Memory Expansion.

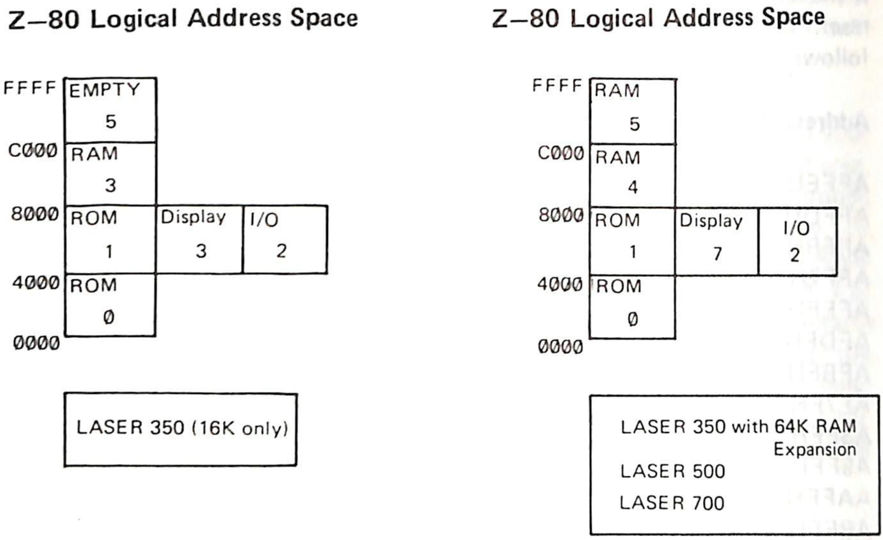


Fig. 1–2 LASER 350 / 500 / 700
Memory Bank Configuration

Three BANKS are placed side by side at Z-80 Logical Address Space 4ØØØH–7FFFH, indicating that these BANKS are temporarily "plugged" in and later "unplugged" from that "SOCKET".

The notation used in this figure is to place the BANK number (Ø–FH) in the centre of the rectangle which represents the 16K "SOCKET" in the Z-80 Logical Address Space. The smaller word at the left hand top corner of the rectangle describes the kind of memory, I/O, or whatsoever mapped from the 256K Physical Address Space.

You might have noticed that Z-80 Logical Address CØØØH–FFFFH in the LASER 350 (16K RAM only) is mapped to BANK 5 but labelled "EMPTY".Therefore reading from, or writting to CØØØH–FFFFH are meaningless exercise, because BANK 5 is not occupied by any physical device.

I/O PORTS USAGE

Apart from Memory Mapped I/O as described in the previous pages, the Z-80 CPU of your computer is capable of performing I/O Mapped I/O via a I/O REQUEST (IORQ) line. A maximum of 256 different I/O Ports could be defined by hardware circuitry. The VLSI Gate Array of the LASER 350/500/700 has already implemented this for you, with some I/O Ports well defined for special functions which controls the operation of your computer.

You should have learned that I/O Ports 4ØH, 41H, 42H, 43H define the Memory Bank Configuration in the Z-80 Logical Address Space. Besides these four, I/O Port 44H and 45H are of special interest to you. Port 44H controls the Screen Display Mode while Port 45H controls the color of some display modes.

(I) Port 44H

The following table summarizes the relationship between the bit pattern written to this port and the display mode invoked. Please note that these are 'write only' Ports and any data read from them are meaningless. Moreover, text mode or graphic mode is determined by bit 3 of I/O location ØA8ØØH–ØAFFFH (BANK 3 of 256K Physical Address Space).

Text Mode (Bit 3 of I/O area = Ø)

Port 44 :	bit 3	bit 2	bit 1	bit 0	Mode	Display RAM at
	Ø	X	X	Ø	40 column X 24 row	BANK 7 1F8ØØH 1FFFFH
	1	X	X	Ø	40 column X 24 row	BANK 3 0F8ØØH ØFFFFH
	Ø	X	X	1	80 column X24 rows	BANK 7 1F8ØØH 1FFFFH
	1	X	X	1	80 column X 24 row	BANK 3 ØF8ØØH ØFFFFH

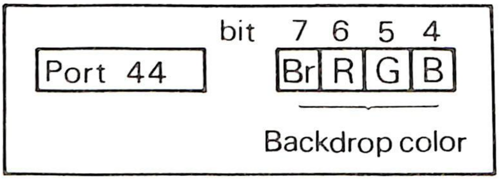
X = don't care condition

Graphic Mode (Bit 3 of I/O area =1)

Port 44 :	bit 3	bit 2	bit 1	bit Ø	Mode	Display RAM at
	Ø	Ø	Ø	X	GR5	BANK 7 1CØØØ–1FFFF
	1	Ø	Ø	X	GR5	BANK 3 ØCØØØ–ØFFFF
	Ø	Ø	1	Ø	GR4	BANK 7 1CØØØ–1FFFF
	1	Ø	1	Ø	GR4	BANK 3 ØCØØØ–ØFFFF
	Ø	Ø	1	1	GR3	BANK 7 1CØØØ–1FFFF
	1	Ø	1	1	GR3	BANK 3 ØCØØØ–ØFFFF
	Ø	1	Ø	X	GRØ	BANK 7 1EØØØ–1FFFF
	1	1	Ø	X	GRØ	BANK 3 ØEØØØ–ØFFFF
	Ø	1	1	Ø	GR2	BANK 7 1EØØØ–1FFFF
	1	1	1	Ø	GR2	BANK 3 ØEØØØ–ØFFFF
	Ø	1	1	1	GR1	BANK 7 1EØØØ–1FFFF
	1	1	1	1	GR1	BANK 3 ØEØØØ–ØFFFF

You might have noticed that bit 3 of Port 44H determines either BANK 7 or BANK 3 for use as the Display RAM, i.e., if Bit 3=Ø, Display RAM is at BANK 7, otherwise at BANK 3. An 'X' in the table entry indicates a don't care condition. Either a logical '1' or 'Ø' written to these bit positions will not affect the result.

Bit 4 through Bit 7 of Port 44H determines the Backdrop color. The format is as follows:



The bit labelled 'Br' refers to the brightness of that color, while the remaining 3 bits labelled 'R', 'G' and 'B' combine to form the 'Hue' of the color.

The combinations of Br, R, G, B to form 16 different colors is duplicated here for your convenience.

Table 1–2 Color Code Table

Br	R	G	B	Color	Br	R	G	B	Color
0	0	0	0	Black	1	0	0	0	Light Grey
0	0	0	1	Blue	1	0	0	1	Light Blue
0	0	1	0	Green	1	0	1	0	Light Green
0	0	1	1	Cyan	1	0	1	1	Light Cyan
0	1	0	0	Red	1	1	0	0	Light Red
0	1	0	1	Magenta	1	1	0	1	Light Magenta
0	1	1	0	Yellow	1	1	1	0	Light Yellow
0	1	1	1	Grey	1	1	1	1	White

Please note that the above Color Code Table also applys to other graphics/text modes which use the "Br RGB" 4 bit combinations.

(II) Port 45H

The bit pattern written to Port 45 affects the Foreground/Background color in some of the display modes including text modes and graphics modes. For details of the Foreground/Background colors in each display mode please refer to the Chapter on graphics commands in the BASIC Programming Manual (PART II).

The following Display Modes will have their Foreground Background color controlled by Port 45H.

(i) 80 column X 24 row text mode
(ii) 640H X 192V graphics mode GR5
(iii) 320H X 192V graphics mode GR2

Figure 1–3 shows the format of I/O Port 45H and the Foreground/Background color of the above three display modes.

bit	7	6	5	4	3	2	1	Ø
Port 45H	Br	R	G	B	Br	R	G	B
	Foreground color				Background color			

Fig. 1–3 Format of Port 45 H and color

As shown above, the upper 4 bits determine the Foreground color while the lower 4 bits determine the Background color.

The color code formed by 'Br' 'R' 'G' 'B' is the same as that used in bit 4–bit 7 of Port 44H (Backdrop color).

CHAPTER 2

DISPLAY MODES

- TEXT MODE
- TEXT MODE SCREEN MAP
- GRAPHICS MODE
- GRAPHICS MODE SCREEN MAP

DISPLAY MODES

Two text modes and six graphic modes are available for the LASER 350/LASER 500/LASER 700. However, LASER 350 users will notbe able to invoke some graphics modes (GR 3, GR 4, GR 5) unless they have a 64K RAM Expansion Module installed.

The following table summarizes the display modes:

Text Mode

Mode	Column X Row	Color
40 column	40 X 24	16 colors
80 column	80 X 24	2 out of 16 colors

GRAPHICS MODE

Mode	Resolution	Color	Display Memory Used	Remarks
GRØ	160 X 96	16	8K	For LASER 350/500/700
GR1	160 X 192	16 (2 out of 16)	8K	For LASER 350/500/700
GR2	320 X 192	2	8K	For LASER 350/500/700
GR 3	160 X 192	16	16K	For LASER 350 with 64K RAM/LASER 500/700
GR4	320 X 192	16 (2 out of 16)	16K	For LASER 350 with 64K RAM/LASER 500/700
GR5	640 X 192	2	16K	For LASER 350 with 64K RAM/LASER 500/700

TEXT MODES

In both 40 column and 80 column display, your computer can display 24 rows of alphanumeric characters, special symbols and graphic characters.
Each alphanumeric character or symbol is comprised of a character font of a 5 dots wide by 7 dots high matrix in a 8 X 8 dots grid. A two dots wide space on the left and one dot wide space on the right is used to keep adjacent words apart. Also a one dot wide space at bottom of each character is used to keep each line apart from characters on a lower row. Some graphic characters may use the entire 8 X 8 matrix with no space left between the adjacent characters.

Upper and lower case characters for alphabets are available in both the 40 and 80 column display.

The inverse display is supported in both 40 and 80 columns, but the hardware circuit does not support a flashing character display. Any character position, including the CURSOR position desired to produce a flashing effect must be done via software, which writes a normal character and an inverse character to that RAM location. RAM locations with MSB set will be displayed as INVERSE characters while those with MSB reset will be displayed as NORMAL characters.

When in multicolor text display mode, the term "Inverse Characters" refers to the active dot matrix displayed in the Background color and the remaining area of the character displayed in the Foreground color. Please refer to the graphics mode command in the BASIC Programming Manual for exact definition of Foreground/Background color.

In 40 column text display, each character has its own Foreground/Background color. Each color is programmable up to 16 different colors. The Backdrop color is also programmable to one of 16 different colors.

For selection of the Backdrop color, please refer to the section on I/O Ports Usage, in particular Port 44H.

In 80 column text display, the active display area of the entire screen can be programmed to 2 out of 16 possible colors which are the Foreground/Background colors for the whole screen.
For selection of Foreground/Background colors, please refer to the section on I/O Ports usage, in particular, Port 45H.

Table 2—1 shows the ASCII Screen Chracters.

Table 2–1 ASCII Screen Characters

		Normal Characters								Inverse Characters							
		Graphic		Alphanumeric						Graphic		Alphanumeric					
Decimal		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
	Hex	$00	$10	$20	$30	$40	$50	$60	$70	$80	$90	$A0	$B0	$C0	$D0	$E0	$F0
0	$0	◲	α		0	@	P	`	p	◲	α		0	@	P	`	p
1	$1	⊤	▖	!	1	A	Q	a	q	⊤	▖	!	1	A	Q	a	q
2	$2	◱	▗	"	2	B	R	b	r	◱	▗	"	2	B	R	b	r
3	$3	⊢	▄	#	3	C	S	c	s	⊢	▄	#	3	C	S	c	s
4	$4	⊞	▝	$	4	D	T	d	t	⊞	▝	$	4	D	T	d	t
5	$5	⊣	▐	%	5	E	U	e	u	⊣	▐	%	5	E	U	e	u
6	$6	◳	▞	&	6	F	V	f	v	◳	▞	&	6	F	V	f	v
7	$7	⊥	▟	'	7	G	W	g	w	⊥	▟	'	7	G	W	g	w
8	$8	◰	▘	(	8	H	X	h	x	◰	▘	(	8	H	X	h	x
9	$9	♠	▚	)	9	I	Y	i	y	♠	▚	)	9	I	Y	i	y
10	$A	♥	▌	*	:	J	Z	j	z	♥	▌	*	:	J	Z	j	z
11	$B	♣	▙	+	;	K	[	k	{	♣	▙	+	;	K	[	k	{
12	$C	♦	▀	,	<	L	\	l	¦	♦	▀	,	<	L	\	l	¦
13	$D	π	▜	—	=	M	]	m	}	π	▜	—	=	M	]	m	}
14	$E	μ	▛	.	>	N	∧	n	~	μ	▛	.	>	N	∧	n	~
15	$F	£	█	/	?	O	—	o	■	£	█	/	?	O	—	0	■

Fig. 2—2 shows all the alphanumeric character symbols and graphic characters available in normal display.

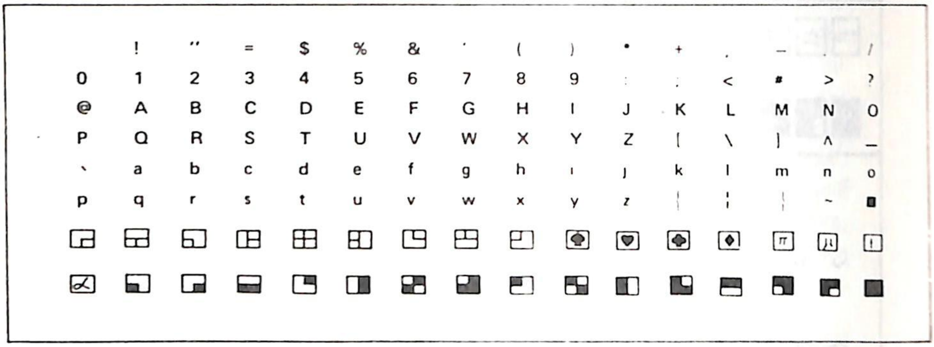


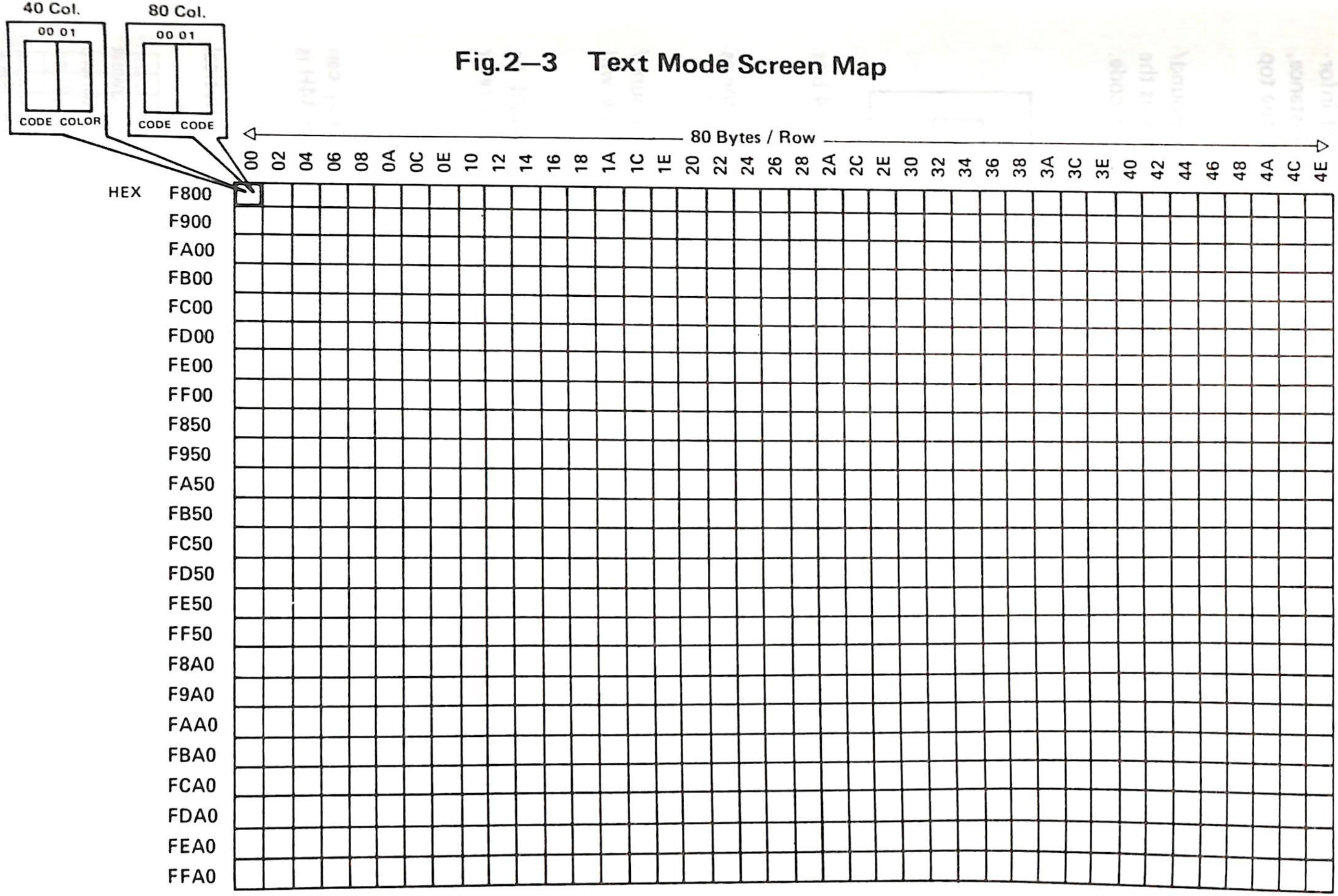
Fig 2—2 Text mode normal character set

TEXT MODE SCREEN MAP

The RAM locations used for text mode display can be chosen from two different areas on the 256K Physical Address Space. If bit 3 of the byte written to I/O Port 44H is a 'Ø', Display RAM will be at BANK 7. Otherwise, the display RAM will be at BANK 3. Please refer to the section on I/O Port usage for details of these display area selections. The same rule applies to the graphics modes.

For clarity, the most significant hexadecimal digit of the Physical Address used on the screen map is omitted.

Fig.2—3 Text Mode Screen Map



Each row of characters on the text screen requires 80 bytes of information regardless of being 40 column or 80 columns. For instance, F8ØØH–F84FH (totally 80 bytes) is responsible for the text on the top row of the screen.

For 40 column text display, which has its own set of Foreground/Background colors in each character, even byte on the row contains the character code and the odd byte that follows contains the color code. The format is as follows:

RAM	bit	7	6	5	4	3	2	1	0
Odd byte =		Br	R	G	B	Br	R	G	B
		Foreground				Background			

The upper 4 bits determine the Foreground color while the lower 4 bits determine the Background color, for that one character only.

The combination of Br, R, G, B to form 16 different colors is same as that described in Chapter 1.

For 80 column text display, which only has one set of Foreground/Background colors for the entire screen, each byte on the row will represent the character code.

The actual Foreground/Background color for that screen would be determined by the bit pattern written to Port 45H. For details refer to the section on I/O Port usage.

GRAPHIC MODE SCREEN MAP

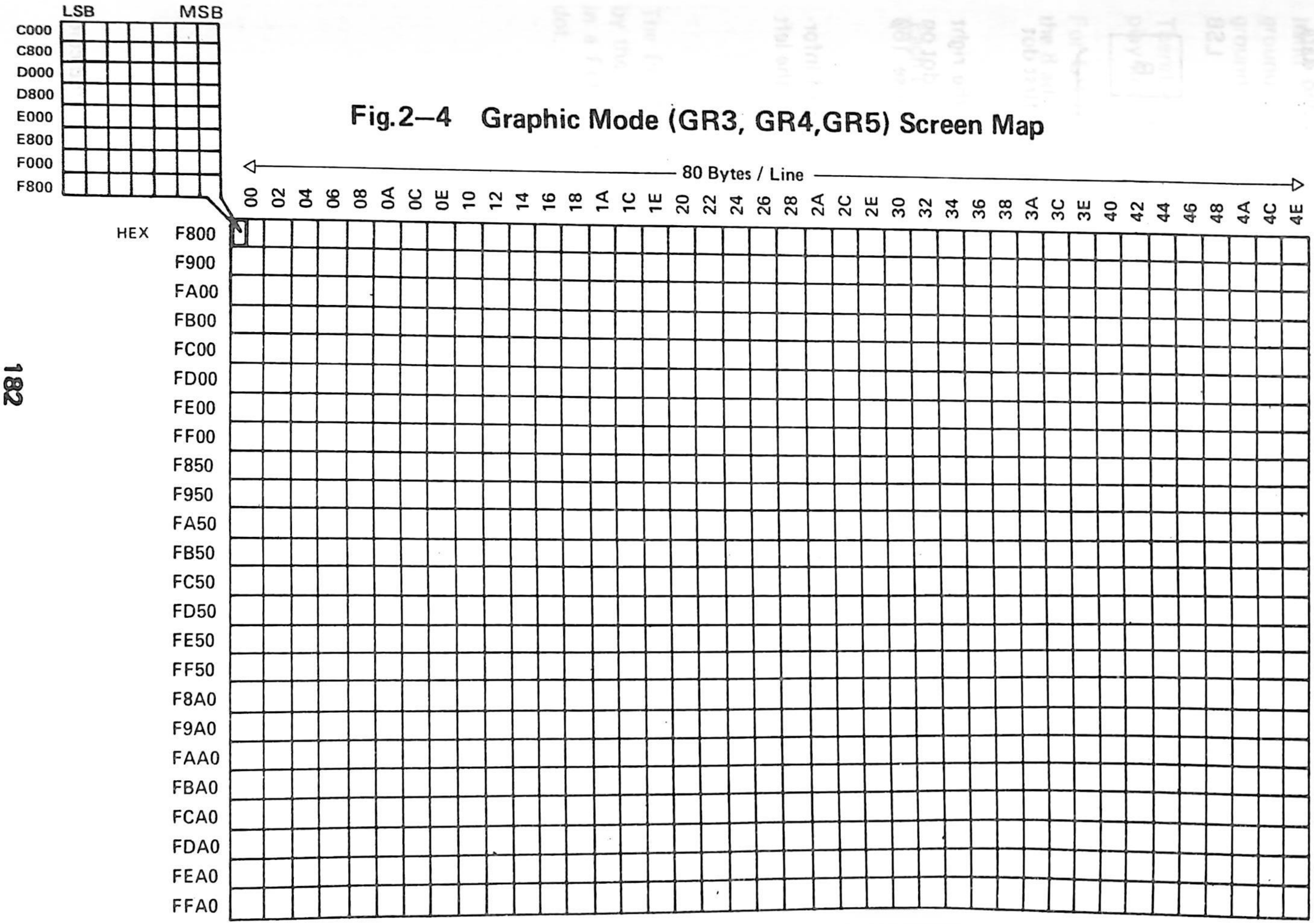
As in text mode display, RAM area for the graphics mode display can be selectable on the 256K Physical Address Space. Bit 3 of Port 44H is chosen for such purpose.

For clarity, the most significant hexadecimal digit of the Physical Address used on the screen map is omitted.

In all graphics modes, the LSB, or the dot represented by the lowest bit, is shifted out first, so that the corresponding dot will be shown on the left hand side.

Fig. 2–4 is the screen map for the 16K graphics modes (GR3, GR4, GR5)

Fig.2—4 Graphic Mode (GR3, GR4,GR5) Screen Map



Each display line is associated with 8Ø bytes in the Display RAM. These RAM data are interpreted differently in each graphics mode.

For GR3 (160H X 192V, 16 colors), each byte of fetched video data represents the color of two adjacent dots, as follows:

1 byte of video data in GR 3

MSB							LSB
Br	R	G	B	Br	R	G	B
Color code of second dot				Color code of first dot			

The upper 4 bits gives the color of second dot, i.e., dot on the right hand side, and the lower 4 bits gives the color of first dot, i.e., dot on the left hand side. Therefore, a total of 8Ø bytes let you use 16Ø different dots on one display line.

For GR4 (320H X 192V, 2 out of 16 colors), each two bytes of information respresents the color of 8 adjacent dots counting from the left end.

Even byte:

MSB							LSB
●	●	●	●	●	●	●	●

● = '1' select Foreground color

● = 'Ø' select Background color

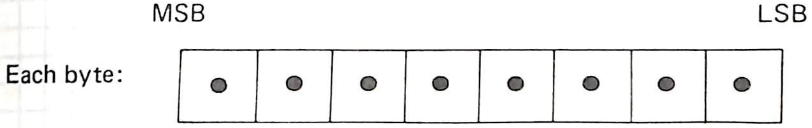
Odd byte:

Br	R	G	B	Br	R	G	B
Color code of Foreground color				Color code of Background color			

The 8 adjacent dots that appear on the screen are a bit-image of the bit pattern on the even byte, except that the LSB is shown on the right hand side of the screen. Each logical '1' bit would result in a Foreground color dot. Similarly, each logical 'Ø' bit would result in a Background color dot.

Therefore, a total of 8Ø bytes gives you 4Ø X 8 = 32Ø dots on one display line.

For GR 5 (640H X 192V 2 colors), each byte of data is a bit-image of the 8 adjacent dots on the screen.



● ='1'→Foreground color

● ='Ø'→Background color

The Foreground/Background color of the entire screen is determined by the byte of data written to I/O Port 45H. Each logical '1' bit results in a Foreground color dot. Each 'Ø' bit results in a Background color dot.

Fig.2–5 Graphic Mode (GR1, GR2) Screen Map

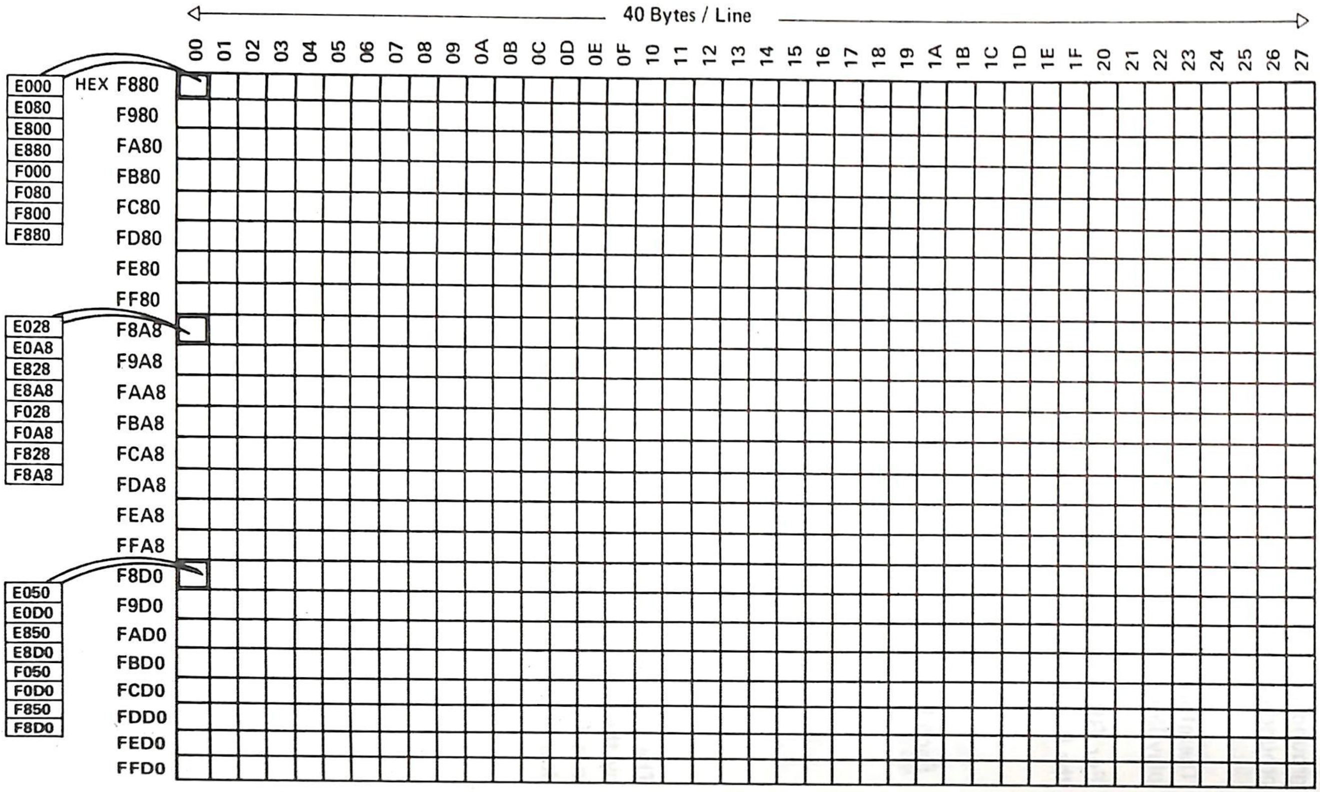


Figure 2–5 is the screen map for 8K graphic modes (GR2, GR1)

Each display line is associated with 40 bytes of RAM data on the display area.

For GR1 (160H X 192V, 2 out of 16 colors), each two consecutive bytes of information represents the color of one set of 8 adjacent dots counting from the left end. The even byte is a bit image of the dot on the screen, while the odd byte gives the Foreground/Background color code of that set of 8 dots.

The situation is exactly the same as in GR4, except that a total of 40 bytes give rise to 20 X 8 = 160 dots on one display line.

Fig.2—3 Text Mode Screen Map

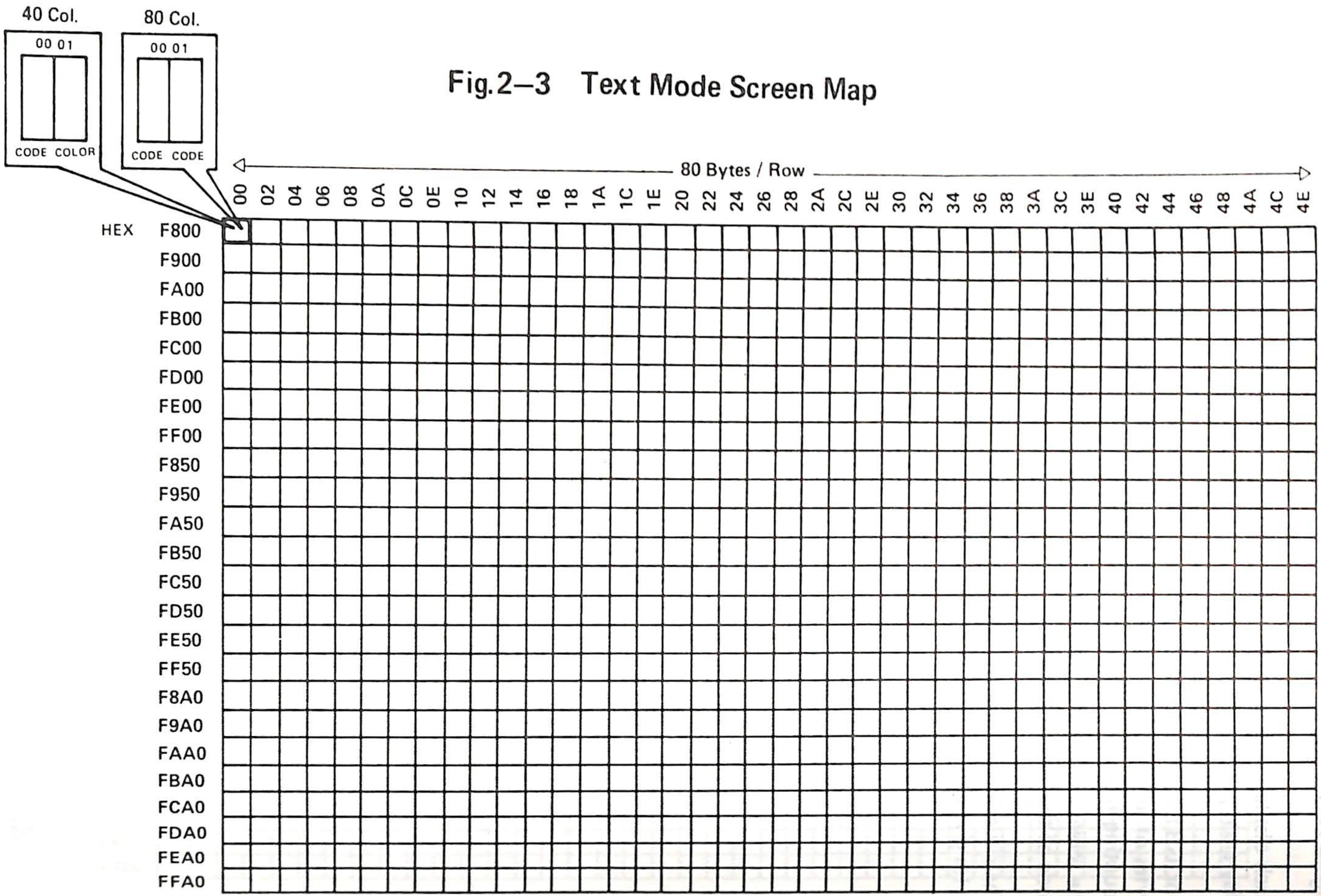


Figure 2–6 is the screen map for 8K graphic mode GRØ (160H X 96V 16 colors)

This graphics mode is a little different from the others, despite the fact that only 96 pixels are available in the vertical direction, there are actually 192 discrete display lines on the active display area. The odd number lines (1, 3, 5 ....., 191) are in fact duplicated from the upper lines (Ø, 2, 4, ...19Ø).

There are 8Ø bytes of video RAM data associated with each display line. Each byte represents the color of two adjacent dots, as follows:

1 byte of Video data in GRØ

MSB							LSB
Br	R	G	B	Br	R	G	B
Color code of second dot				color code of first dot			

The situation is similar to GR 3, except that this display line is duplicated on the line immediately below.

There, a total of 80 bytes give rise to 80 X 2 = 160 dots on one display line.

CHAPTER 3

HARDWARE CONFIGURATION

- SYSTEM OVERVIEW
- Z-80A CPU
- RAM SUBSYSTEM
- ROM SUBSYSTEM
- MASTER TIMING GENERATOR
- INTEGRATED VIDEO, DYNAMIC RAM TIMING AND I/O PROCESSOR SUBSYSTEM
- KEYBOARD
- SOUND OUTPUT
- CASSETTE INTERFACE
- VIDEO OUTPUT
- RF OUTPUT
- POWER ADAPTOR MODULE (LASER 350/500)
- POWER SUPPLY UNIT (LASER 700)
- FLOPPY DISK DRIVE CONTROLLER INTERFACE (LASER 700)
- CENTRONICS PRINTER INTERFACE (LASER 700)
- EXPANSION BUS CONNECTORS

HARDWARE CONFIGURATION

SYSTEM OVERVIEW

Your computer is built around a Z-80A microprocessor chip. Surrounding this CPU is a number of other chips that help to complete the system. These include a VLSI Gate Array which controls the overall 256K Address Space Configuration the system timing, Dynamic RAM timing, ROM decoding. I/O decoding and video signal generation. Other digital circuit include RAM and ROM chips, RAM address multiplexer, character generator ROM, latch and tri-state buffer chips.

Apart from these, there are linear circuits which perform the remaining task of master clock generation, video signal processing, sound channel, cassette input/output and power supply regulating circuit.

For the LASER 700 additional digital and linear circuits are built-in incorporate the Power Supply unit, the Printer Interface and the Floppy Disk Drive Controller.

Figure 3—1 shows the System Block Diagram

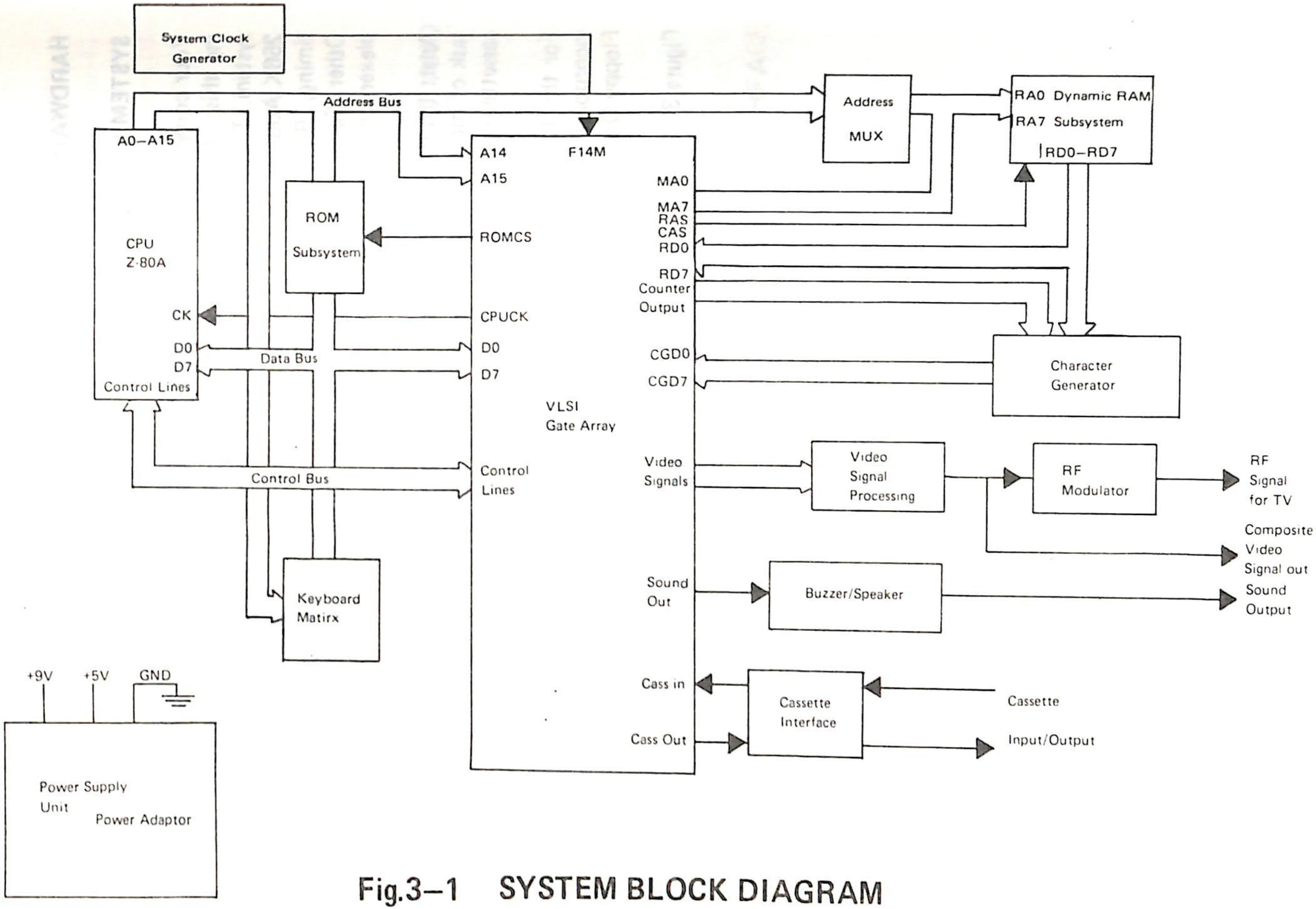


Fig.3–1 SYSTEM BLOCK DIAGRAM

The Z-80A CPU

The Z-80A CPU that your computer uses is a third-generation microprocessor with exceptional computational power and a high system throughput.

If features 16 address lines (64K Bytes of address directly), 8 data lines, and an operating speed of up to 4MHz. The internal registers contain 2Ø8 bits of read/write memory accessible to the programmer. These registers include two sets of six general purpose registers which may be used individually as either 8-bit registers or as 16-bit register pairs. In addition, there are two sets of accumulator and flag registers. A group of "Exchange" instructions allows either set to accessible to the programmer. This microprocessor also contains a Stack Pointer, Program Counter, and an Interrupt register. It requires only a single +5V power source. All output signals are fully decoded and timed to control external memory or peripheral circuits.

For details of the software programming model and hardware information, please refer to the manufacturer's data sheets.

The Z-80A inside your computer (LASER 350/500/700) operates at a clock frequency of 3.7MHz (3.6947MHz). This clock frequency is derived from a crystal oscillator which generates a high frequency master clock which is processed in an integrated Video/Input Output/Dynamic RAM timing cirucitry.

The 3.7MHz clock frequency which Z-80A is running at, does not vary with the screen display mode.

Fig 3—2 shows the interface to the Z-80A CPU.

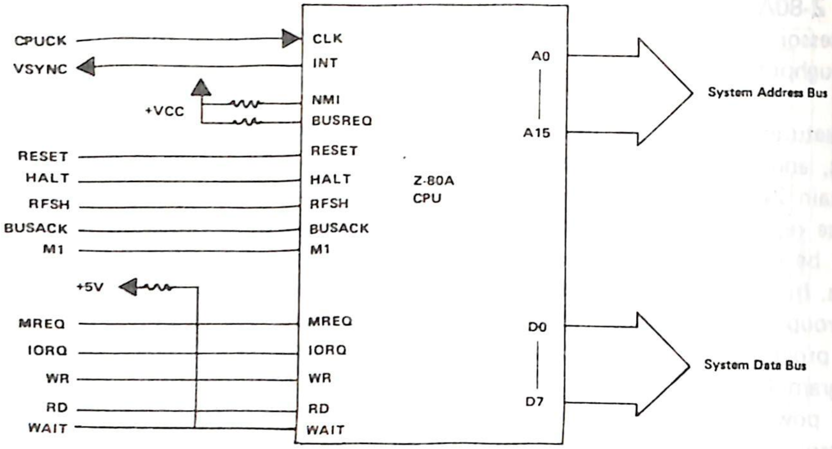


Fig. 3—2 Z—80A CPU Interface

As shown in the figure for Z-80A CPU Interfacing, the microprocessor directly drives 16 lines of the System Address Bus and the 8 lines of the System Data Bus. Other control lines input to, or output from the CPU for system timing control, memory and Input/Output purposes.

Two major control lines are of interest to for Assembly Language Programmers.

1. The INT (Interrupt Request) input to the CPU is connected to the VSYNC (Vertical Synchronization) output of the VLSI Gate Array. This implies that the CPU would receive an Interrupt Request for every vertical synchr period. Normally, the BASIC Interpreter uses this interrupt service to read the keyboard, flash the cursor and perform some internal operations. If you are doing some time critical task which can't be interrupted, you should set the Interrupt mask flag in your program thus preventing the VSYNC from Interrupting your program.

2. The WAIT (CPU WAIT) signal is connected to one of the outputs in the VLSI Gate Array. This signal is implemented so as to insert one WAIT state for every opcode fetch, memory read or memory write cycle. As a result, any software loop written in the Assembly Language Program must take into account this extra WAIT state. The purpose of this WAIT state is to synchronize the memory system with the video system.

The timing diagram for the CPU/Video system is shown in Fig 3–3

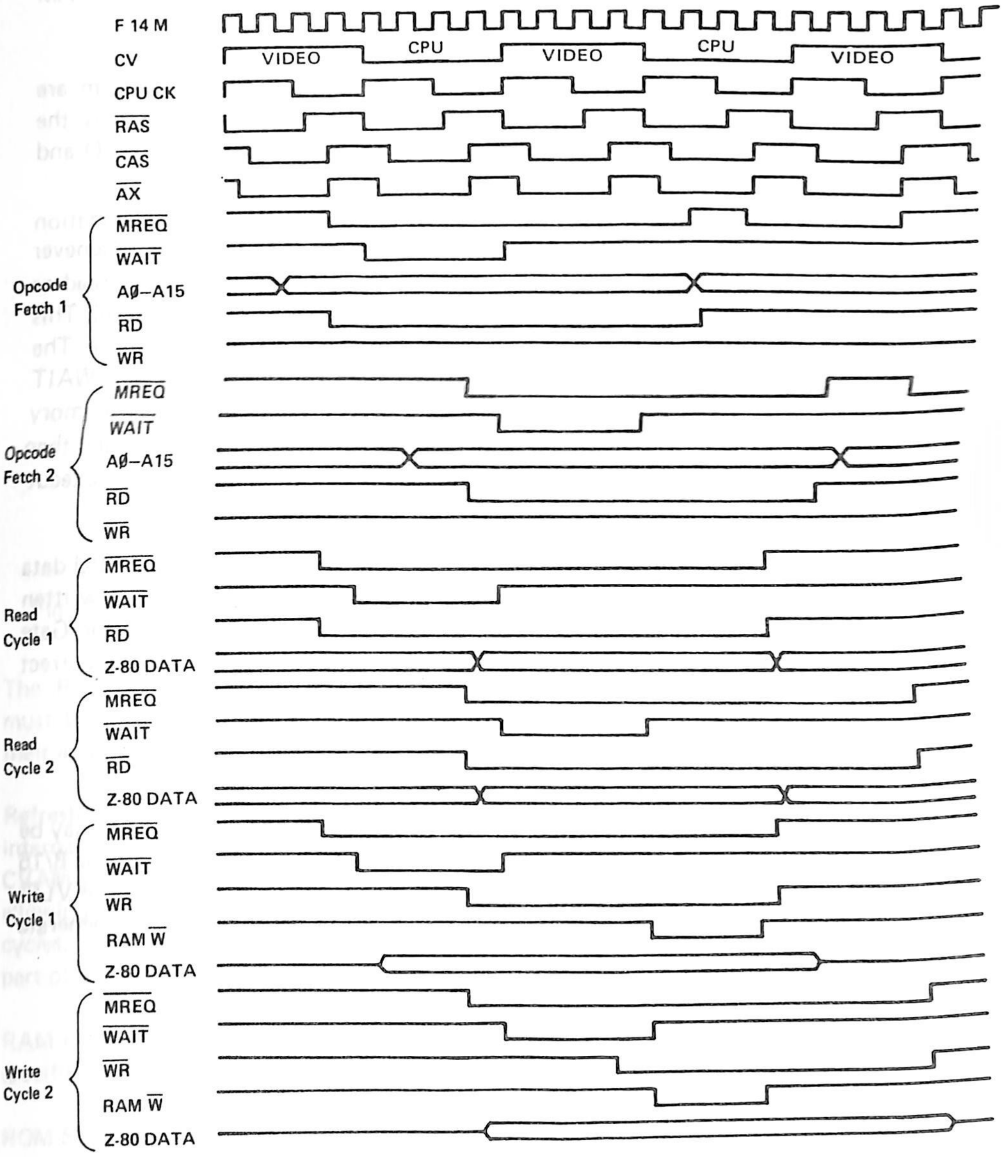


Fig. 3–3 CPU/Video system

Fig 3–2 shows the interface to the Z-80A CPU.

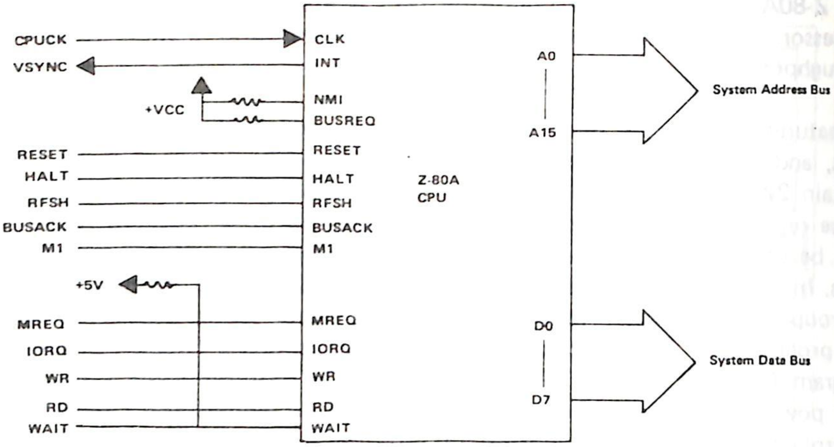


Fig. 3–2 Z–80A CPU Interface

As shown in the figure for Z-80A CPU Interfacing, the microprocessor directly drives 16 lines of the System Address Bus and the 8 lines of the System Data Bus. Other control lines input to, or output from the CPU for system timing control, memory and Input/Output purposes.

Two major control lines are of interest to for Assembly Language Programmers.

1. The INT (Interrupt Request) input to the CPU is connected to the VSYNC (Vertical Synchronization) output of the VLSI Gate Array. This implies that the CPU would receive an Interrupt Request for every vertical synchr period. Normally, the BASIC Interpreter uses this interrupt service to read the keyboard, flash the cursor and perform some internal operations. If you are doing some time critical task which can't be interrupted, you should set the Interrupt mask flag in your program thus preventing the VSYNC from Interrupting your program.

2. The WAIT (CPU WAIT) signal is connected to one of the outputs in the VLSI Gate Array. This signal is implemented so as to insert one WAIT state for every opcode fetch, memory read or memory write cycle. As a result, any software loop written in the Assembly Language Program must take into account this extra WAIT state. The purpose of this WAIT state is to synchronize the memory system with the video system.

The timing diagram for the CPU/Video system is shown in Fig 3–3

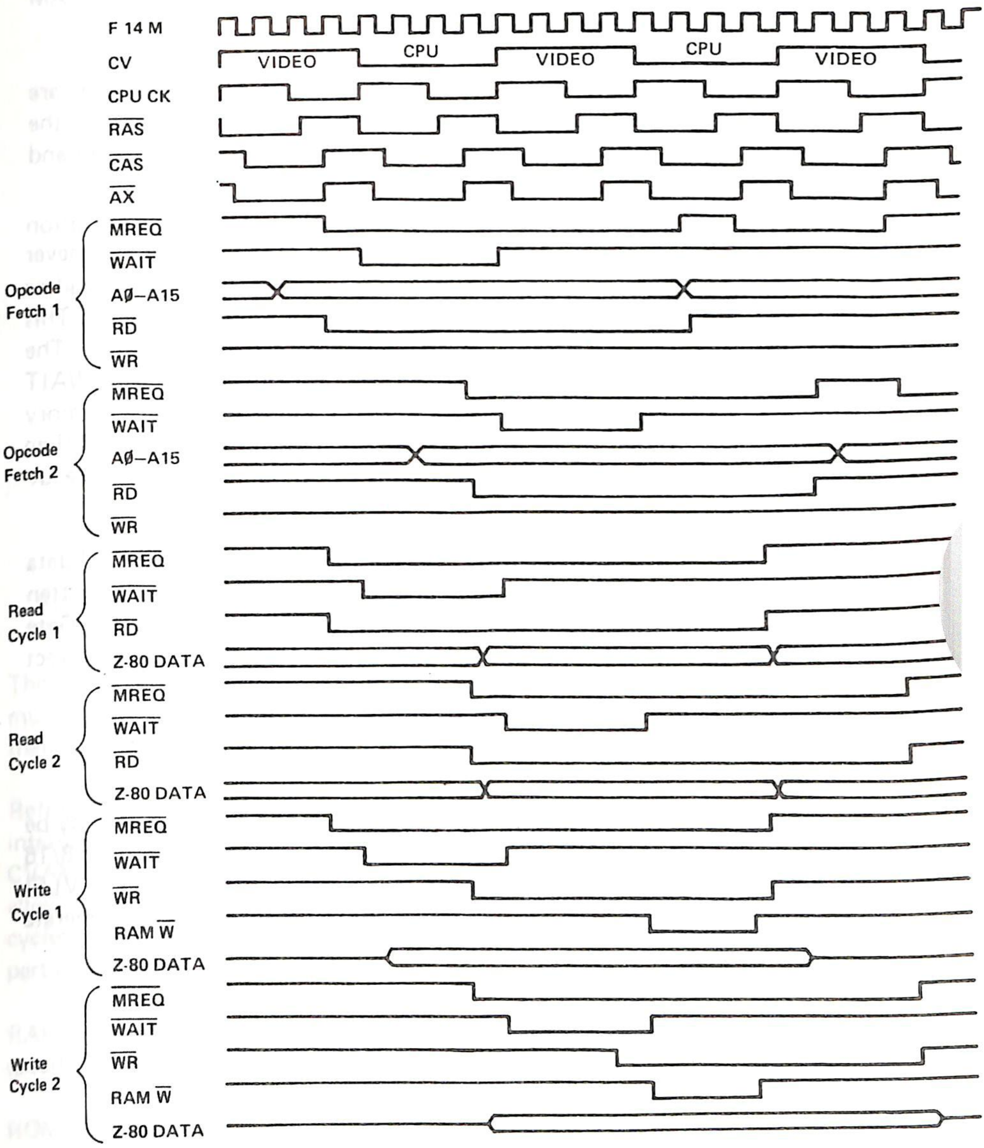


Fig. 3–3 CPU/Video system

The CPU CLOCK (CPUCK) is shown as a train of rectangular pulses at a 50% duty cycle with frequency 3.7MHz and a period of 270nS. CV is a signal used internally in the VLSI Gate Array to partition the system timing into VIDEO and CPU cycles. When CV = 1, the RAM data would be video data, otherwise CPU data.

The RAS/CAS and AX signal in the Dynamic RAM Subsystem are shown as rectangular pulses running at the same frequency as the CPU. Their edges are timed precisely to match the CPU/VIDEO and RAM requirements.

A special technique is used in the VLSI Gate Array for synchronization between CPU data timing and the multiplexed RAM timing. Whenever a memory operation takes place, such as Opcode fetch, memory read or memory write, the CPU pulls the MREQ line to a low logic level. This informs the other units that a memory transaction will take place. The VLSI Gate Array will respond to that signal by pulling the CPU WAIT line to low logic level, thus inserting a WAIT cycle into every memory operation. This WAIT signal will last for one CPU clock cycle and than go back to high logic level, thus releasing the CPU to continue execution.

With this "WAIT Cycle Insertion", we can guarantee that all CPU data are read at the right phase of CV, and that all CPU data are written corresponding to the CPU phase of CV. Internal circuitry of the Gate Array takes care of latching up the data and releasing it at the correct time.

RAM SUBSYSTEM

The RAM subsystem consists of the on-board RAM chips which may be 2 pieces of 16K X 4 bits Dynamic RAM (for LASER 350) or 8/16 pieces of 64K X 1 bit Dynamic RAM (for LASER 500/700). The VLSI Gate Array , together with multiplexer chips LS257 serves to generate the RAM timing.

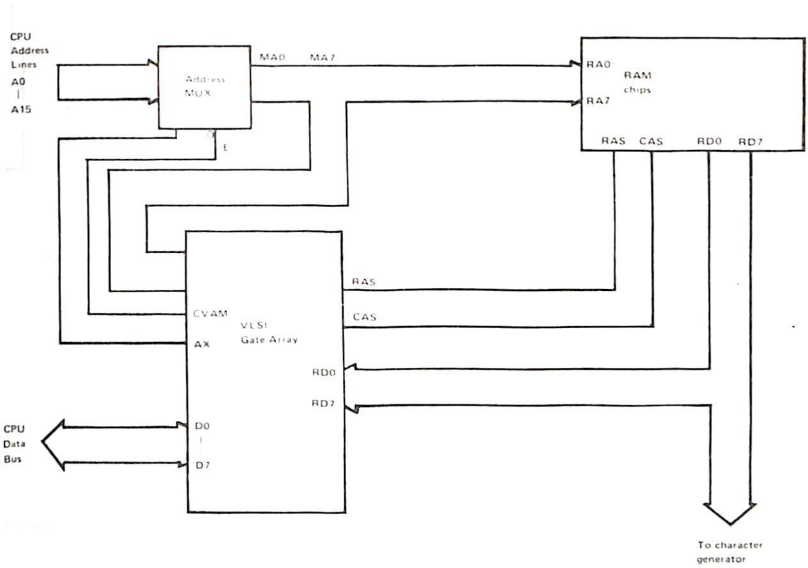


Fig. 3–4 RAM Subsystem block diagram

The RAM chips used in the computer are Dynamic RAM, i.e., they must be refreshed at a certain rate to keep their data alive. Moreover, their addresses are multiplexed as row address and column address.

Refreshing counter, RAS timing and CAS decoding are all done by the internal circuitry of the VLSI Gate Array. Moreover, an enable signal CVAM is output from the Gate Array to the Address Multiplexer, thus allowing the Gate Array to take control of the RAM during Video cycles. RAM data is also fed to a Character Generator Latch which is part of the video display circuit.

RAM chips with access time of 150ns are used. For details of the RAM specifications, please refer to data sheets from the manufacturer.

ROM SUBSYSTEM

The ROM subsystem consists of a 32K Byte (256K bit) mask programmed Read–Only–Memory chip. Eariler products may use two pieces of 27128 EPROM, together with decoding logic to divide the 32K Bytes into two 16K Bytes range.

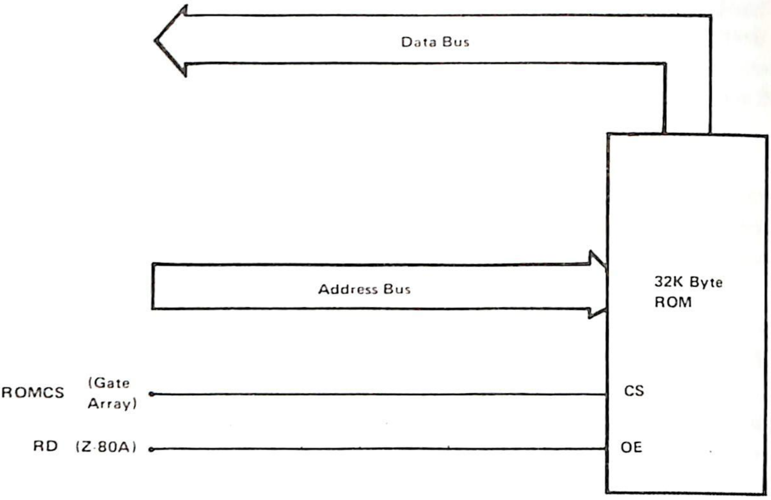


Fig. 3—5 ROM Subsystem

The decoding of the ROM is done inside the VLSI Gate Array, which outputs the ROMCS (ROM SELECT) signal to the ROM CS input. Data output from ROM is controlled via an RD (READ) signal from the Z-80A CPU.

The ROM chip is a high speed 250ns access time device. For details on ROM specifications, please refer to the manufacturer's data sheets.

MASTER TIMING GENERATOR

The master timing generator inside your computer is basically a high frequency crystal oscillator. The actual frequency of the crystal used in your model may vary with different TV systems. The output from this oscillator is further buffered and processed by digital circuits to produce a 14MHz digital pulse train. The VLSI Gate Array makes use of this system clock to generate all related timing, CPU CLOCK and color reference signal.

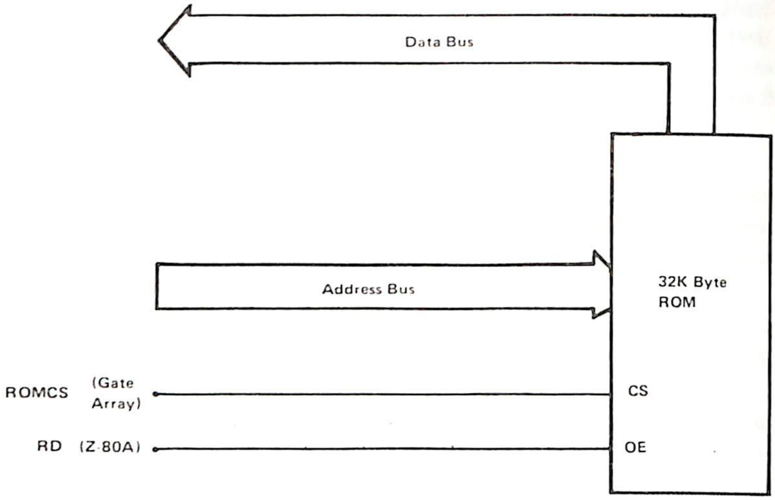


Fig. 3—5 ROM Subsystem

The decoding of the ROM is done inside the VLSI Gate Array, which outputs the ROMCS (ROM SELECT) signal to the ROM CS input. Data output from ROM is controlled via an RD (READ) signal from the Z-80A CPU.

The ROM chip is a high speed 250ns access time device. For details on ROM specifications, please refer to the manufacturer's data sheets.

MASTER TIMING GENERATOR

The master timing generator inside your computer is basically a high frequency crystal oscillator. The actual frequency of the crystal used in your model may vary with different TV systems. The output from this oscillator is further buffered and processed by digital circuits to produce a 14MHz digital pulse train. The VLSI Gate Array makes use of this system clock to generate all related timing, CPU CLOCK and color reference signal.

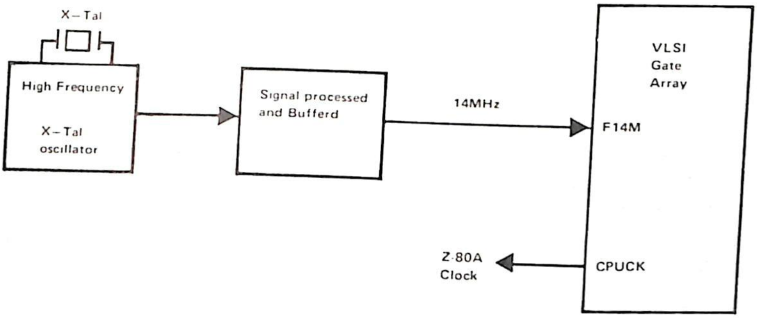


Fig. 3–6 Master Timing Generator

THE INTEGRATED VIDEO, DYNAMIC RAM TIMING AND I/O PROCESSOR SUBSYSTEM

The integrated video, dynamic RAM timing and I/O processor subsystem consists of one custom designed VLSI Gate Array and external chips including character generator ROM, TTL latch for character pattern data and dynamic RAM address multiplexer chips.

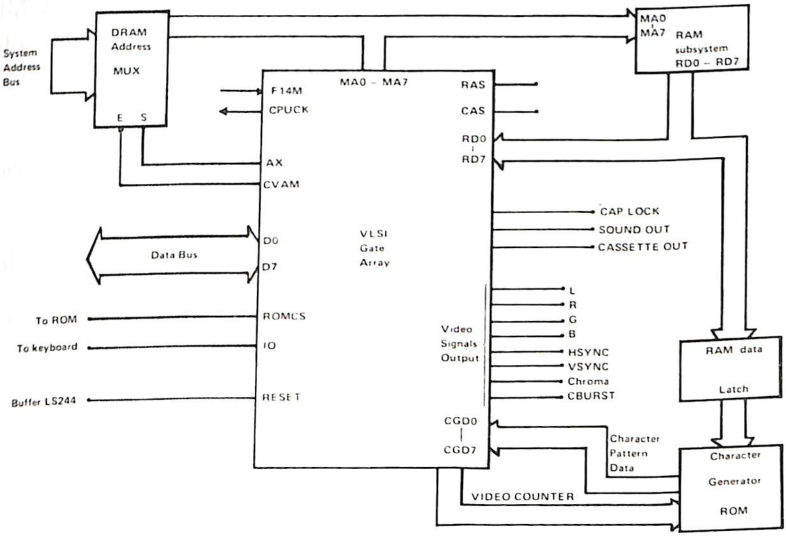


Fig. 3–7 Integrated Video, Dynamic Ram timing ,I/O Processor Subsystem Block Diagram

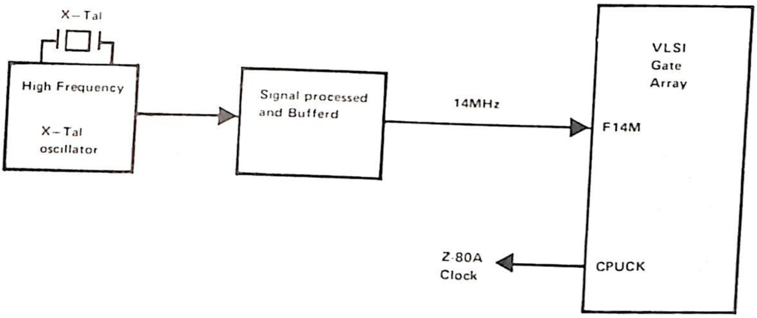


Fig. 3–6 Master Timing Generator

THE INTEGRATED VIDEO, DYNAMIC RAM TIMING AND I/O PROCESSOR SUBSYSTEM

The integrated video, dynamic RAM timing and I/O processor subsystem consists of one custom designed VLSI Gate Array and external chips including character generator ROM, TTL latch for character pattern data and dynamic RAM address multiplexer chips.

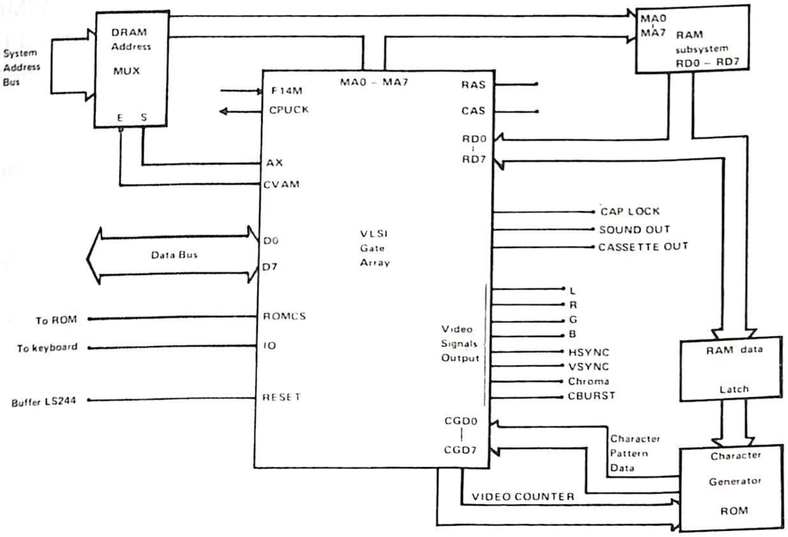


Fig. 3–7 Integrated Video, Dynamic Ram timing ,I/O Processor Subsystem Block Diagram

The internal circuitry of the VLSI Gate Array contains the counter, shift register, multiplexer and related logic functions necessary for the video section in which graphics and text mode are handled. The text mode character pattern data are supplied by an external character generator ROM. Video signals output from the Gate Array are in digital form as found in signals L, R, G, B, HSYNC, VSYNC, Chroma and CBURST. These digital signals are combined to form the composite video signal by a linear circuit which will be described in the coming pages.

The dynamic RAM timing generation is another major function of this VLSI Gate Array. The RAM timing is partitioned into the CPU phase and VIDEO phase. The CPU can only access the RAM during the CPU phase while video circuitry will take control of the RAM during the VIDEO phase. Simultaneously, dynamic RAM refreshing is done in the VIDEO phase. The Gate Array is also responsible for latching up the RAM data and releasing it to the System Data Bus for CPU use.

System timing and the CPU clock generation forms another function of this Gate Array. A 14MHz signal (F14M) is connected from the system clock generator into the Gate Array. All system timing, and CPU clock are based on this 14MHz signal. Moreover, an 17.73447MHz signal is used for the color signal. The rest of the memory-mapped I/O decoding, I/O ports decoding. ROM decoding and Memory Bank-switching are done inside the VLSI Gate Array.

To summarize, this VLSI custom chip replaces hundreds of discrete TTL I.C. chips that would be necessary in a conventional design.

The following table summarizes the pinout and describes the functions of each pin on the VLSI Gate Array.

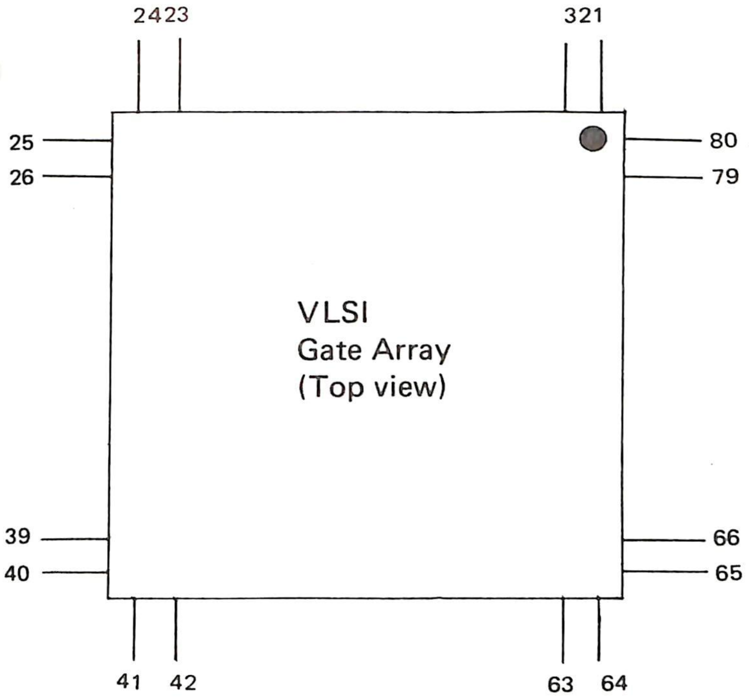


Fig. 3—8 Pin Assignment of VLSI Gate Array

PIN	NAME	DESCRIPTION
1	MAØ	RAM address line
2	MA1	RAM address line
3	MA2	RAM address line
4	MA3	RAM address line
5	MA4	RAM address line
6	MA5	RAM address line
7	MA6	RAM address line
8	MA7	RAM address line
9	CAS1	RAM control line
10	CAS4	RAM control line
11	CAS5	RAM control line
12	Vss	GND
13	RAS	RAM control line
14	RAMW	RAM control line
15	RDØ	RAM data line
16	RD1	RAM data line
17	RD2	RAM data line
18	RD3	RAM data line
19	RD4	RAM data line
20	RD5	RAM data line
21	RD6	RAM data line
22	RD7	RAM data line
23	F17M	17.73447 MHz input
24	F14M	14.77873 input
25	F3M	2.958MHz output
26	GT	1 = graphic mode
27	CGS	character generator strobe
28	CGDØ	character generator data line
29	CGD1	character generator data line
30	CGD2	character generator data line
31	CGD3	character generator data line
32	CGD4	character generator data line
33	VDD	5V
34	CGD5	character generator data line
35	CGD6	character generator data line
36	CGD7	character generator data line
37	VA	Vertical address
38	VB	Vertical address
39	VC	Vertical address
40	ROMCS	ROM chip select
41	CAPLK	Cap lock LED indicator
42	BUZZER	Buzzer output
43	IO	IO select
44	A15	CPU address input

PIN	NAME	DESCRIPTION
45	A14	CPU address input
46	WAIT	CPU control line
47	CPUCK	CPU clock
48	RD	CPU control line
49	WR	CPU control line
50	MREQ	CPU control line
51	IORQ	CPU control line
52	Vss	GND
53	D7	CPU data line
54	D6	CPU data line
55	D5	CPU data line
56	D4	CPU data line
57	D3	CPU data line
58	D2	CPU data line
59	D1	CPU data line
60	DØ	CPU data line
61	BA17	Extended address line
62	BA16	Extended address line
63	BA15	Extended address line
64	BA14	Extended address line
65	AX	ROW/COLUMN address multiplex
66	CVAM	CPU/VDP address multiplex
67	TOPBK	Top bank
68	GRESET	reset
69	CASOUT	cassette output
70	REMOTE	remote control output
71	CBURST	color burst
72	CHROMA	color output
73	VDD	5V
74	FRSEL	50Hz/60Hz select
75	VSYNC	Vertical sync
76	HSYNC	horizontal sync
77	L	luminance
78	B	blue output
79	G	green output
80	R	red output

THE KEYBOARD

The keyboard of the LASER 350/LASER 500/LASER 700 computers are not encoded ASCII keyboards. They will not generate any code by themselves, as is the case in most other personal computer keyboards. Rather, they are read by the CPU (Central Processing Unit) and ASCII encoding is done by the software program as part of the resident ROM BASIC Interpreter.

The keyboard of the LASER 350 consists of a 8 row by 7 column Key Matrix. The keyboard of LASER 500/LASER 700 consists of a 12 row by 7 column Key Matrix.

The horizontal lines (rows) of the matrix are connected to the system Address Bus lines via diodes.

The vertical lines (columns) of the matrix are normally tied to +5V by resistors. These lines are also buffered and gated to the system data Bus through a TTL 74LS 244 tri-state Buffer, which is controlled by the I/O (pin 43) of the Gate Array (VLSI) chip.

When the CPU is reading the keyboard for any key enclosure, it will send out a series of Ø bit patterns on the system Address lines. If any key is pressed, one of the lines KDØ–KD6 would be tied to 'Ø' logic level. This would be reflected on the Data Bus when IO (pin 43) of Gate Array goes to the 'Ø' logic level. The CPU then reads the logic level of KDØ–KD6, compares it with the bit pattern on the Address Bus, and determines which key was actually pressed.

If more than one key is found to be depressed, the CPU assigns a different code in response to that key pressed. For instance the SHIFT or the CTRL key pressed with other keys will generate either the upper case letter of the key, another symbol, or a special function.

If the CPU detects that any key has been depressed for a certain length of time, say 1 second, it will continuously generate the associated code. That is, all keys are auto-repeating.

Similarly, the CAP LOCK key of the LASER 500/700 is sensed by software and toggles for each key pressed. The cap lock light indicates a CAP LOCK condition and comes directly from the CAPLK (pin 41) of the Gate Array.

The RESET key of LASER 500/700 is not itself part of the key matrix. It is not read by the keyboard polling software. Rather, it is a hardware RESET line going directly to tthe Z-80 CPU. Each time the user presses this key, the RESET line will be grounded and a hardware RESET cycle is initiated. Refer to the Z-80 data sheets for details of this RESET cycle.

Two jumper positions are present for the standard (English), French and German keyboards. They are at the intersection points of (A5, KD5) and (A5, KD4). In the French/Germany versions, one of the intersection points would be permanently shorted. The CPU would be able to detect this and then the keycode assignments of some keys would be modified automatically. This allows three different language versions to use the same keyboard polling software.

SOUND OUTPUT

Sound output is obtained by a pizeo-electric buzzer located on the bottom cabinet of your computer. The buzzer's signal comes from your computer. The buzzer's signal comes from the Buzzer (pin 42) output of the VLSI Gate Array. This is the same for the LASER 350/LASER 500/LASER 700.

This digital signal is further boosted by a simple transistor circuit as shown here:

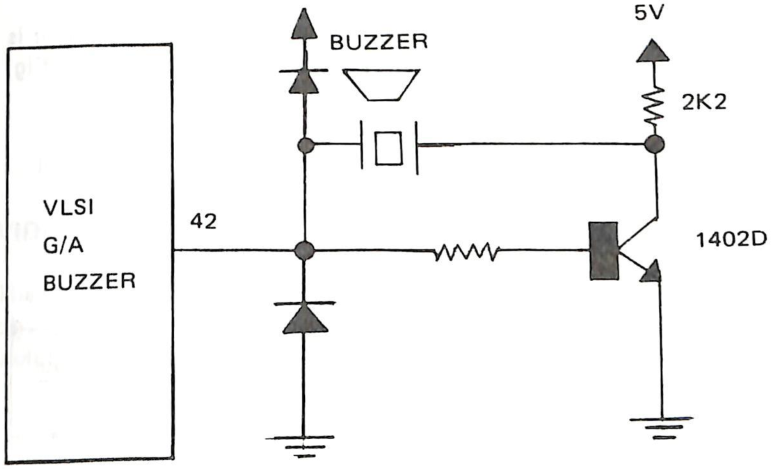


Fig. 3–9 Sound Output

CASSETTE INTERFACE

On the rear panel of your computer is a miniature stereo phone jack into which you can plug a cassette cable and connect the other end to a data recorder or audio cassette recorder. You computer will then be able to save information onto a cassette tape and read it back again.

The circuitry, signal requirements and connector pinout assignment of the LASER 350/LASER 500/LASER 700 are the same.

The cassette-out signal (digital) comes directly from VLSI Gate Array pin 69 and is translated to suitable level and waveform by the circuit shown in Fig. 3–10

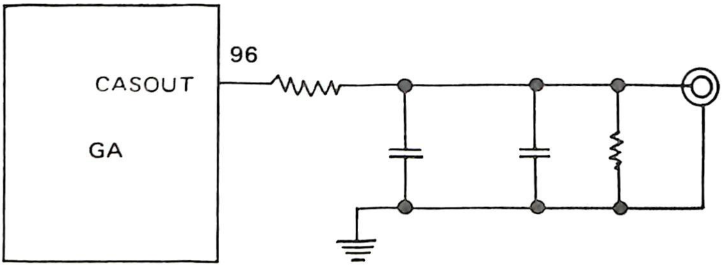


Fig. 3–10 Cassette output

The cassette in signal from the data cassette or audio cassette player is transformed into digital level by the transistor circuit shown in Fig. 3–11

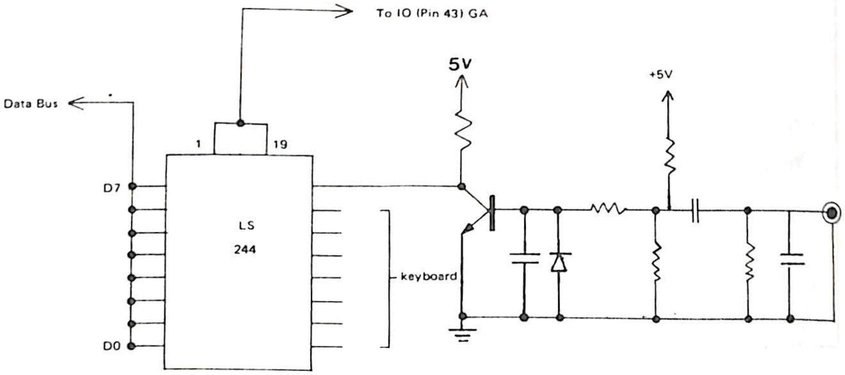
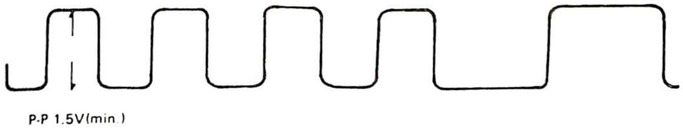


Fig. 3–11 Cassette in Circuit

This digital logic level signal is gated by a TTL 74LS244, which is also used by the keyboard, before going into D7 (MSB) of the system data bus.

When the CPU reads programs or data from cassette tape, the associated location would be read out but only changes in D7 (MSB) of a byte would be used as the cassette data.

CASSETTE INPUT SIGNAL



CASSETTE OUTPUT SIGNAL

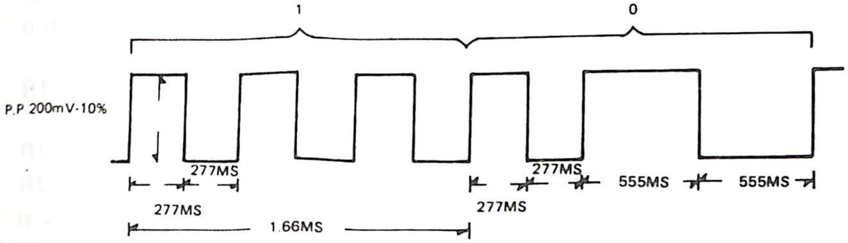


Fig. 3—12 Cassette output signal and Cassette input signal

VIDEO OUTPUT

The internal circuitry of the VLSI Gate Array handles the generation of video signal components in the form of digital signals. These signals include HSYNC, VSYNC, L, R, G, B, CHROMA and CBURST.

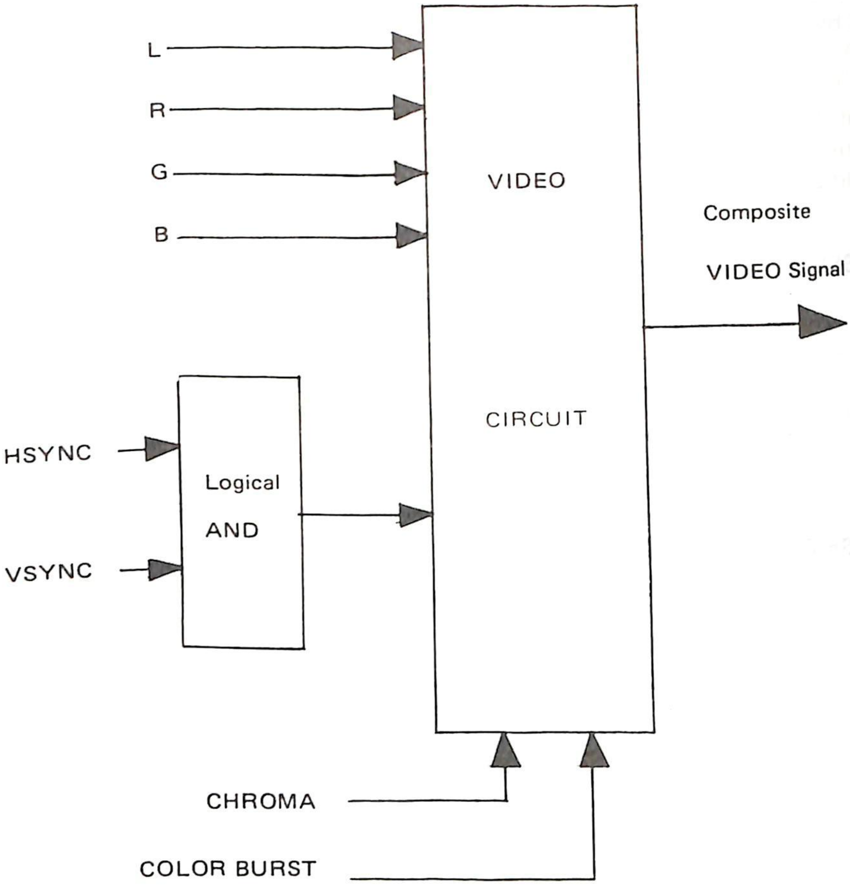


Fig. 3—13 Video Output Block Diagram

PAL and NTSC models of your computer are equipped with a COLOR DEFEAT switch at the lower cabinet (not available to SECAM models). Flipping this switch to the ON position will disconnect the CHROMA and CBURST signal from the video summer, thus suppressing any color display.

The role of the video output circuit is to combine all these video components into the composite video signal that your Color/Black and White monitor will accept. This composite signal has the following specifications:

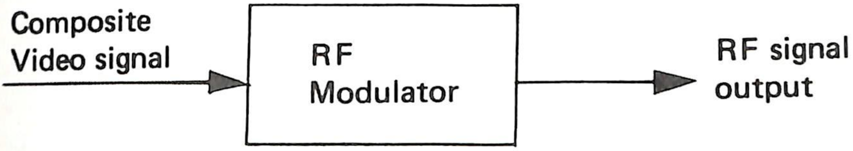
Composite signal, negative going sync
1VP–P Level on 75 OHM Load

Color Reference Signal
PAL System : 4.43MHz
NTSC System : 3.579MHz

The composite video signal is available at the standard RCA phono jack on the rear panel of your computer. The sleeve of this jack is connected to the common ground and the tip is connected to the actual video output.

RF OUTPUT

RF output, which your home TV set accepts, is available at another RCA phono jack on the rear panel of your computer. This RF signal is a modulated version of the composite video signal mentioned on the previous section.



The sleeve of the RCA phono jack is connected to the common ground and the tip is connected to the RF modulator output. . . . . .

THE POWER ADAPTOR MODULE FOR LASER 350/500

The DC power of the LASER 350/500 main unit is taken from an external Power Adaptor Module via a power connector jack located on the rear panel of the computer. Its pin assignments are as follows:

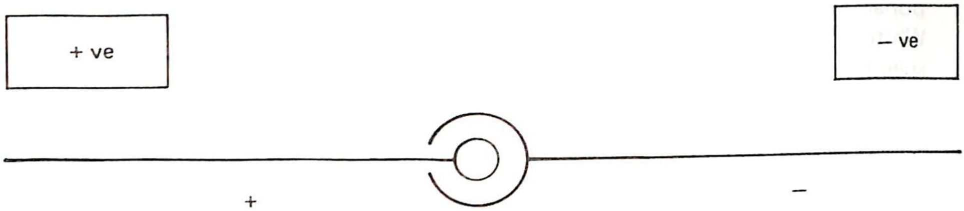


Fig 3—14 The pin assignments of the LASER 350/500 Power Adaptor Module.

Power Adaptor Module Specifications :

Input Voltage : 120V / 220V / 240V

Output Voltage : 9V

Output Current : 1A

Power Consumption : 12VA

The +9V DC from the Power Adaptor Module is further regulated by a 7805 three terminal voltage regulator IC to produce a stable +5V.

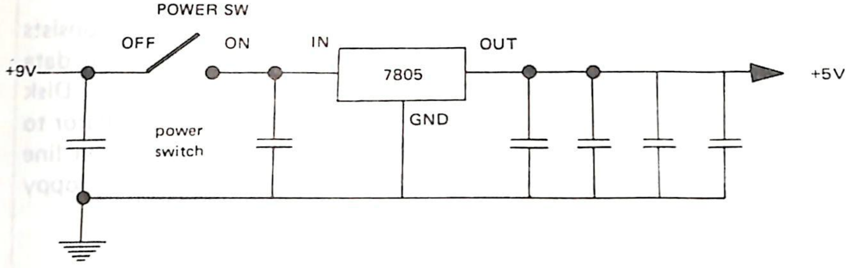


Fig. 3–15 +5V Regulator

THE POWER SUPPLY UNIT FOR LASER 700.

The built-in Power Supply Unit of the LASER 700 is located in the left side of the lower cabinet. It is a linera type power supply with the following specifications as shown on the table.
There is also a +12V power outlet for an optional Floppy Disk Drive.

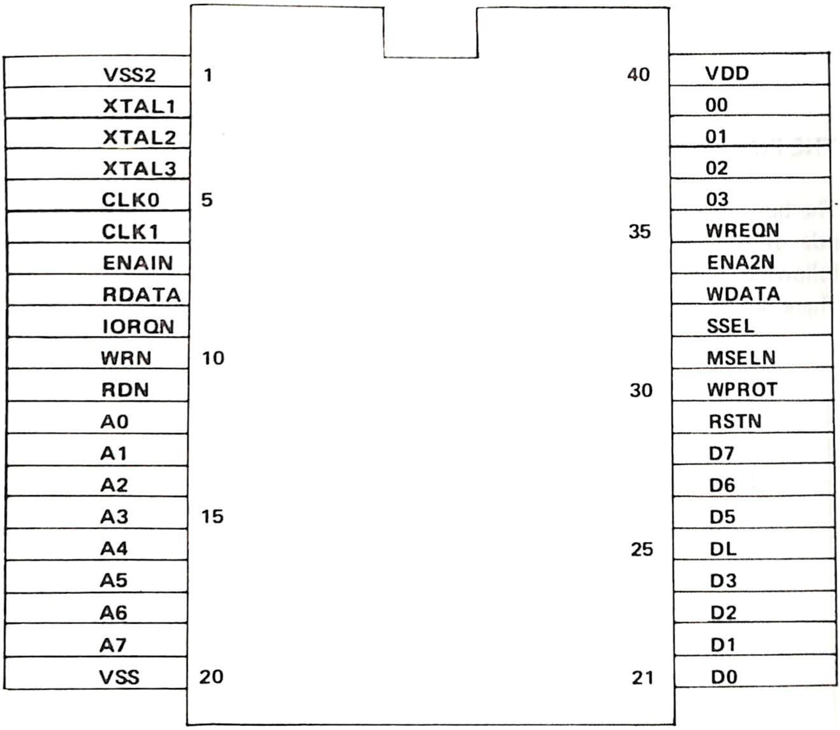
Specification of the Power Supply Unit

Line input voltage : 120V/220V/240V
Output voltages : 5V, 12V
Output current : 1.0A at 5V
1.0A at 12V
Power consumption : 28W

FLOPPY DISK CONTROLLER INTERFACE (LASER 700 only)

The Floppy Disk Controller Interface is a built-in feature of the LASER 700. It is also available for LASER 350/500 users as an optional expansion module.

This interface circuit is the same for LASER 350/500/700. It consists of only one custom designed chip which handles address decoding, data latch, parallel/serial conversion and the control signals for Floppy Disk Drive stepper motor. This custom chip requires an external oscillator to drive its on-chip oscillator. It is packaged in a 40 PIN dual in line Package. The following is a list of the pinout assignments of the Floppy Disk Controller Interface, with a description for each pin.



The interface from your LASER 700 to your Floppy Disk Drive is through two 20 pin sockets located on the rear panel. These sockets also carry the +5V, +12V power required by the Floppy Disk Drive.

For LASER 350/500 users with the optional Floppy Disk Drive Interface Module, a separate power adaptor module for the Floppy Disk Drive is required.

The I/O Ports decoded and used by the Floppy Disk Controller Interface circuit, with their functions are listed as follows:

I/) Port	Read Function	Write Function	Remarks
10 H	X	LATCH 1	Write only
11 H	X	LATCH 2	Write only
12 H	STATUS	X	Read only
13 H	READ DATA	WRITE DATA	Read/Write

X= No application

Table 3–16 Description of I/O port for Floppy Disk Drive Interface

DESCRIPTION:

LATCH 1: DØ– D3 PHASE Ø to PHASE 3 of direct stepper control
D4 Drive enable master select, (active HI)
D5 Drive select, Ø for drive 1
1 for drive 2
D6 Write request, (active LO)
D7 Side select, 1 for side 1
Ø for side Ø

LATCH 2: When write request set low, the controller will be in writing 1Ø bit data mode. The controller will toggle between 8 bit data mode and 1Ø bit data mode on every access of this I/O Port.

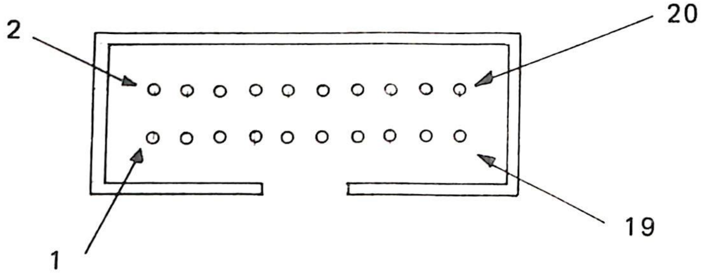
STATUS: DØ Sense write protect (active Hi)
D1–D6 Not used
D7 When write, indicate buffer empty
When read, indicates data ready

READ DATA DØ–D7 8 bit READ DATA from drive. This buffer should be read when Data Ready Flag (STATUS bit 7 HI) is set, to avoid data lost.

WRITE DATA DØ–D7 8 bit WRITE DATA to drive. This buffer should be written when buffer empty flag (STATUS bit 7 HI) is set, to avoid data lost.

The 20 pin Disk Drive Cable connector has the following pin assignments:

Pin no.	Description	Pin no.	Description
1	GND	11	+5V
2	ØØ	12	+5V
3	GND	13	+12V
4	Ø1	14	ENABLE
5	GND	15	+12V
6	Ø2	16	RD DATA
7	GND	17	+12V
8	Ø3	18	WR DATA
9	Side Select	19	+12V
10	WR REQ	20	WR PROT



CENTRONICS PRINTER INTERFACE (LASER 700 only)

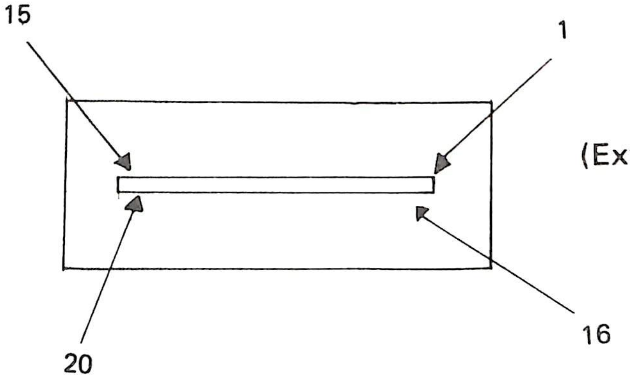
Centronics Printer Interface is another built-in feature of the LASER 700. It is available to LASER 350/500 users as an optional expansion module.

The printer interface hardware circuit is composed of simple decoding logic and data latch. A number of I/O Ports are used (00H–0FH hexadecimal). These Ports play different roles for read and write, as listed below.

I/O Port	Read/Write	Function	Remarks
ØØH–ØFH, even no.	Read	Printer Busy line	Data line DØ Printer Busy line
ØØH–ØFH, odd no.	Write	Printer Strobe	Printer strobe change from '1' – 'Ø'
ØØH–ØFH, even no.	Write	Parallel	8 bit data to printer

The connection for an optional printer cable is a-20 way edge connector located at the right side of the cabinet on the LASER 700. The pin assignments of this connector are as follows:

Pin no.	Name	Direction	Description
1	ACK	IN	Printer acknowledge (Not used)
2	BUSY	IN	From Printer. 'HI' indicates not yet ready
3	PR1	OUT	data to printer
4	PRØ	OUT	data to printer
5	PR4	OUT	data to printer
6	PR5	OUT	data to printer
7	PR2	OUT	data to printer
8	GND	–	signal ground
9	PR6	OUT	
10	PR7	OUT	
11	STROBE	OUT	active low pulse to strobe data
12	NC		
13	+5V		
14	+5V		
15	+5V		
16	NC		
17	NC		
18	PR3	OUT	data to printer
19	NC		
20	NC		



(External view)

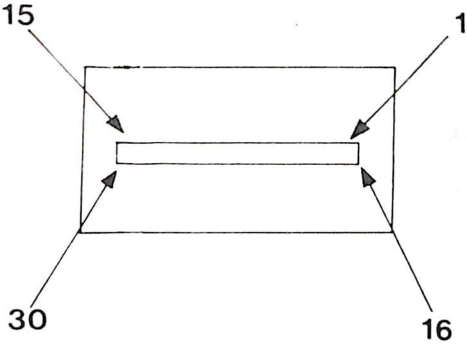
EXPANSION BUS CONNECTORS

At the rear panel of your computer, there are one, or two PCB edge connector into which you can plug in different Expansion Modules such as Joystick, Light Pen, RS232 Interface ..... etc.

Two different expansion bus connectors are available for LASER 350/500 users. The first one, which we call P1, is a 30 pin connector you would use for an optional Printer Interface Module. The same connection is used for Joystick and Light Pen Interface. P1 is also available to LASER 700 users.

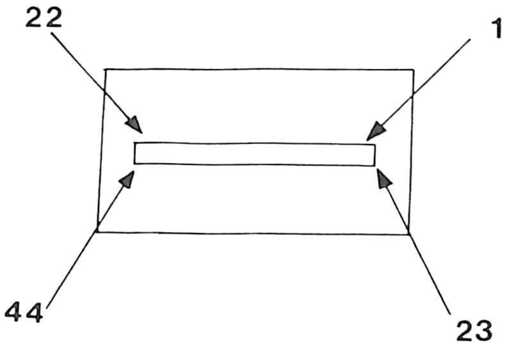
The other connector, located at the right corner on the back panel of LASER 350/500, is a 44 pin connector which we call P2. This is specially reserved for an optional Floppy Disk Drive Interface Module. P2 is not available to LASER 700 users because the Floppy Disk Drive Interface is already built inside the main unit of LASER 700. Instead, the space for P2 is used for two Disk Drive cable connectors as described in the previous pages.

Pin no.	Name
1	N.C
2	N.C
3	N.C
4	+5V
5	IORQ
6	D3
7	D6
8	D2
9	DØ
10	A5
11	A2
12	A6
13	A3
14	WR
15	GND
16	N.C
17	N.C
18	N.C
19	+5V
20	N.C
21	D4
22	D5
23	D7
24	D1
25	RD
26	A1
27	A4
28	AØ
29	A7
30	GND



P1 (External View)

Pin no.	Name
1	GND
2	RESET
3	A10
4	A9
5	A8
6	A7
7	A6
8	A5
9	A4
10	A3
11	A2
12	A1
13	D2
14	D7
15	RFSH
16	M1
17	WAIT
18	NMI
19	RD
20	IORQ
21	+5V
22	GND
23	GND
24	A11
25	A12
26	A13
27	BA14
28	BA15
29	CPUCK
30	D4
31	D3
32	D5
33	D6
34	TOPBK
35	AØ
36	DØ
37	D1
38	INT
39	HALT
40	MREQ
41	WR
42	BA16
43	9V
44	BA17



P2 (External view)

CHAPTER 4

SOFTWARE ASPECTS

- ROM Cartridge Entry Points
- Bootstrapping of Floppy Disk
- Useful Subroutines
- System Variables
- User interrupt routine
- Basic text pointers
- Screen Control Code

ROM CARTRIDGE ENTRY POINTS

As mentioned in the previous chapters, four 16K BANKS on the top of the 256K Physical Address Space are reserved for Expansion ROM. These expansion ROM can be organized as externally plug-in ROM cartridges to hold application programs or games. Your computer is able to recognize the presence of these ROM cartridges and execute their program directly after power up. This is done by checking 4 signature bytes on some predefined locations.

In fact, the resident ROM is able to detect ROM cartridge at five areas on the 256K Physical Address Space:

```
Ø8ØØØ H
3ØØØØ H
34ØØØ H
38ØØØ H
3CØØØ H
```

Where Ø8ØØØ H is on BANK 2, i.e., Memory Mapped I/O area.

The power up initialization routine will perform Bank-switching and checks for four consecutive locations starting from the above listed Address. If they match the patterns AAH, 55H, E7H and 18H, then that 16K BANK will be brought into Z-80 Address Space ØØØØH–3FFFH and the program will jump to ØØØ4H.

For instance, if you want to place your own Assembly Language Program in a ROM cartridge and have an automatic "turnkey" system, just assemble your program starting at ØØØ4H, and place the four bytes AAH, 55H, E7H, 18H in the four starting locations.

The lower BANK ROM areas are checked first and thus, have higher priority than the upper BANKS.

BOOTSTRAPPING OF FLOPPY DISKETTE

Besides having a turnkey system for ROM-based programs, your computer is also capable of automatically booting the Floppy Diskette if the Interface is installed.

Please note that ROM cartridges are always checked first and hence, are of higher priority. For example, if you have ROM cartridge as well as Floppy Disk Interface both installed, then the ROM cartrdige would always be executed first without booting the diskette.

The resident ROM BASIC Interpreter checks for the existance of Floppy Disk Interface by writting to, and reading from I/O Port 13H. If such a Read/Write Port exists, the built-in software will read track Ø, sector Ø of the Floppy diskette, place the 256 bytes of data in RAM location A2ØØH (Z-80 Address Space) onwards, then jump to A2ØØH. Normally, A2ØØH contains a further bootstrapping code to load the rest of the operating system into main memory.

USEFUL SUBROUTINES

The following list contains some useful entry points to the ROM subroutines which are part of the BASIC Interpreter. User may call these subroutines via the BASIC command CALL, or directly executes CALL from their own Assembly Language program.

Z-80 Address	Subrotuine Name	Description
0BH	CONOUT	Outputs a character whose code is placed in A Register, to the text screen.
13H	CONIN	Inputs a character from the terminal (keyboard) and returns the code in A Register.
23H	ROLLOVER (user should put I/O BANK IN 4ØØØH–7FFFH first)	Scans the keyboard and returns a code in A Register.
2BH	OUTDO	Outputs a character whose code is in A Register, to the text screen, or printer, or cassette.
33H	READSC	Reads a sector from disk Track no. in 86Ø8H Sector no. in 86Ø9H Primary buffer pointer in 86ØDH (reserve 342 byte) Data buffer pointer in 860BH (reserve 256 byte) Reads table pointer in 860FH (at power up the table is moved to RAM A196H–A1FFH)
47H	RDBYTE	Reads a byte from cassette & places code in A Register.
4AH	WRBYTE	Writes a byte to cassette, whose code is placed in A Register.

SYSTEM VARIABLES

Address (HEX)	Description
8ØØØ	RST 8 jumps here
8ØØ3	RST 1Ø jumps here
8ØØ6	RST 18 jumps here
8ØØ9	RST 2Ø jumps here
8ØØC	RST 28 jumps here
8ØØF	RST 3Ø jumps here
8Ø12	INTERUPT ROUTINE EXIT POINT
8Ø15–8Ø1A	reserved
8Ø1B–8Ø2E	USR function addresses
8Ø2F	number of nulls
8Ø3Ø	store eaten char when not CTRL-C
8Ø31	save error number
8Ø32	last line printer operation. Ø=LF
8Ø33	position of printer head
8Ø34	Ø = output to screen 1 = output to printer 8Ø = output to cassette
8Ø35	last column number beyond which no more comma field
8Ø36	default line printer width
8Ø37	line length
8Ø38	position of last comma column
8Ø39	reserved
8Ø3A	supress output flag
8Ø3B–803C	reserved
8Ø3D	top location to use for the stack
8Ø3F	current line number
8Ø41	pointer to beginning of text
8043	address of message to print (overflow)
8Ø45–8149	reserved
814A–8287	krunch buffer
8288	store a comma
8289–838B	keyboard buffer, type in stored here
838C	reserved
838D	in getting a pointer to a variable
838E	variable type
838F	store operator number
839Ø	reserved
8391	save text pointer used by CHRGET
8393	saved token for a constant
8394	saved constant type
8395	saved constant value
839D	highest location in memory

839F	pointer at first free temp descriptor
83A1	storage for numtmp temp descriptors
83BF	string function build answer descriptor here
83CØ	where string address is stored in 83BF
83C2	top of string free space
83C4	used to store the address of the end of string arrays in garbage collection
83C9	used by garbage collection
83C8	saved text pointer at end of "FOR" statement
83CA	data line number – remember for errors
83CC	flag whether subscripted variable allowed
83CD	flag whether we are doing input
83CE	reserved
83DØ	zero if no line number converted to pointer
83D1	flag to indicate auto command in
83D2	current line being inserted by auto
83D4	auto increment
83D6	place where text pointer are saved
83D8	stack saved here before execution
83DA	line number where last error occured
83DC	keeps current line for list
83DE	text pointer for use by "RESUME"
83EØ	line to goto when an error occurs
83E2	equals to one if an error trap routine executing
83E3	formula evaluator temp old line number (set up by CTRL–C, STOP)
83E7	old text pointer
83E9	pointer to start of simple variable
83EB	pointer to beginning of array table
83ED	end of storage in use
83EF	POINTOR TO DATA, used by READ statement
83F1–84ØA	This table gives the deafult variable type for each letter of the alphabet
84ØB	previous definition block on stack
84ØD	number of bytes in the active table
84ØF–8472	active parameter table
8473	the pointer at the previous parameter block (for garbage collection.)
8475	size of parameter block being built
8477	place to keep parameters being made
84DB	used by PTRGET to flag if PARM1 has been search
84DC	stopping point for simple search
84DE	zero if no functions active
84DF	garbage collection temp

84E1	count of active function
84E3	flag telling whether input is scanning first or second time. Ø if first
84E4	save text pointer at start of next
84E6	Ø is 'FOR' is using NEXT code
84E7	use to store the start value of the loop variable
84EB	the line number during scan for 'NEXT'
84ED	Ø = OPTION BASE Ø 1 = OPTION BASE 1
84EE	NON-zero if option base scanned
84EF–8515	reserved
8516	pointer to end line to delete
8518	pointer to start line to delete
851A–851C	reserved
851D	value of first 'swap' variable stored here
8525	Ø = no trace in progress
8527–852F	floating point accumulator
853Ø	overflow print flag
8531	place to store overflow flag
8532	flag to force fixed output
8534–8584	used by math package
8585	RAM EXIT for COMMON
8588	RAM EXIT for CHAIN
858B	RAM EXIT for SYSTEM
858E	RAM EXIT for OPEN
8591	RAM EXIT for FIELD
8994	RAM EXIT for GET
8997	RAM EXIT for PUT
859A	RAM EXIT for CLOSE
859D	RAM EXIT for LOAD
85AØ	RAM EXIT for MERGE
85A3	RAM EXIT for FILES
85A6	RAM EXIT for NAME
85A9	RAM EXIT for KILL
85AC	RAM EXIT for LSET
85AF	RAM EXIT for RSET
85B2	RAM EXIT for SAVE
85B5	RAM EXIT for RESET
85B8	RAM EXIT for PAINT
85BB	RAM EXIT for EOF
85BE	RAM EXIT for LOC
85C1	RAM EXIT for LOF
85C4	RAM EXIT from Execution phase
85C7	RAM EXIT from Screen driver
85CA	RAM EXIT from keyboard driver
85CD	RAM EXIT from RUN
85DØ	RAM EXIT from COPY

85D3–85E1	Reserve
85E2	cursor address
85E4	top of window
85E5	bottom of window
85E6	left margin of window
85E7	width of window
85E8	row number of cursor
85E9	column number of cursor
85EA	pointer to un-shift key code table
85EC	pointer to shift key code table
85EE	pointer to control key code table
85FØ	buffer to hold key pressed
85F1–85F4	reserved
85F5	auto-repeat delay
85F6	auto-repeat period
85F7	reserved
85F9	image of peripheral register
85FA–85FC	reserved
85FD	1 = 8Ø coulmn
	Ø = 4Ø column
85FE	graphic mode
85FF	max width of graphic mode
86Ø1	max height of graphic mode
86Ø2–8603	reserved
86Ø4	copy of character covered by cursor
86Ø5	address curosr flag
86Ø6	cursor flash counter
86Ø7	flash period
86Ø8	seek track
86Ø9	sector
86ØA	current track number
86ØB	pointer to disk buffer
86ØD	primary buffer pointer
86ØF	read table vector
8611	retry counter
8612	controller latch image
8613	disk drive stepper phase image
8614	graphic cursor X-coordinate
8616	graphic cursor Y-coordinate
8617–861C	used by draw line routine
861D	reset routine address
861F	warm start signature byte
	= reset high address + low address +E1H
863C	check sum of tape
863E	length of file name
863F	tape filename
865Ø	temp area for tape filename
8662	used by tape routine
8664	location of video bank

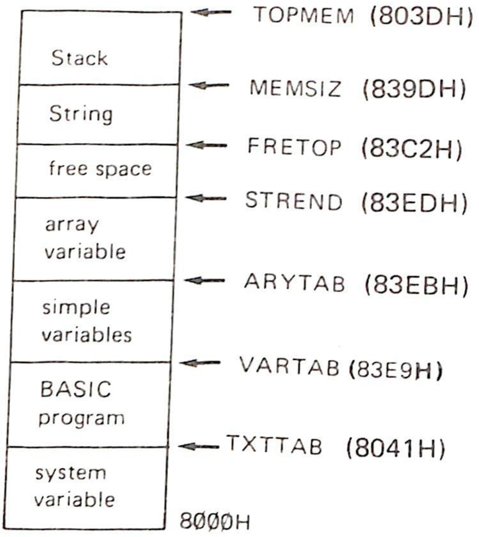
8665	image of I/O Port 4ØH
8666	image of I/O Port 41H
8667	image of I/O Port 42H
8668	image of I/O Port 43H
8669	image of I/O Port 44H
866A	image of I/O Port 45H
866B	used by tape routine
866D–8684	screen line status
8685–8992	disk buffer

USER INTERRUPT ROUTINE

The CPU will be interrupted for every 20ms. The interrupt service routine will push all registers to stack, and call an exit point at location 8012H. When power up this location is initialized to RET. The user can modify this vector to jump to his service routine if he wish to.

BASIC TEXT POINTERS

The BASIC interpreter arrange the memory as follows:



The system variables occupy 8ØØØH to 8993H. They are used by the interpreter as scratch pad.

The starting address of the BASIC program is pointed by TXTTAB (8041H and 8042H). The normal value is 8995H. The ending address is pointed to by VARTAB (83E9H). Whenever you enter a new BASIC statement, the program will grow upward.

Simple variables are stored behind the BASIC program and followed by array variables. The starting addresses are pointed to by VARTAB and ARYTAB respectively. String variable are stored differently. They grow downward from the top of free memory. The bottom of string space is pointed to by FRETOP.

The top of RAM is pointed to by TOPMEM. 512 byte are then reserved as stack for the CPU. End of stack is pointed to by MEMSIZ.

SCREEN CONTROL CODES

The following control codes are supported by the screen driver. The user can use the PRINT CHR$ statement to send these code to the screen.

DECIMAL	HEXADECIMAL	FUNCTION
7	7H	ring the buzzer
8	8H	back space
9	9H	TAB
1Ø	ØAH	line feed
13	ØDH	carriage return
24	18H	cursor up one line
25	19H	cursor right
27	1BH	escape (please refer to chapters on escape — sequences for details)
28	1CH	home cursor
29	1DH	cursor down one line
3Ø	1EH	clear to end of line
31	1FH	clear whole screen
127	7FH	rubout

CHAPTER 5

SYSTEM MONITOR

- System Monitor Overview
- Entering and Leaving the monitor
- Monitor Commands—

:	Fill/Change Memory Content
M	Display Memory Content
T	Transfer range of Memory
I	Display Input Port Value
O	Send Data to output Port
W	Save Memory to Cassette Tape
R	Read Cassette Tape into Memory
L	Disassemble Listing of Memory
Z	Assemble Mnemonic to Memory
G	Execute Maching Language Program
X	Display and Edit CPU Registers
P	Toggle Printer Output
+	Hexadecimal Addition
—	Hexadecimal Subtraction
Q	Leave System Monitor

SYSTEM MONITOR OVERVIEW

Apart from writing and running BASIC programs, the BASIC Interpreter of your computer also provides with a powerful System Monitor. The System Monitor is in itself a program implementing a set of very fundamental commands. It allows you to do very low level jobs such as examine memory locations, change memory location. On the other hand, these commands are powerful enough to take control of your computer. So take great care when using these commands or you will lost your valuable program.

The System Monitor provides you with a number of useful features, such as examining and altering memory locations, moving blocks of memory quickly, disassembled listing of machine language program, direct access of I/O Ports, examine and change CPU registers content, execute machine level program and lots more.

This chapter assumes that you have some background knowledge of machine language programming, in particular, the Z-80 assembly language program. Hexadecimal notations would be used throughout.

If you are not familiar with these, please refer to relevant text before reading this chapter.

ENTERING AND LEAVING THE SYSTEM MONITOR

To enter the system monitor, type:

MON

There is a special prompt sign < MON > which indicates that you are under the control of the System Monitor. Once you have typed MON, your computer assume that anything you typed in thereafter are interpreter as monitor commands. BASIC commands are not accepted until you instruct the computer to go back to BASIC level.

To leave the System Monitor and go back to BASIC command level, type:

Q RETURN

Pressing the RESET button on LASER 500/700 will normally leave the System Monitor and get you back in BASIC.

MONITOR COMMAND

Monitor commands available in your computer are single letter commands preceeded and followed by arguments and parameters.

You have to follow the exact syntax for each Monitor command. Extra spaces, commas or full stop are not allowed. If you type in anything that the Monitor cannot understand, then the Monitor will response by printing a message "??" on the next line to tell you that the command was not accepted.

The System Monitor allows for multiple commands input on a single line, provided that the commands are seperated by at least one space from the adjacent commands. Each command will be executed in turn from left to right. However, if command syntax error is encountered at the middle of the line, a "??" would be printed and all subsequent commands on the right side will be ignored.

When you are under the control of the System Monitor, the screen editing keys, CUROSR KEYS, Escape Sequence are still available to you.

The following notation would be used in the discussion of Monitor commands:

NOTATION	MEANING
bb ee ss	one byte (8 bits) value expressed in two Hexadecimal digits
ssss eeee dddd	two bytes (16 bits) value expressed in four Hexadecimal digits

CHANGE MEMORY CONTENT

```
ssss : bb   bb   bb..........
```

The colon (:) command is for changing memory content from address ssss onwards. The values to be placed in memory are entered on the right of the colon, each separated from the next one by a space character. These values would be allocated in the order that they are entered.

Example: To place Hexadecimal value ØØ, Ø1, Ø2 into memory location AØØØH, AØØ1H, AØØ2H.

```
AØØØ : ØØ Ø1 Ø2
```

FILE MEMORY CONTENT

```
ssss, eeee : bb
```

The colon (:) command can be extended to fill a range of memory with the same value as their content. With the above format. Hexadecimal address ssss through eeee inclusively are filled with the same hexadecimal byte content given by bb.

Example: To fill value of ØØ into memory locations AØØØH through AFFFH (4K byte)

```
AØØØ, AFFF : ØØ
```

DISPLAY MEMORY CONTENT

```
ssss M
```

The M command is for displaying memory content of address ssss on to the screen. The memory content will be converted into 2 digit hexadecimal number and displayed on the next line. The address and a colon would also be outputed along with that value.

Example: To display the content of memory address ABCDH.

```
ABCDM
```

For instant, if ABCDH contains a Ø2, the following line would be outputed by your computer.

```
ABCD : Ø2
```

DISPLAY A RANGE OF MEMORY CONTENT

```
ssss, eeee M
```

The M command can be extended to display a range of memory content from Hexadecimal address ssss through eeee inclusive. Similarly, the memory content are converted into a series of 2 digit hexadecimal numbers and displayed on the next line onwards.

For the sake of clarity, the System Monitor will arrange the display so that the first byte displayed will have address ending with a digit of 'Ø' or '8'. 8Ø columns text mode allows for sixteen bytes maximum on a line and 40 column text mode allows eight bytes on a line.

Example: To display memory content 2ØØEH through 2Ø2FH.

```
2ØØE, 2Ø2FM
```

The screen would show the data similar to the following form:

```
2ØØE : ØØ 14
2Ø1Ø : 1Ø 2F 4C 2B 5D Ø4 Ø8 17
2Ø18 : 43 26 5D 48 12 3Ø Ø2 15
```

40 column is assumed in the above example. If 80 column were used, the second and third line would be combined to one line containing sixteen bytes.

You will soon find the screen editing feature of the computer very helpful in modifying memory contents. Just display the range of the memory to be modified, then move the cursor to any byte you want to change, type in the new value and press RETURN .

TRANSFER A RANGE OF MEMORY

```
ssss, eeee, dddd T
```

The T command is for transferring a range of memory data from source starting address ssss through source ending address eeee inclusive, to destination address dddd. This "transfer" process is actually a "copy" process. The data content at the source is normally not affected, except when the source and destination area overlapps.

Example: To transfer memory content 2ØØØH through 2FFFH to destination starting at AØØØH.

```
2ØØØ, 2FFF, AØØØ  T
```

The transfer process may take certain time (1 to 2 second) to finish if the the block size is large. So be patient when your computer is performing memory transfer.

DISPLAY INPUT PORT VALUE

```
ss    I
```

The I command is for examing the byte value developed at input port given by hexadecimal number ss. The 8 bit value is converted into a 2 digit hexadecimal number and displayed on the next line. Format is similar to displaying of a single memory content.

In order to have meaningful result, users must ensure that their hardware circuitry is properly set up for that Input Port.

Example: To examine the data developed at Input Port 2ØH,

```
2Ø  I
```

Output from the screen may be:

```
2Ø : FF
```

DISPLAY A NUMBER OF INPUT PORT VALUE

```
ss, ee  I
```

The I command is extended to displaying a number of Input Ports all together, from Port number ss to ee inclusively. The 8 bit value developed at these Input Ports are converted to a series of 2 digit hexadecimal numbers and displayed on the next lines. Format is similar to displaying of a range of memory content.

Example: To display the data developed at Input Ports 2AH through 3BH,

```
2A, 3B  I
```

The screen would probably show data in the following form.

```
2A : ØØ   Ø1   Ø3   DF   FF
3Ø : 12   14   56   38   2D   41   32   45
38 : ØF   Ø4   53   CD
```

SEND DATA TO OUTPUT PORTS

```
ss O  bb   bb   bb .....
```

This O command is for sending a series of 8 bit data to Output Ports starting at hexadecimal number ss and then ss + 1, ss + 2 ...... onwards. Each byte of data is separated from the adjacent one by at least one space.

Similar to the case of Input Ports, users must assure that their hardware circuitry is properly set up for the Output Ports.

Example: To send five bytes of data to Output Ports 5ØH, 51H, 52H, 53H, 54H, with values 00, 01, 02, 03, 04.

```
5Ø  O  ØØ  Ø1  Ø2  Ø3  Ø4
```

SAVE A RANGE OF MEMORY TO CASSETTE TAPE

```
ssss, eeee W ''filename''
```

The W command is for saving a range of memory starting from address ssss through eeee to Cassette Tape and assign the name ''FILENAME'' for that cassette file. The file is treated as Binary file, which means a binary image of memory.

Before pressing the RETURN key while typing the above line, you must have started the cassette running in RECORD mode and adjusted input control (if any) to a suitable level. Refer to the chapter on using a cassette for storage if any problem should arise.

Example: To save a range of memory from AØØØH through AFFFH using filename "TEST",

```
AØØØ, AFFF W "TEST"
```

Be patient when your computer is busy generating the cassette signal out.

To escape from writing to Cassette, press Control-C.

READ CASSETTE TAPE BINARY FILE AND PLACE IN MEMORY

```
ssss, eeee R "filename"
```

The R command is for reading a cassette tape file and place the data in memory address ssss through eeee. The cassette file is accepted as Binary file no matter how it was saved onto the cassette tape. The System Monitor would also wait for the filename given by the expression "filename" before loading the data into specified memory locations.

Before pressing the RETURN key while typing the above line, you should have started the cassette tape running in PLAY mode and output adjusted to a suitable level. If any error should arise during the loading process, the same error message would be issued as you were loading BASIC program from cassette.

Example: To load the cassette file TEST into memory location AØØØH through AFFFH,

```
AØØØH, AFFF R "TEST"
```

Please be patient when your computer is reading data from the cassette tape.

To leave cassette reading, press Control-C.

Note: If the starting and ending addresses are omitted, the file will be loaded to where it was saved.

The file saved by the monitor has the binary extension. If it is loaded under BASIC . If will be executed automatically.

DISASSEMBLE LISTING

```
ssss     L
```

The L command is for obtaining disassemble listing of memory content starting from location ssss. These memory content will be treated as Z-80 instructions nomatter what nature they are. Therefore, the disassembled listing may not be meaningful to you.

The address, content, the disassembled opcode and operand would be listed starting from the next line until they fills one screen.

Example: To disassemble memory starting from AØØØH,

```
AØØØ   L
```

The screen would be filled with address value, memory content and disassembled opcode mnemonic similar to following format:

```
A000  : 3A  10  E0                   LD A, (E010)
A003 : 3A  00  E0                    LD A, (E000)
A006 : B7                            OR A
A007 : F2  03  A0                    JP P, A003
A00A : 47                            LD B, A
A00B : 3A  10  E0                    LD A, (E010)
A00E : 78                            LD A, B
A00F : E6  7F                        AND 7F
A011 : EF  0D                        CP 0D
A013 : C2  03  A0                    JP NZ, A003


Address     Content                  Opcode      Operand
```

ssss, eeee L

The L command can be extended to disassemble a range of memory from ssss through eeee.

ASSEMBLE NMEMONIC

```
ssss  Z
```

The Z command is for assembling Z-80 instruction mnemonic and place the object code starting from location ssss onwards. This allows the user to write simple, short assembly language program directly into your computer's memory. However, this is not a means to write large assembly language program. Standard assemblers are recommanded in that case.

While entering the Z-80 instruction mnemonics code, you must follow exactly the assembler syntax requirements. For instance, the opcode field must be seperated from the operand field by at least one space.

After you have entered one line of instruction mnemonics and the return key is pressed, the System Monitor will do the assemble for that line. Address for that instruction is displayed, followed by hexadecimal content and then your instruction mnemonic. The cursor will go to the next line and wait for further input of instruction. The process will repeat for every line that you type in, until you leave Assemble mode by pressing control-C. Consecutive address will be assigned for subsequent instructions.

Example: To enter the instruction LD B, A and place the assembled code starting from address AØØØH.

```
AØØØZ                                    (You type in)
LD  B, A                                 (You type in)
AØØØ : 78             LD B,A             (computer response)
    :
    :
Control-C                                (To leave assemble)
```

Note: Hexadecimal number starting with 'A' to 'F' should be preceded by a 'Ø'. For example, IN A, (C) and IN A, (ØC) will be treated as different instruction.

START PROGRAM EXECUTION

```
ssss G
```

The G command allows the user to execute a Z-80 machine language program starting at location ssss. Users must ensure for themselves that meaningful code are already placed in the assoicated memory address. Otherwise, your computer may go to a dead lock condition. Only System Reset or power down can bring you back the control.

The System Monitor treats your machine language program as subroutine when the G command is executed. So you will usually gȯ back to Monitor command level if the last instruction in your program is a RET (Return from Subroutine), provided that the stack is in proper condition.

Example: To start exectuion at memory location AØØØH in which you have placed with a meaningful program,

```
AØØØ  G
```

DISPLAY AND EDIT CPU INTERNAL REGISTERES

```
X
```

The X command allows the user to examine and edit the Z-80 CPU internal registers. These new values are not immediately loaded into the CPU, but they will be put aside in RAM locations and loaded into CPU when the G (Execute Program) is issued.

When the X command is issued, the System Monitor response by displaying the currently assigned CPU register values in the order of AF, BC, DE ......SP. Then you are allowed to type in the new values, preceed with the colon (:) sign, also in the same order. These new value will be used in the next G command.

```
     X                                            (You typed in)
AF=ØØØØ  BC=ØØØØ  DE=1111  HL=1111  IX=1111
IY=ØØØØ  SP=ØØØØ                                  (Computer response)
: FF  ØØ  23                                      (You typed in)
X                                                 (You typed in)
AF=ØØFF  BC=ØØ 23 DE=1111  HL=1111  IX=1111
IY=ØØØØ·SP=ØØØØ                                   (Computer response)
```

SEND OUTPUT TO PRINTER/DISABLE PRINTER OUTPUT

```
P
```

The P command allows the user to obtain printed output at the printer when running under the control of the System Monitor. After P command is first issued, anything displayed on the screen will be sent to the printer as well. This P command also acts as a toggle switch for disabling the Printer output , that is, the next time you issue P, the Printer output will be stopped. You have to ensure for yourself that a printer is properly installed for your computer.

When you leave the Sytem Monitor via the Q command or by pressing RESET, the printer output will be disabled automatically nomatter whether it has been turned on by P command or not.

ADDITION SUBTRACTION OF FOUR DIGIT HEXADECIMAL NUMBERS

```
Addition  :  ssss, eeee +
Substraction  :  ssss, eeee –
```

The command +, – are implemented to allow for simple arithmetic between two numbers expressed in four digit hexadecimal form. The value represented by ssss and eeee are added, result also represented in four digit hexadecimal form. Any overflow or carry is neglected. The result is displayed on the next line which follows the command line. With the above format, eeee is added to ssss, and eeee is subtracted from ssss.

These hexadecimal arithmetic are usually used for calculating JUMP Address offsets while you are assembling your machine language program.

Example: To add/subtract two numbers
2Ø5FH, Ø14AH

```
295F , Ø14A +
21A9

2Ø5F, Ø14A –
1F15
```

APPENDIX

APPENDIX A

BASIC ERROR MESSAGE AND ERROR CODES

CODE	NUMBER	MESSAGE
NF	1	NEXT without FOR A variable in a NEXT statement does not correspond to any previously executed, unmatched FOR statement variable.
SN	2	Syntax error A line is encountered that contains some incorrect sequence of characters (such as unmatched parenthesis, a misspelled command or statement, incorrect punctuation, etc.)
RG	3	Return without GOSUB A RETURN statement is encountered for which there is no previous, unmatched GOSUB statement.
OD	4	Out of data A READ statement is executed when there are no DATA statements with unread data remaining in the program.
FC	5	Illegal function call A parameter that is out of range is passed to a math or string function. An FC error may also occur as the result of: 1. a negative or unreasonably large subscript 2. a negative or zero argument with LOG 3. a negative argument to SQR 4. a negative mantissa with a non-integer exponent 5. a call to a USR function for which the starting address has not yet been given 6. an improper argument to MID$, WAIT, LEFT$, RIGHT$, INP, OUT. PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, or ON . . . GOTO.

OV 6 Overflow

The result of a calculation is too large to be represented in BASIC number format. If underflow occurs, the result is zero and execution continues without an error.

OM 7 Out of memory

A program is too large, has too many FOR loops or GOSUBs, too many variables, or expressions that are too complicated.

UL 8 Undefined line number

A line reference in a GOTO, GOSUB, IF . . . THEN . . . ELSE or DELETE is to a nonexistent line.

BS 9 Subscript out of range

An array element is referenced either with a subscript that is outside the dimensions of the array, or with the wrong number of subscripts.

DD 10 Duplicate Definition

Two DIM statements are given for the same array, or a DIM statement is given for an array after the default dimension of 10 has been established for that array.

/0 11 Division by zero

A division by zero is encountered in an expression, or the operation of involution results in zero being raised to a negative power. Machine infinity with the sign of the numerator is supplied as the result of the division, or positive machine infinity is supplied as the result of the involution, and execution continues.

ID 12 Illegal direct

A statement that is illegal in direct mode is entered as a direct mode command.

TM	13	Type mismatch A string variable name is assigned a numeric value or vice versa; a function that expects a numeric argument is given a string argument or vice versa.
OS	14	Out of string space String variables have caused BASIC to exceed the amount of free memory remaining. BASIC will allocate string space dynamically, until it runs out of memory.
LS	15	String too long An attempt is made to create a string more than 255 characters long.
ST	16	String formula too complex A string expression is too long or too complex. The expression should be broken into smaller expressions.
CN	17	Can't continue An attempt is made to continue a program that: 1. has halted due to an error, 2. has been modified during a break in execution, or 3. does not exist.
UF	18	Undefined user function A USR function is called before the function definition (DEF statement) is given.
	19	No RESUME An error trapping routine is entered but contains no RESUME statement.
	20	RESUME without error A RESUME statement is encountered before an error trapping routine is entered.

21 Unprintable error
An error message is not available for the error condition which exists. This is usually caused by an ERROR with an undefined error code.

22 Missing operand
An expression contains an operator with no operand following it.

23 Line buffer overflow
An attempt is made to input a line that has too many characters.

26 FOR without NEXT
A FOR was encountered without a matching NEXT.

29 WHILE without WEND
A WHILE statement does not have a matching WEND.

30 WEND without WHILE
A WEND was encountered without a matching WHILE.

APPENDIX B

DERIVED MATHEMATICAL FUNCTIONS

Functions that are not intrinsic to the BASIC interpreter may be calculated as follows:

FUNCTION	EQUIVALENT FUNCTION
SECANT	SEC(X)=1/COS(X)
COSECANT	CSC(X)=1/SIN(X)
COTANGENT	COT(X)=1/TAN(X)
INVERSE SINE	ARCSIN(X)=ATN(X/SQR(–X*X+1))
INVERSE COSINE	ARCCOS(X)=–ATN(X/SQR(–X*X+1))+1.5708
INVERSE SECANT	ARCSEC(X)=ATN(X/SQR(X*X–1))+(SGN(X)–1)* 1.5708
INVERSE COSECANT	ARCCSC(X)=ATN(X/SQR(X*X–1))+(SGN(X)–1)*1.5708
INVERSE COTANGENT	ARCCOT(X)=ATN(X)+1.5708
HYPERBOLIC SINE	SINH(X)=(EXP(X)–EXP(–X))/2
HYPERBOLIC COSINE	COSH(X)=(EXP(X)+EXP(–X))/2
HYPERBOLIC TANGENT	TANH (X)=EXP(–X)/(EXP(X)+EXP(–X))*2+1
HYPERBOLIC SECANT	SECH(X)=2/(EXP(X)+EXP(–X))
HYPERBOLIC COSECANT	CSCH(X)=2/(EXP(X)–EXP(–X))
HYPERBOLIC COTANGENT	COTH(X)=EXP(–X)/(EXP(X)–EXP(–X))*2+1
INVERSE HYPERBOLIC SINE	ARCSINH(X)=LOG(*X+1))
INVERSE HYPERBOLIC COSINE	ARCCOSH(X)=LOG(X*X–1))
INVERSE HYPERBOLIC TANGENT	ARCTANH(X)=LOG((1+X)/(1–X))/2
INVERSE HYPERBOLIC SECANT	ARCSECH(X)=LOG((SQR(–X*X+1)+1)/X)
INVERSE HYPERBOLIC COSECANT	ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X
INVERSE HYPERBOLIC COTANGENT	ARCCOTH(X)=LOG((X+1)/(X–1))/2

APPENDIX C

ASCII CHARACTER CODE, CHART

ASCII Character Table

Decimal:		128	144	160	176	192	208	224	240
Hexadecimal:		$80	$90	$A0	$B0	$C0	$D0	$E0	$F0
0	$0	nul	dle		0	@	P	`	P
!	$1	soh	dc1	!	1	A	Q	a	q
2	$2	stx	dc2	''	2	B	R	b	r
3	$3	etx	dc3	=	3	C	S	c	s
4	$4	eot	dc4	$	4	D	T	d	t
5	$5	enq	nak	%	5	E	U	e	u
6	$6	ack	syn	&	6	F	V	f	v
7	$7	bel	etb	'	7	G	W	g	w
8	$8	bs	can	(	8	H	X	h	x
9	$9	ht	em	)	9	I	Y	i	y
10	$A	lf	sub	*	:	J	Z	j	z
11	$B	vt	esc	+	;	K	[	k	{
12	$C	ff	fs	,	<	L	/	l	|
13	$D	cr	gs	—	=	M	]	m	}
14	$E	so	rs	.	>	N	^	n	~
15	$F	si	us	/	?	O	_	o	rub

APPENDIX D

ESCAPE SEQUENCE FUNCTION

ESCAPE SEQUENCE FUNCTION

The screen driver of the BASIC Interpreter in your computer is able to perform some special functions via the Escape Sequences. These functions might be invoked by directly typing in from the keyboard, or by the PRINT command.

The following table lists the code to be followed by the ESC code, and the function performed.

Code followed by ESC (Hex.)	Function
AØH	Clear to end of page
41H	Set normal text mode
42H	Set inverse text mode
A1H	Address cursor
43H	Set keyboard silent
44H	Set keyboard to beep
A2H	Invoke 80 column display
A3H	Invoke 40 column display

Note: The Address cursor function is for placing the cursor to a specific (X, Y) position, where X is the column value and Y is the row value.
An offset of 32 should be added to X and Y.
An example of Address cursor is:

```
PRINT CHR$ (27); CHR$(161); CHR$ (32+X);
CHR$ (32+Y)
```

X is always followed by Y.

APPENDIX E

KEY CODE ASSIGNMENT

		NORMAL			SHIFT	
KEY		CAP LOCK ON	CAP LOCK OFF	CTRL	CAP LOCK ON	CAP LOCK OFF
SPACE		20	20	1D	20	20
1	!	31	31	(No code)	21	21
2	@	32	32	09	40	40
3	=	33	33	1E	23	23
4	$	34	34	1C	24	24
5	%	35	35	1F	25	25
6	^	36	36	07	5E	5E
7	&	37	37	7B	26	26
8	*	38	38	7D	2A	2A
9	(	39	39	5B	28	28
0	)	30	30	5D	29	29
-	—	2D	2D	5C	5F	5F
=	+	3D	3D	7C	2B	2B
\	¦	5C	5C	5C	7C	7C
[	{	5B	5B	5B	7B	7B
]	}	5D	5D	5D	7D	7D
;	:	3B	3B	15	3A	3A
"	'	22	22	7F	2C	2C
~	`	7E	7E	FF	60	60
,	<	2C	2C	08	3C	3C
.	>	2E	2E	19	3E	3E
/	?	2F	2F	18	3F	3F
u	£	DE	DE	DE	DF	DF

	NORMAL			SHIFT	
KEY	CAP LOCK ON	CAP LOCK OFF	CTRL	CAP LOCK ON	CAP LOCK OFF
A	41	61	AB	61	41
B	42	62	9F	62	42
C	43	63	03	63	43
D	44	64	86	64	44
E	45	65	83	65	45
F	46	66	8D	66	46
G	47	67	89	67	47
H	48	68	A0	68	48
I	49	69	85	69	49
J	4A	6A	8F	6A	4A
K	4B	6B	8A	6B	4B
L	4C	6C	93	6C	4C
M	4D	6D	AA	6D	4D
N	4E	6E	94	6E	4E
O	4F	6F	88	6F	4F
P	50	70	91	70	50
Q	51	71	82	71	51
R	52	72	8E	72	52
S	53	73	DD	73	53
T	54	74	DB	74	54
U	55	75	8B	75	55
V	56	76	9E	76	56
W	57	77	DA	77	57
X	58	78	99	78	58
Y	59	79	A2	79	59
Z	5A	7A	97	7A	5A

	NORMAL			SHIFT	
KEY	CAP LOCK ON	CAP LOCK OFF	CTRL	CAP LOCK ON	CAP LOCK OFF
F1	E0	E0	F4	EA	EA
F2	E1	E1	F5	EB	EB
F3	E2	E2	F6	EC	EC
F4	E3	E3	D7	ED	ED
F5	E4	E4	F8	EE	EE
F6	E5	E5	F9	EF	EF
F7	E6	E6	FA	F0	F0
F8	E7	E7	FB	F1	F1
F9	E8	E8	FC	F2	F2
F10	E9	E9	FD	F3	F3
GRAPH	FF	FF	FF	FF	FF
↑	18	18	18	18	18
↓	1D	1D	1D	1D	1D
←	08	08	08	08	08
→	19	19	19	19	19
DEL	7F	7F	7F	7F	7F
INS	15	15	15	15	15
DEL LINE	1E	1E	1E	1E	1E
CLS HOME	1C	1C	1C	1F	1F
BS	08	08	08	08	08
RETURN	0D	0D	0D	0D	0D
ESC	1B	1B	1B	1B	1B
TAB	09	09	09	09	09

Note: CTRL 1 ! , as well as the CAP LOCK keys generates no code. They only perform toggling of CAP LOCK conditions.

APPENDIX F

DEFAULT FUNCTION KEY DEFINATION (for LASER 500/700 only)

Function Key No.	Code Generated	Function Description
F1	(1FH) LIST (CR)	Clears the text screen, LISTs the BASIC Program
F2	RUN(1EH) (CR)	Clears to end of current line, RUNs the BASIC program
F3	TEXT 40 (1EH) (CR)	Clears to end of current line, FLIPS to 80 column text screen
F4	TEXT 80 (1EH) (CR)	Clears to end of current line, FLIPS to 80 column text screen
F5	COLOR 15, 0, 0 (1EH) (CR)	Clears to end of current line, Sets color to be white Foreground , Black Background and Black Backdrop
F6	COLOR 1, 9, 1, (1EH) (CR)	Clears to end of current line, Sets color to be blue Foreground, light blue Background and blue Backdrop
F7	PRINT CHR$(27); "A" (1EH) (CR)	Clears to end of line, Sets NORMAL display mode
F8	PRINT CHR$(27); "B" (1EH) (CR)	Clears to end of line, Sets INVERSE display mode
F9	KEY S(1EH) (CR)	Clear to end of line, Sets keyboard silent
F10	KEY B(1EH) (CR)	Clears to end of line, Sets keyboard beep

Note: The characters enclosed on parenthesis on the "Code Generated" column are special control codes embedded in the Function Key Definition. For example, (1FH) means hexadecimal 1FH is sent to the screen driver and (CR) means Carriage Return code (ØD) sent to the screen driver. The other code (1EH) is normally necessary because any garbage text on the right of the current line might be interpreted as part of the Function Key Definition. This code (1EH) will clear to the end of the current line, thus avoiding the "SYNTAX ERROR" that would be produced.

APPENDIX G

GRAPHICS CHARACTER SET

DECIMAL	HEX	GRAPHIC CHARACTER	KEY PRESS
128	80		@
129	81		A
130	82		B
131	83		C
132	84		D
133	85		E
134	86		F
135	87		G
136	88		H
137	89	♠	I
138	8A	♥	J
139	8B	♣	K
140	8C	♦	L
141	8D	π	M
142	8E	µ	N
143	8F	£	O
144	90	α	P
145	91	▖	Q
146	92	▗	R
147	93	▄	S
148	94	▝	T
149	95	▐	U
150	96	▞	V
151	97	▟	W
152	98	▘	X
153	99	▚	Y
154	9A	▌	Z
155	9B	▙	[
156	9C	▀	\
157	9D	▜	]
158	9E	▛	^
159	9F	█	—

Note: The Decimal or Hex value shown on the left hand columns of the above table is the value to be used in a BASIC command to generate graphic characters on the screen, you should set or reset the MSB of the byte, depending on INVERSED graphics characters or NORMAL graphics characters respectively. To generate the graphics characters directly on the screen, or by typing from the keyboard, please refer to the "Key Press" on the right most column. LASER 350 users may need to use CONTROL key combinations.

APPENDIX H

BASIC TOKEN TABLE

BASIC TOKEN TABLE

BASIC commands and functions are stored as tokens instead of as individual text. These tokens may consist of one to two bytes.

Tokens for commands and functions are shown separately on the following tables.

TOKEN TABLE (COMMANDS)

81	END	97	WAIT	AD	DEFSTR
82	FOR	98	DEF	AE	DEFINT
83	NEXT	99	POKE	AF	DEF SNG
84	DATA	9A	CONT	B0	DEF DEL
85	INPUT	9B	CSAVE	B1	LINE
86	DIM	9C	CLOAD	B2	SET
87	READ	9D	OUT	B3	RES
88	LET	9E	LPRINT	B4	WHILE
89	GOTO	9F	LLIST	B5	WEND
8A	RUN	A0	CLS	B6	CALL
8B	IF	A1	WIDTH	B7	WRITE
8C	RESTORE	A2	ELSE	B8	COMMON
8D	GOSUB	A3	TRON	B9	CHAIN
8E	RETURN	A4	TROFF	BA	OPTION
8F	REM	AS	SWAP	BB	RANDOMIZE
9Ø	STOP	A6	ERASE	BC	TEXT
91	PRINT	A7	COLOR	BD	SYSTEM
92	CLEAR	A8	ERROR	BE	GR
93	LIST	A9	RESUME	BF	OPEN
94	NEW	AA	DELETE		
95	ON	AB	AUTO		
96	NULL	AC	RENUM		

CØ	FIELD	D6	MON	EC	
C1	GET	D7		ED	
C2	PUT	D8		EE	
C3	CLOSE	D9		EF	>
C4	LOAD	DA	TO	FØ	=
C5	MERGE	DB	THEN	F1	<
C6	FILES	DC	TAB	F2	+
C7	NAME	DD	STEP	F3	—
C8	KILL	DE	USR	F4	*
C9	LSET	DF	FN	F5	/
CA	RSET	E0	SPC	F6	^
CB	SAVE	E1	NOT	F7	AND
CC	RESET	E2	ERL	F8	OR
CD	KEY	E3	ERR	F9	XOR
CE	SOUND	E4	STRING	FA	EQV
CF	CRUN	E5	USING	FB	INP
D0	VERIFY	E6	INSTR	FC	MOD
D1	MOTOR	E7		FD	\
D2	COPY	E8	VARPTR	FE	
D3	MOVE	E9	INKEY	FF	
D4	DRAW	EA	POINT		
D5	PAINT	EB			

TOKEN TABLE (FUNCTIONS)

81	LEFT$	97	PEEK	AD	CVD
82	RIGHT$	98	SPACE	AE	
83	MID$	99	OCT$	AF	EOF
84	SGN	9A	HEX$	BØ	LOC
85	INT	9B	LPOS	B1	LOF
86	ABS	9C	CINT	B2	MKI
87	SQR	9D	CSNG	B3	MKS
88	RND	9E	CDBL	B4	MKD
89	SIN	9F	FIX	B5	
8A	LOG	AØ	JOY	B6	
8B	EXP	A1		B7	
8C	COS	A2		B8	
8D	TAN	A3		B9	
8E	ATN	A4		BA	
8F	FRE	A5		BB	
9Ø	INP	A6		BC	
91	POS	A7		BD	
92	LEN	A8		BE	
93	STR$	A9		BF	
94	VAL	AA			
95	ASC	AB	CVI		
96	CHR$	AC	CVS		

CØ
C1
C2
C3
C4
C5
C6
C7
C8
C9
CA
CB
CC
CD
CE
CF
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
DA
DB
DC
DD
DE
DF
E0
E1
E2
E3
E4
E5
E6
E7
E8
E9
EA
EB
EC
ED
EE
EF
F0
F1
F2
F3
F4
F5
F6
F7
F8
F9
FA
FB
FC
FD
FE
FF

Note: Tokens for some commands and functions enclosed in the rectangular box are disk commands or functions. They can be executed only after you have booted the Disk Operating System. (DOS)

The above table also represents a set of reserved words for the BASIC Interpreter. This implies that you cannot use any of these words for a variable name. Otherwise, a "SYNTAX ERROR" will be returned.

Tokens for BASIC commands are one byte in length starting from 81H onwards. Tokens for BASIC Functions are two bytes in length, in which the first byte is always FFH.

APPENDIX I

COLOR CODE

Table I—1 shows the color code for the BASIC command: COLOR I, J, K where I, J, K are decimal values (0—15) respresenting the Foreground, Background and Backdrop colors respectively.

Code	Color	Code	Color
Ø	black	8	light grey
1	blue	9	light blue
2	green	1Ø	light green
3	cyan	11	light cyan
4	red	12	light red
5	magenta	13	light magenta
6	yellow	14	light yellow
7	grey	15	white

TABLE I—1

Table I—2 shows the 4 bit color code for bit pattern in the from of

Br	R	G	B

where Br, R, G, B are binary values Ø or 1 representing the Brightness, Red, Green and Blue components respectively.

Br	R	G	B	Color	Br	R	G	B	Color
0	0	0	0	black	1	0	0	0	light grey
0	0	0	1	blue	1	0	0	1	light blue
0	0	1	0	green	1	0	1	0	light green
0	0	1	1	cyan	1	0	1	1	light cyan
0	1	0	0	red	1	1	0	0	light red
0	1	0	1	magenta	1	1	0	1	light magenta
0	1	1	0	yellow	1	1	1	0	light yellow
0	1	1	1	grey	1	1	1	1	white

TABLE I—2

It can be easily seen that the color codes used in BASIC commands are indeed the decimal equivalent of the 4 bit number formed from Br, R, G, B.

APPENDIX J

SUMMARY OF BASIC COMMANDS AND FUNCTIONS

Commands	Meaning	Page no.
ABS	Absolute magnitude	62
AND	Relational and logical operation	80
ASC	ASCII value in decimal	71
ATN	Arctangent function	62
AUTO	Automatic line number generation	136
CALL	Call machine code subroutine	158
CDBL	Convert to double precision	151
CHR$	Character represented by ASCII code	71
CINT	Convert to integer	151
CLEAR	Clear space for variable	59
CLOAD	Load program from Tape	123
CLS	Clear text screen	41
COLOR	Set color	107
CONT	Continue program execution	58
COS	Cosine function	62
CRUN	Load and run program from tape	125
CSAVE	Save program on tape	124
CSNG	Convert to single precision	151
CVD	Convert string to double precision number	152
CVI	Convert string to integer number	152
DATA	Assign data for READ command	98
DEF FN	Define function	65
DEFDBL/ INT/SNG/STR	Define data type	150
DEF USR	Define address for user's Assembly Language Subroutine	156
DELETE	Delete program lines	55
DIM	Assign size of array	95
DRAW	Draw straight line	112
END	Terminate program	58
ERASE	Erase arrays	96
ERL	Error line number	139
ERR	Error code	139
ERROR	Error condition	140
EXP	Exponential function	62
FIX	Truncated integer part of number	62
FOR . . . NEXT	Perform looping	86
FRE	Free up memory	137
GOSUB	Go to a subroutine	90
GOTO	Jump to specified line number	57

GR	Invoke graphic mode	104
HEX$	String of hexadecimal value of number	72
IF . . . THEN . . GOTO	Conditional branching	78
IF . . .THEN . . . ELSE	Conditional branching	78
INKEY$	Check keyboard	72
INP	Return content read from I/O Port	141
INPUT	Ask for value or string input	52
INPUT$	Ask for string input	74
INSTR	Search for string occurance in another string	73
INT	Get integer part	62
KEY	Function key/auto-repeat key setting	142
LEFT$	Left characters of a string	70
LEN	Length of a string	69
LET	Assign value	48
LINE INPUT	Ask for input a whole line	74
LIST	List program lines on screen	54
LLIST	List program lines to printer	134
LOG	Natural Logarithm	62
LPRINT	Print to Printer	134
LPRINT USING	Print to printer with format	134
MID$	Middle characters of a string	70
MKD$	Convert double precision number to string	152
MKI$	Convert integer to string	152
MKS$	Convert single precision number to string	152
MOD	MODULUS of a number	44
MON	Enter System Monitor	228
MOVE	MOVE without drawing	113
NEW	Clear BASIC program from memory	52
NULL	Set null characters to be printed	129
OCT$	String of octal value of number	71
ON ERROR GOTO	Error trapping entry point	83
ON . . . GOSUB	Conditional branching to subroutine	82
ON . . . GOTO	Conditional branching to specified line	82
OPTION BASE	Set minimum value for array subscripts	96
OR	Relational and logical operation	80
OUT	Sent data to I/Oport	141
PEEK	Return decimal value stored at memory location	102

POINT	Return color code	112
POKE	Load data to memory location	102
POS	Returns cursor position	138
PRINT	Print number or string on screen	35
PRINT #	Print data to cassette	125
PRINT USING	Print with certain format	130
RANDOMIZE	Make random number reseed every time	64
READ	Assign value in DATA statement to variable	98
REM	Remark	52
RENUM	Renumber BASIC program	136
RES	Clear a point on graphic screen	111
RESTORE	Recover the same DATA	98
RESUME	To continue program after recovery from error	83
RETURN	Terminate subroutine	90
RIGHT$	Right characters of a string	70
RND	Generate random number	62
RUN	Starts program execution	53
SET	Plot a dot on the graphic screen	111
SGN	Sign of a number	62
SIN	Sine function	62
SPACE$	Assign a string of some space	129
SPC	Prints a number of blanks	129
SQR	Square root of a number	62
STOP	Terminate program execution	58
STR$	String representation of a number	69
STRING$	String of a series of same characters	72
SWAP	Exchange value of variables	138
TAB	Spaces to specified column on the screen	128
TAN	Tangent Function	62
TROFF	Trace mode turn off	139
TRON	Trace mode turn on	139
USR	Call user's Assembly Language Subroutine	157
VAL	Numerical value of string	69
VARPTR	Address of variable	158
WAIT	Wait for pattern set up at I/O Port	141
WHILE . . . WEND	Execute a loop as long as condition meet	88
WIDTH	Set printable width of screen	128
WRITE	Print with delimitors	132

APPENDIX K

SUMMARY OF SYSTEM MONITOR COMMANDS

To enter System Monitor:MON
To leave System Monitor:Q

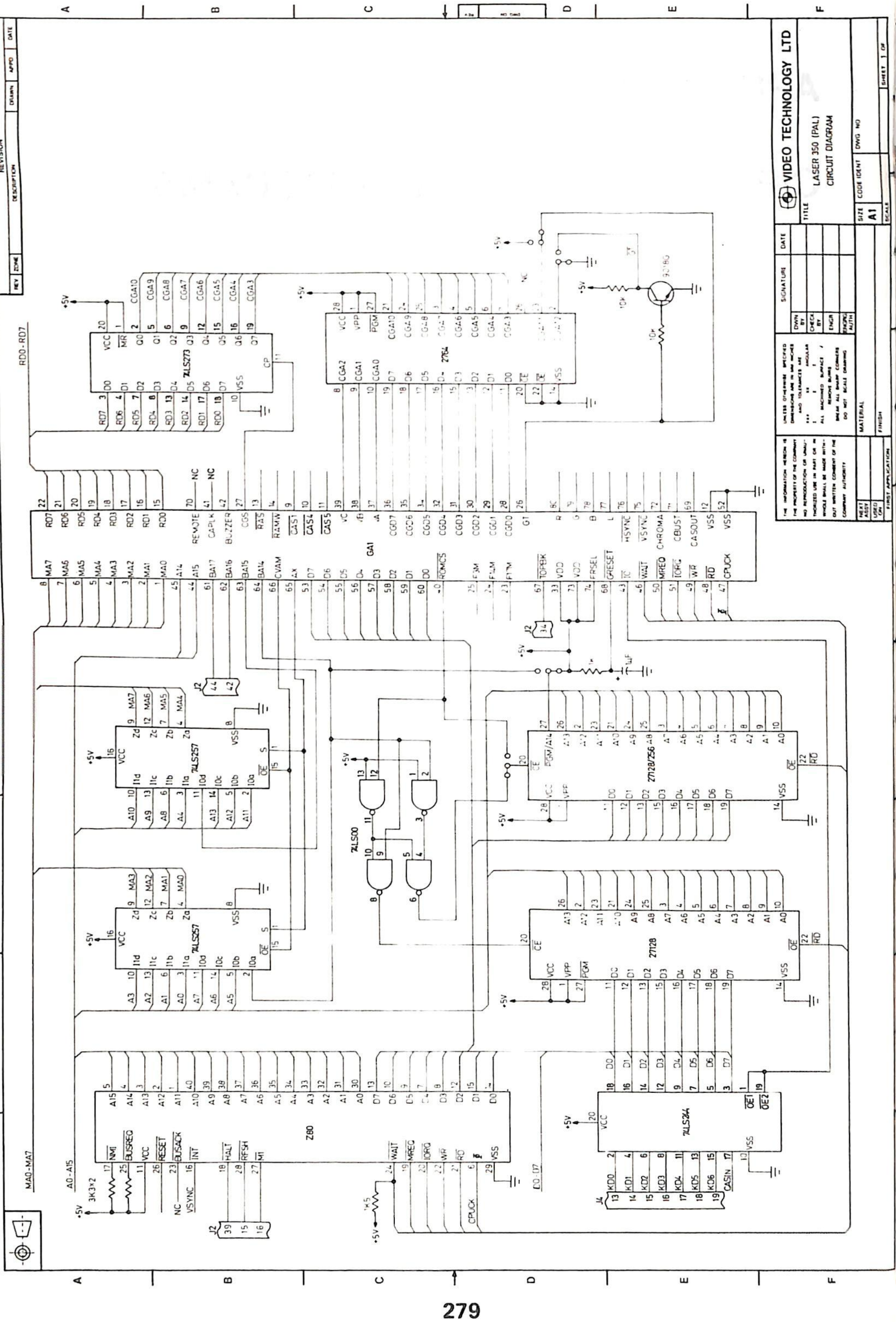
Monitor Command Syntax	Function
ssss : bb bb bb	Change memory content from ssss onwards
ssss, eeee : bb	Fill a range of memory with the same byte bb, from ssss to eeee
ssss M	Display memory content of location ssss
ssss, eeee M	Display a range of Memory content from ssss to eeee
ssss, eeee, dddd T	Transfer a range of memory data from ssss, through eeee, to new area starting dddd
ss I	Display the byte value of Input Port ss
ss, ee I	Display the byte values of Input Port number ss to ee
ss O bb bb bb	Send a series of bytes to Output Ports number ss, ss + 1, ss + 2 ..... Sequentially
ssss, eeee W "filename"	Saving a range of memory from location ssss through eeee, onto cassette tape using the filename
ssss, eeee R "filename"	Load a cassette file with the specified filename and place it to memory starting ssss, ending eeee
ssss L	Disassemble listing from memory location ssss onwards.
ssss, eeee L	Disassemble listing from memory ssss through eeee

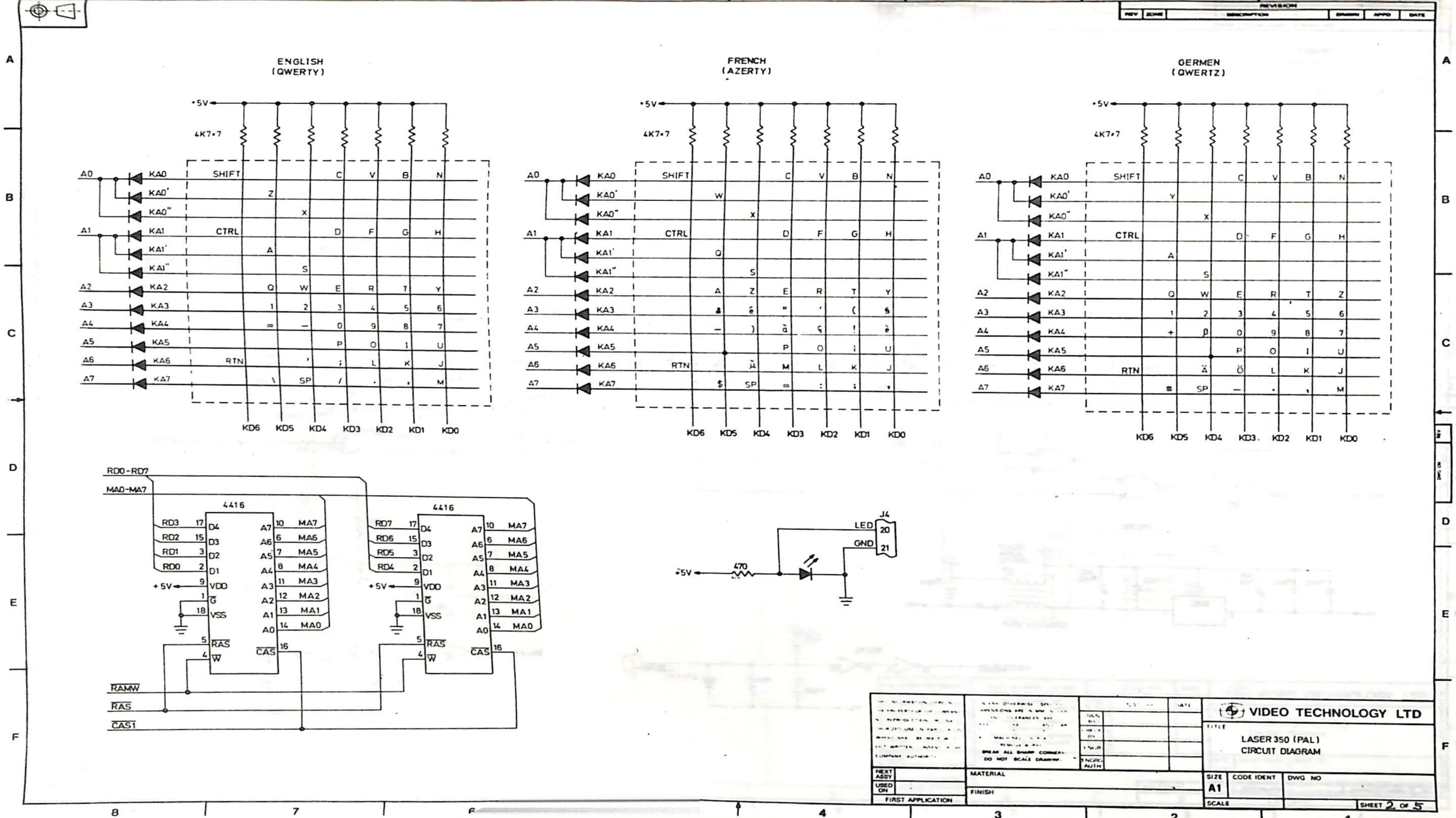
ssss Z	Assemble Z-80 instruction mnemonic and place in memory starting from location ssss
ssss G	Execute machine language program starting at location ssss
X	Display and Edit CPU Registers
P	Turn ON/OFF printer output
ssss, eeee +	Hexadecimal addition of ssss and eeee
ssss, eeee —	Hexadecimal subtration of eeee from ssss
Q	Quit system monitor and return to BASIC

Note: bb, ee, ss are one byte value expressed in two hexadecimal digits
ssss, eeee, dddd are two bytes value expressed in four hexadecimal digits

APPENDIX L

CIRCUIT SCHEMATIC DIAGRAMS


VIDEO TECHNOLOGY LTD
TITLE
LASER 350 (PAL)
CIRCUIT DIAGRAM
SIZE
A1
CODE IDENT
DWG NO
SCALE
SHEET 1 OF
REVISION
REV
ZONE
DESCRIPTION
DRAWN
APPD
DATE
SIGNATURE
DWN BY
CHECK BY
ENGR
ENGRG AUTH
MATERIAL
FINISH
NEXT ASSY
USED ON
FIRST APPLICATION
Z80
GA1
74LS273
2764
74LS257
74LS257
74LS00
27128/256
27128
74LS244
9018G
RD0-RD7
MA0-MA7
A0-A15
D0-D7
3K3×2
1K5
10K
10K
1K
1uF
+5V
NC
J2
J4
CPUCK
VSYNC
HSYNC
CHROMA
CBUST
CASOUT
CASIN
REMOTE
CAPLK
BUZZER
TOPBK
FRSEL
GRESET
ROMCS
KD0
KD1
KD2
KD3
KD4
KD5
KD6



ENGLISH (QWERTY)

	KD6	KD5	KD4	KD3	KD2	KD1	KD0
KA0	SHIFT			C	V	B	N
KA0'		Z					
KA0"			X				
KA1	CTRL			D	F	G	H
KA1'		A					
KA1"			S				
KA2		Q	W	E	R	T	Y
KA3		1	2	3	4	5	6
KA4		=	–	0	9	8	7
KA5				P	O	I	U
KA6	RTN		'	;	L	K	J
KA7		\	SP	/	.	,	M

FRENCH (AZERTY)

	KD6	KD5	KD4	KD3	KD2	KD1	KD0
KA0	SHIFT			C	V	B	N
KA0'		W					
KA0"			X				
KA1	CTRL			D	F	G	H
KA1'		Q					
KA1"			S				
KA2		A	Z	E	R	T	Y
KA3		&	é	"	'	(	§
KA4		–	)	à	ç	!	è
KA5				P	O	I	U
KA6	RTN		µ	M	L	K	J
KA7		$	SP	=	:	;	,

GERMEN (QWERTZ)

	KD6	KD5	KD4	KD3	KD2	KD1	KD0
KA0	SHIFT			C	V	B	N
KA0'		Y					
KA0"			X				
KA1	CTRL			D	F	G	H
KA1'		A					
KA1"			S				
KA2		Q	W	E	R	T	Z
KA3		1	2	3	4	5	6
KA4		+	β	0	9	8	7
KA5				P	O	I	U
KA6	RTN		Ä	Ö	L	K	J
KA7		#	SP	–	.	,	M

MATERIAL

FINISH

NEXT ASSY

USED ON

FIRST APPLICATION

VIDEO TECHNOLOGY LTD

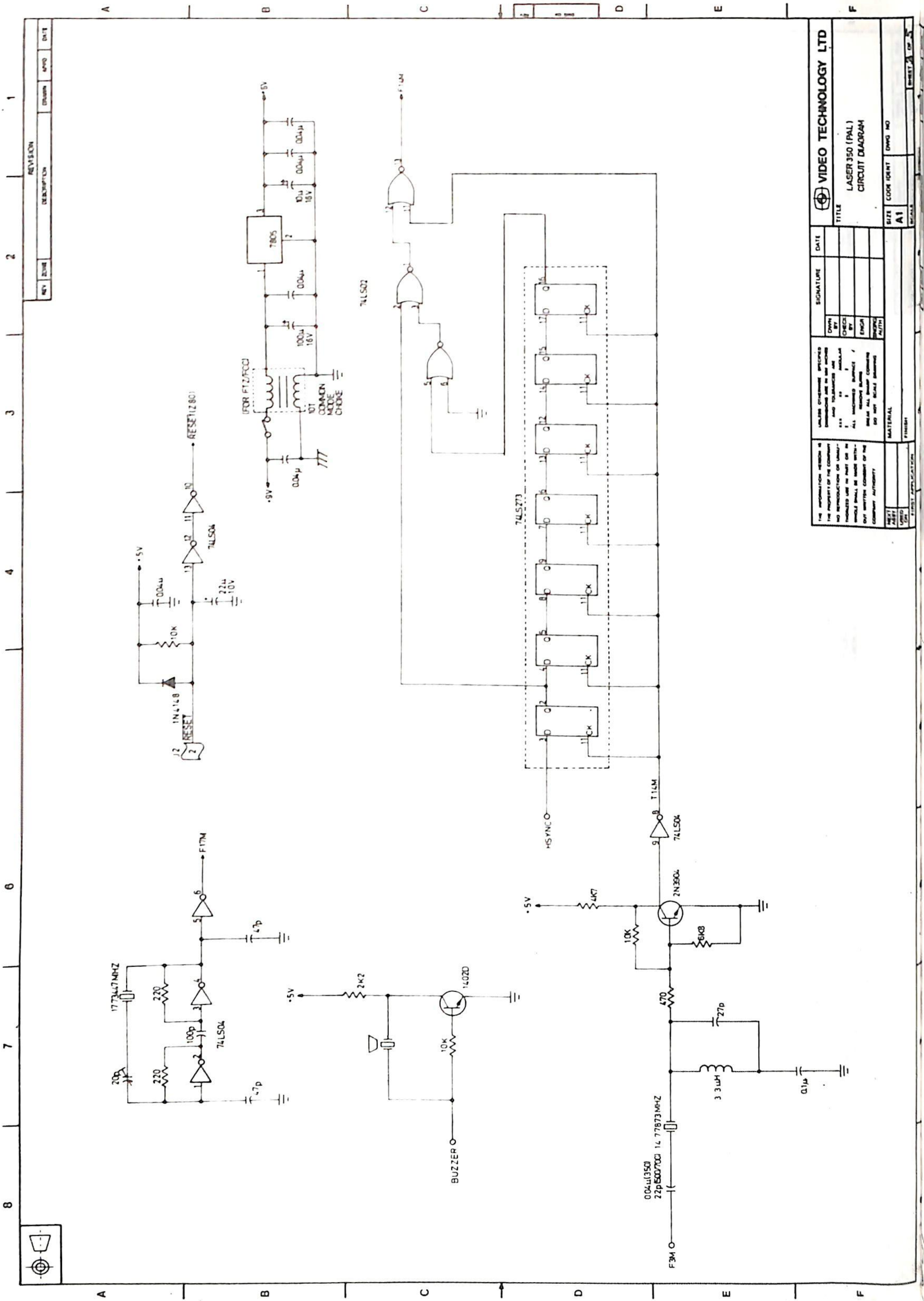
TITLE

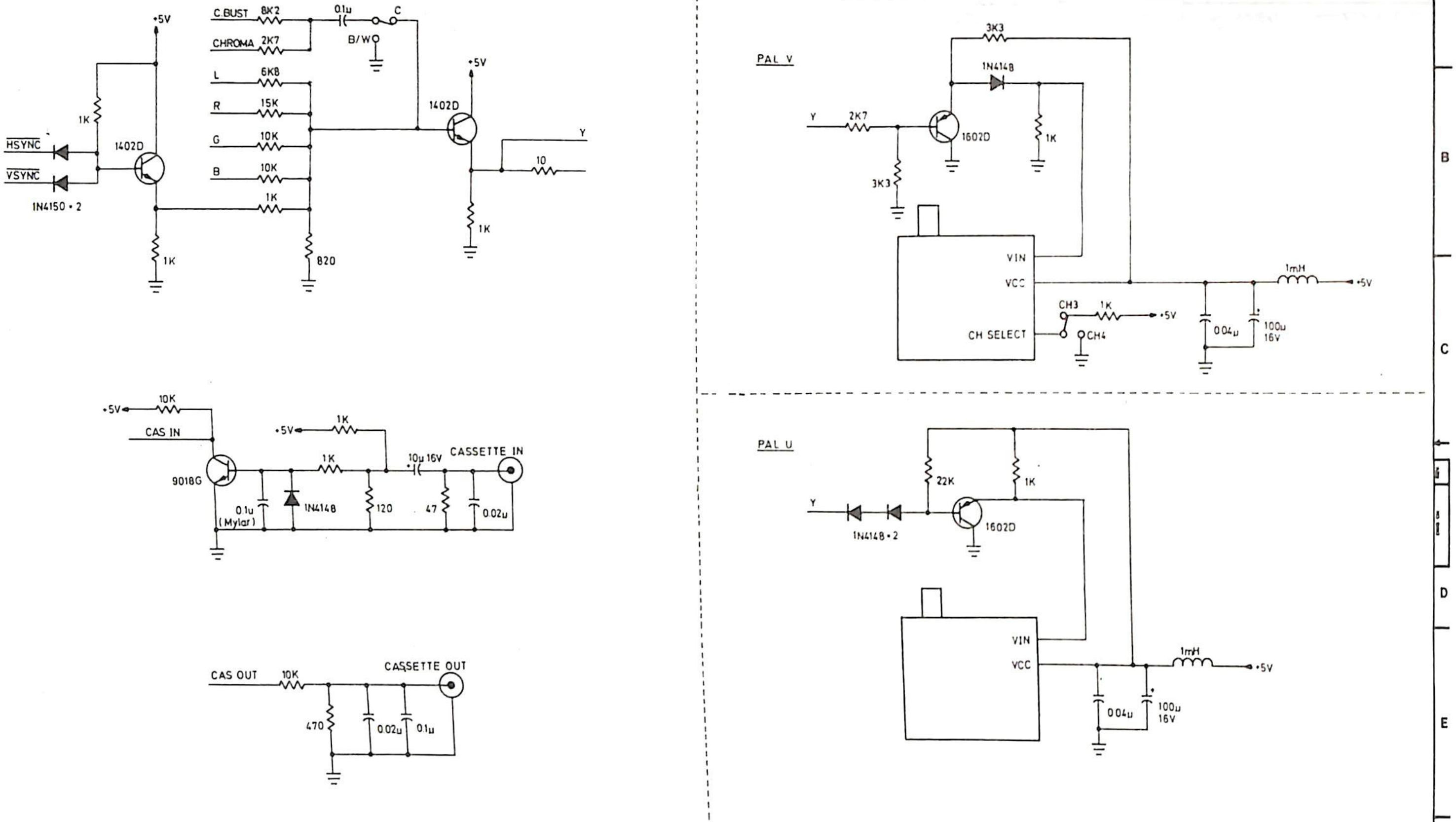
LASER 350 (PAL)
CIRCUIT DIAGRAM

SIZE	CODE IDENT	DWG NO
A1		

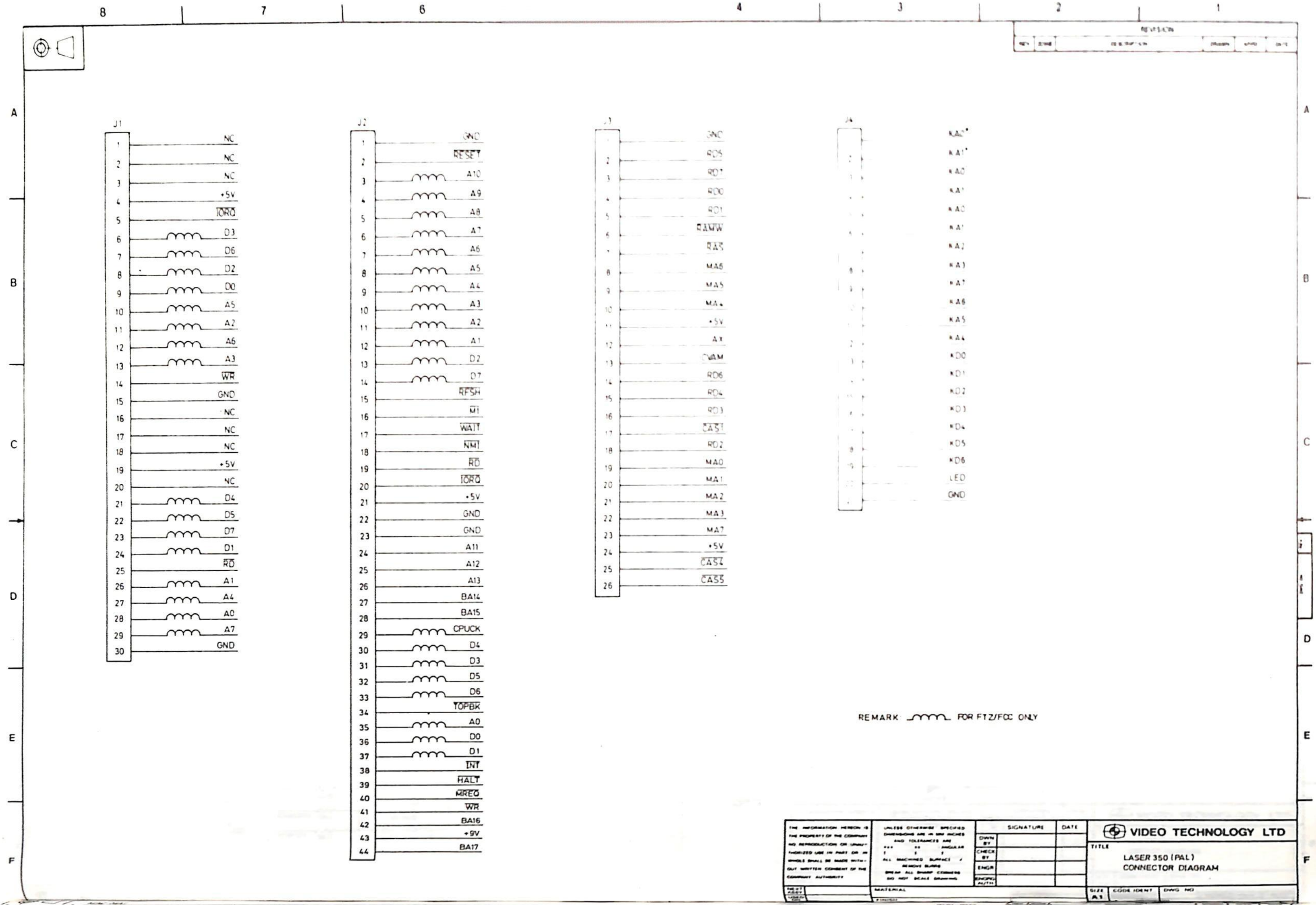
SCALE

SHEET 2 OF 5


VIDEO TECHNOLOGY LTD
TITLE
LASER 350 (PAL)
CIRCUIT DIAGRAM
SIZE A1
CODE IDENT
DWG NO
SHEET 5 OF 5
SIGNATURE
DATE
DRWN BY
CHECK BY
ENGR
MATERIAL
FINISH
REVISION
REV
ZONE
DESCRIPTION
DRAWN
APPD
DATE
+5V
-9V
7805
0.04µ
10µ 16V
100µ 16V
(FOR FT2/FCC)
101 COMMON MODE CHOKE
F14M
74LS02
74LS273
HSYNC
T14M
74LS04
RESET(Z80)
J2
RESET
1N4148
10K
0.04µ
22µ 10V
F17M
17.73447MHZ
220
100p
47p
20p
2K2
1402D
10K
BUZZER
4K7
2N3904
6K8
470
27p
3.3µH
0.1µ
14.77873MHZ
22p(500/700)
00.41(350)
F3M



THE INFORMATION HEREIN IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAU-THORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITH-OUT WRITTEN CONSENT OF THE COMPANY AUTHORITY	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE XXX XX ANGULAR ± ± ± ALL MACHINED SURFACE ✓ REMOVE BURRS BREAK ALL SHARP CORNERS DO NOT SCALE DRAWING	SIGNATURE	DATE	VIDEO TECHNOLOGY LTD
		DWN BY		TITLE LASER 350 (PAL) CIRCUIT DIAGRAM
		CHECK BY		
		ENGR		
		ENGRG AUTH		
NEXT ASSY	MATERIAL			SIZE A1 | CODE IDENT | DWG NO
USED ON	FINISH			SCALE | SHEET 4 OF 5
FIRST APPLICATION				



REMARK: ⌒⌒⌒ FOR FTZ/FCC ONLY

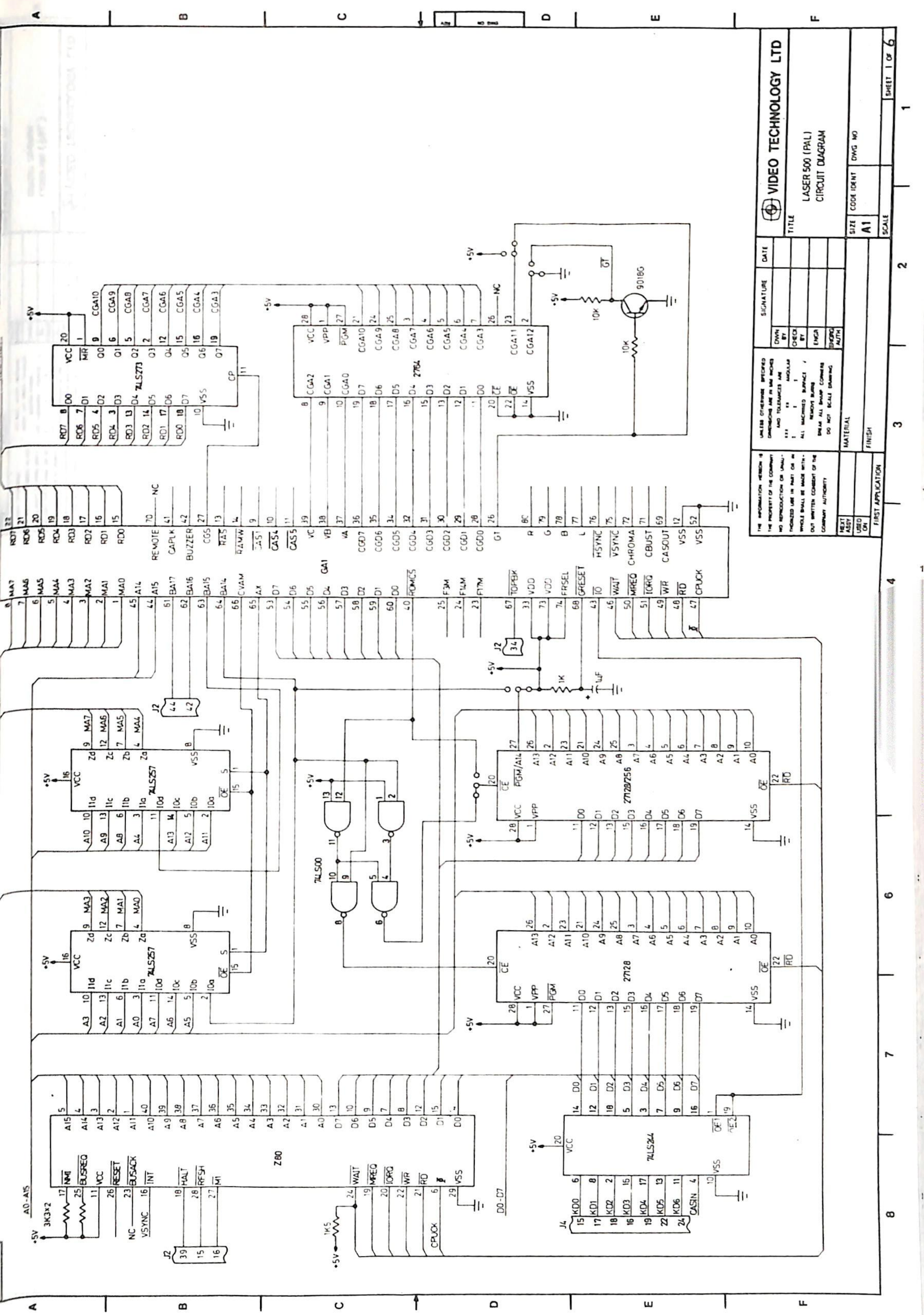
VIDEO TECHNOLOGY LTD

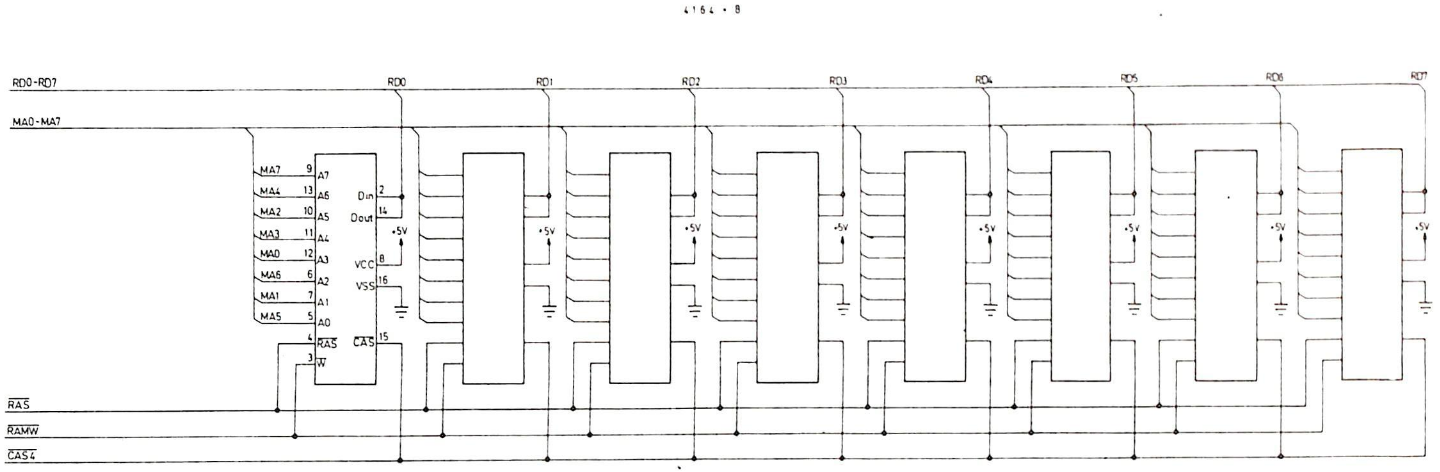
TITLE LASER 350 (PAL) CONNECTOR DIAGRAM

SIGNATURE	DATE
DWN BY	
CHECK BY	
ENGR	

SIZE A1 | CODE IDENT | DWG NO

MATERIAL

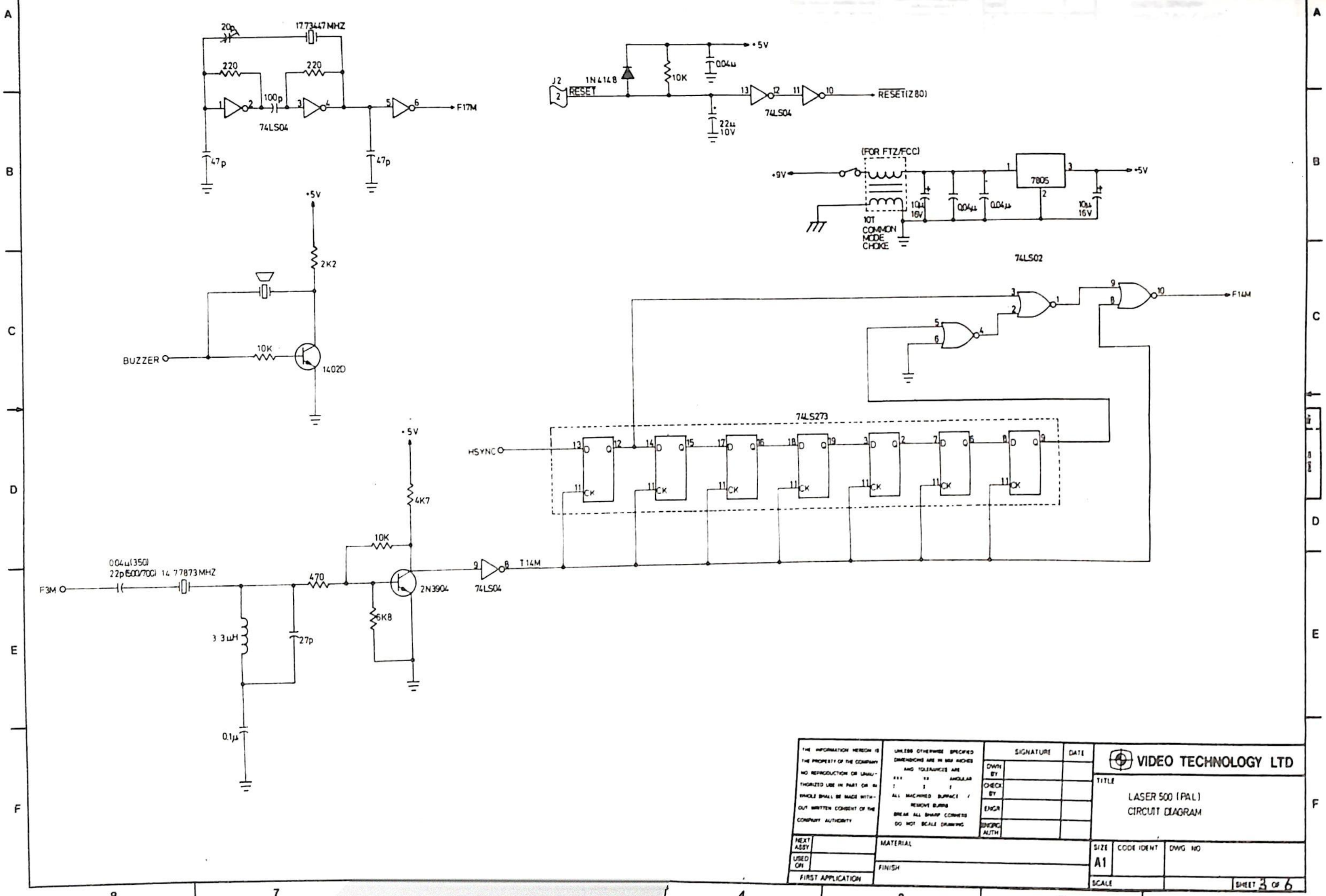

VIDEO TECHNOLOGY LTD
TITLE
LASER 500 (PAL)
CIRCUIT DIAGRAM
SIZE A1
CODE IDENT
DWG NO
SCALE
SHEET 1 OF 6
DATE
SIGNATURE
DWN BY
CHECK BY
ENGR
ENG'G AUTH
MATERIAL
FINISH
NEXT ASSY
USED ON
FIRST APPLICATION
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
ALL MACHINED SURFACE REMOVE BURRS BREAK ALL SHARP CORNERS DO NOT SCALE DRAWING
THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT THE CONSENT OF THE COMPANY AUTHORITY
Z80
74LS273
2764
GA1
74LS257
74LS257
74LS00
27128/256
27128
74LS244
9018G
10K
10K
1K
1K5
3K3X2
1uF
+5V
NC
A0-A15
D0-D7
J2
J4
CPUCK
VSYNC
HSYNC
CHROMA
CBUST
CASOUT
CASIN
REMOTE
CAPLK
BUZZER
ROMCS
GRESET
TOPBK
FRSEL
F3M
F14M
F17M
GT
KD0
KD1
KD2
KD3
KD4
KD5
KD6
A
B
C
D
E
F
1
2
3
4
5
6
7
8



REVISION

REV	ZONE	DESCRIPTION	CHKR	APPD	DATE

		SIGNATURE	DATE	VIDEO TECHNOLOGY LTD
THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE .X.X .XX ANGULAR ± ± ± ALL MACHINED SURFACE / REMOVE BURRS BREAK ALL SHARP CORNERS DO NOT SCALE DRAWING	DWN BY CHECK BY ENGR ENGRG AUTH		TITLE LASER 500 (PAL) CIRCUIT DIAGRAM
NEXT ASSY USED ON FIRST APPLICATION	MATERIAL FINISH			SIZE A1 · CODE IDENT · DWG NO SCALE · SHEET 2 OF 6



THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
.XXX .XX ANGULAR
± ± ±
ALL MACHINED SURFACE √
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING

	SIGNATURE	DATE
DWN BY		
CHECK BY		
ENGR		
ENGRG AUTH		

VIDEO TECHNOLOGY LTD

TITLE
LASER 500 (PAL)
CIRCUIT DIAGRAM

NEXT ASSY	
USED ON	
FIRST APPLICATION	

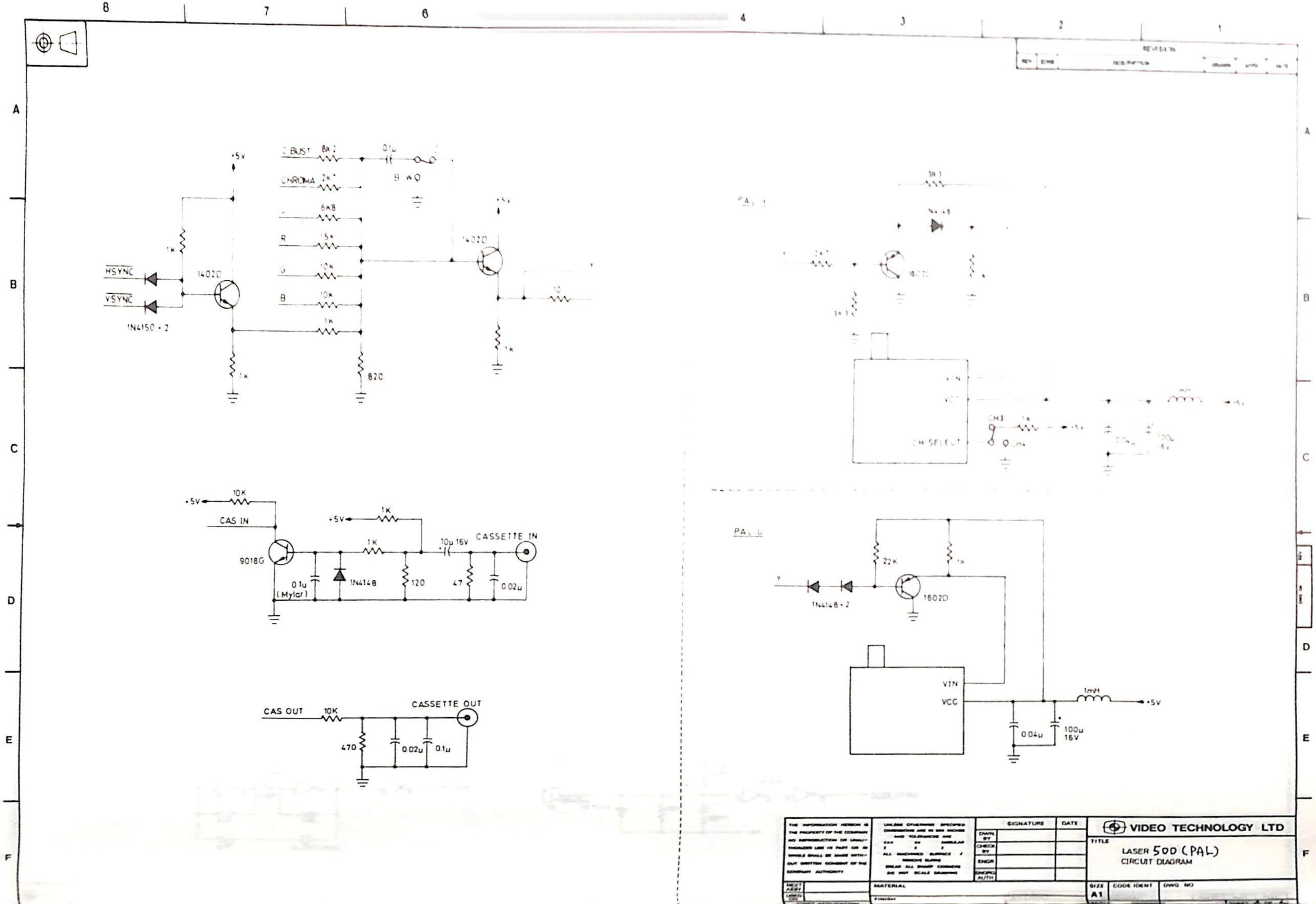
MATERIAL

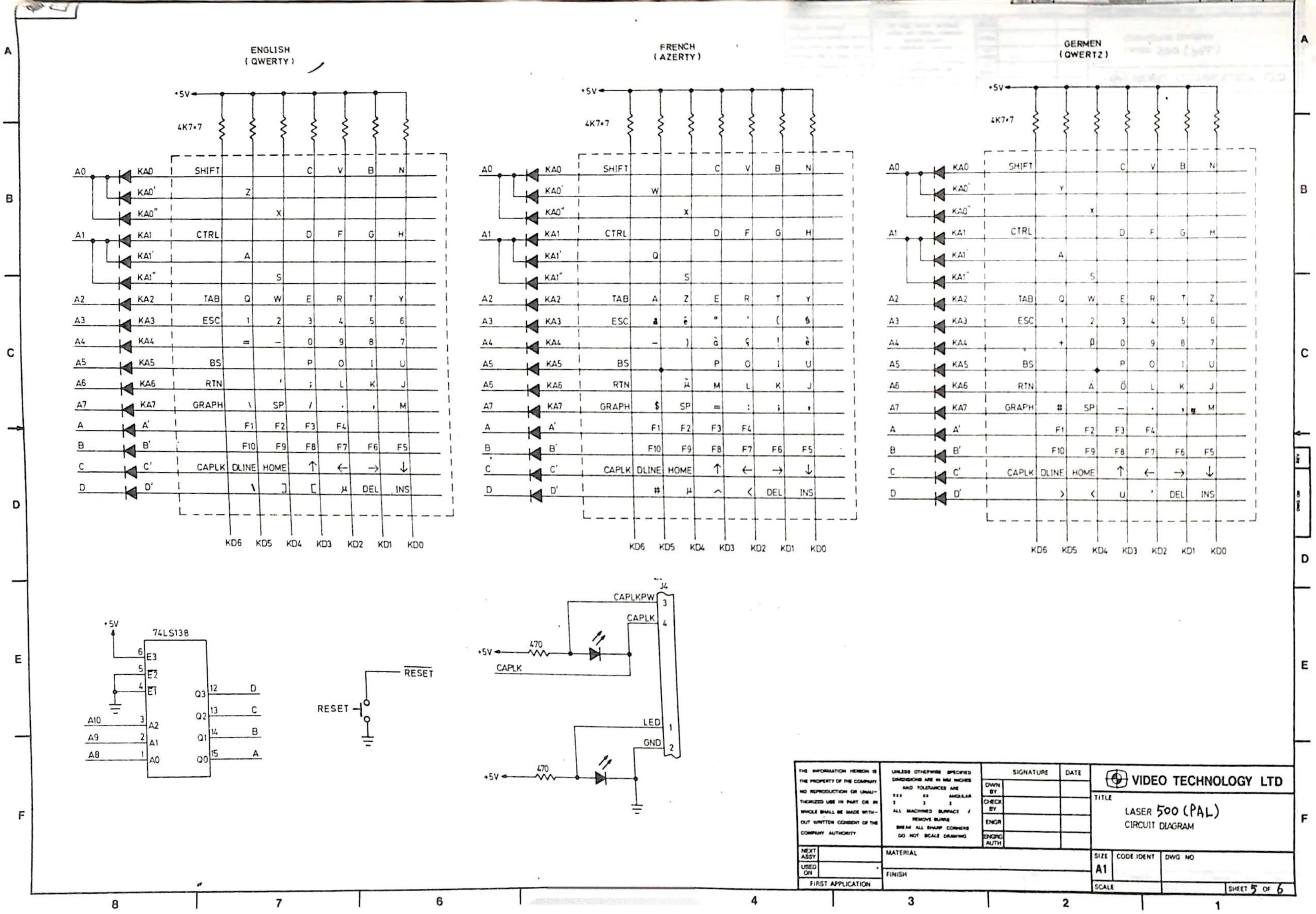
FINISH

SIZE	CODE IDENT	DWG NO
A1		

SCALE

SHEET 3 OF 6


VIDEO TECHNOLOGY LTD
LASER 500 (PAL)
CIRCUIT DIAGRAM
SIZE A1
CODE IDENT
DWG NO
SIGNATURE
DATE
TITLE
MATERIAL
+5V
C BUST
8K2
CHROMA
2K7
6K8
R
15K
G
10K
B
10K
1K
0.1u
B W Q
1402D
1K
HSYNC
VSYNC
1N4150 • 2
1K
820
1402D
10
1K
10K
CAS IN
9018G
0.1u (Mylar)
1N4148
1K
120
10u 16V
47
0.02u
CASSETTE IN
CAS OUT
10K
470
0.02u
0.1u
CASSETTE OUT
PAL I
PAL II
Y
1N4148 • 2
22K
1K
1802D
VIN
VCC
CH SELECT
CH3
CH4
1mH
0.04u
100u 16V



ENGLISH (QWERTY)

	KD6	KD5	KD4	KD3	KD2	KD1	KD0
KA0	SHIFT			C	V	B	N
KA0'		Z					
KA0"			X				
KA1	CTRL			D	F	G	H
KA1'		A					
KA1"			S				
KA2	TAB	Q	W	E	R	T	Y
KA3	ESC	1	2	3	4	5	6
KA4		=	–	0	9	8	7
KA5	BS			P	O	I	U
KA6	RTN		'	;	L	K	J
KA7	GRAPH	\	SP	/	.	,	M
A'		F1	F2	F3	F4		
B'		F10	F9	F8	F7	F6	F5
C'	CAPLK	DLINE	HOME	↑	←	→	↓
D'		\	]	[	μ	DEL	INS

FRENCH (AZERTY)

	KD6	KD5	KD4	KD3	KD2	KD1	KD0
KA0	SHIFT			C	V	B	N
KA0'		W					
KA0"			X				
KA1	CTRL			D	F	G	H
KA1'		Q					
KA1"			S				
KA2	TAB	A	Z	E	R	T	Y
KA3	ESC	&	é	"	'	(	§
KA4		–	)	à	ç	!	è
KA5	BS			P	O	I	U
KA6	RTN		ù	M	L	K	J
KA7	GRAPH	$	SP	=	:	;	,
A'		F1	F2	F3	F4		
B'		F10	F9	F8	F7	F6	F5
C'	CAPLK	DLINE	HOME	↑	←	→	↓
D'		#	μ	^	<	DEL	INS

GERMEN (QWERTZ)

	KD6	KD5	KD4	KD3	KD2	KD1	KD0
KA0	SHIFT			C	V	B	N
KA0'		Y					
KA0"			X				
KA1	CTRL			D	F	G	H
KA1'		A					
KA1"			S				
KA2	TAB	Q	W	E	R	T	Z
KA3	ESC	1	2	3	4	5	6
KA4		+	β	0	9	8	7
KA5	BS			P	O	I	U
KA6	RTN		Ä	Ö	L	K	J
KA7	GRAPH	#	SP	–	.	,	M
A'		F1	F2	F3	F4		
B'		F10	F9	F8	F7	F6	F5
C'	CAPLK	DLINE	HOME	↑	←	→	↓
D'		>	<	U	'	DEL	INS

THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE .xxx .xx ANGULAR ± ± ± ALL MACHINED SURFACE √ REMOVE BURRS BREAK ALL SHARP CORNERS DO NOT SCALE DRAWING

	SIGNATURE	DATE
DWN BY		
CHECK BY		
ENGR		
ENGRG AUTH		

NEXT ASSY | USED ON | FIRST APPLICATION

MATERIAL

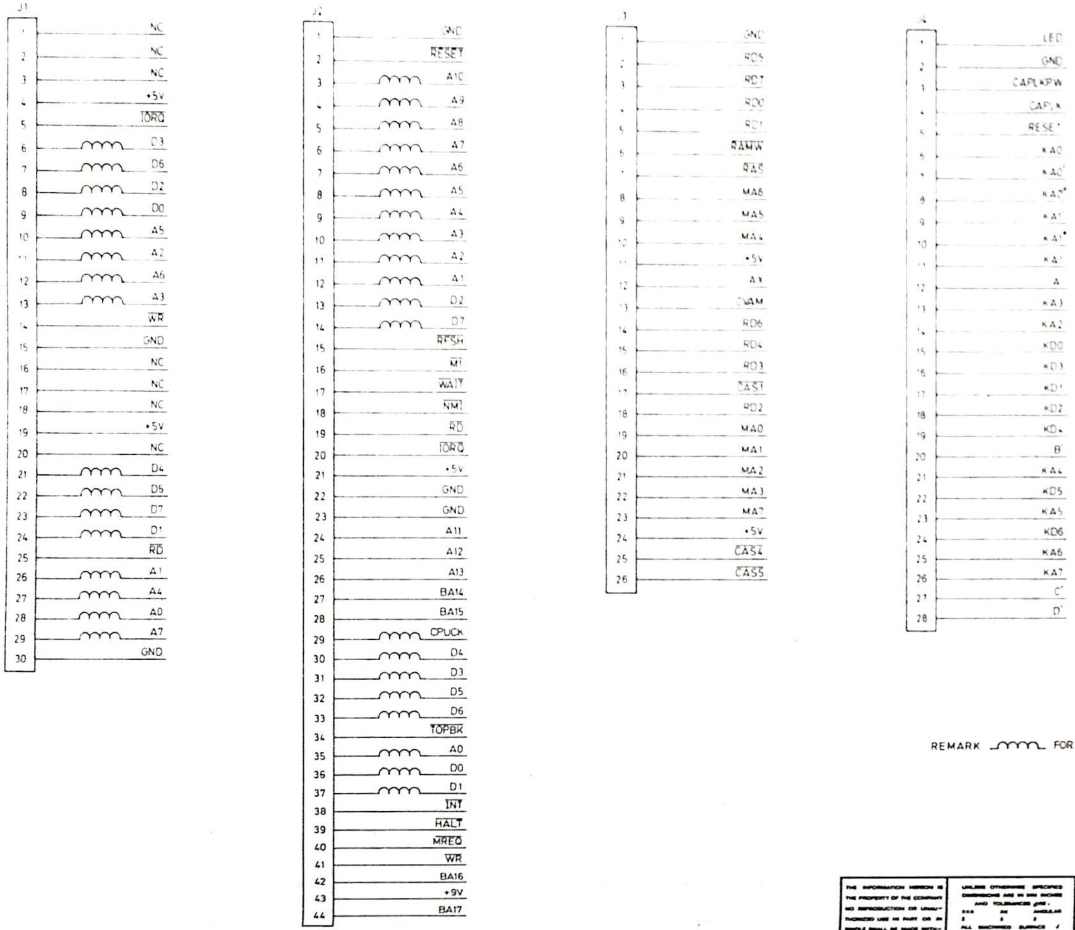
FINISH

VIDEO TECHNOLOGY LTD

TITLE LASER 500 (PAL) CIRCUIT DIAGRAM

SIZE A1 | CODE IDENT | DWG NO

SCALE | SHEET 5 OF 6



REMARK ⌒⌒⌒ FOR FTZ/FCC ONLY

REVISION					
REV	ZONE	DESCRIPTION	DRAWN	APPD	DATE

VIDEO TECHNOLOGY LTD

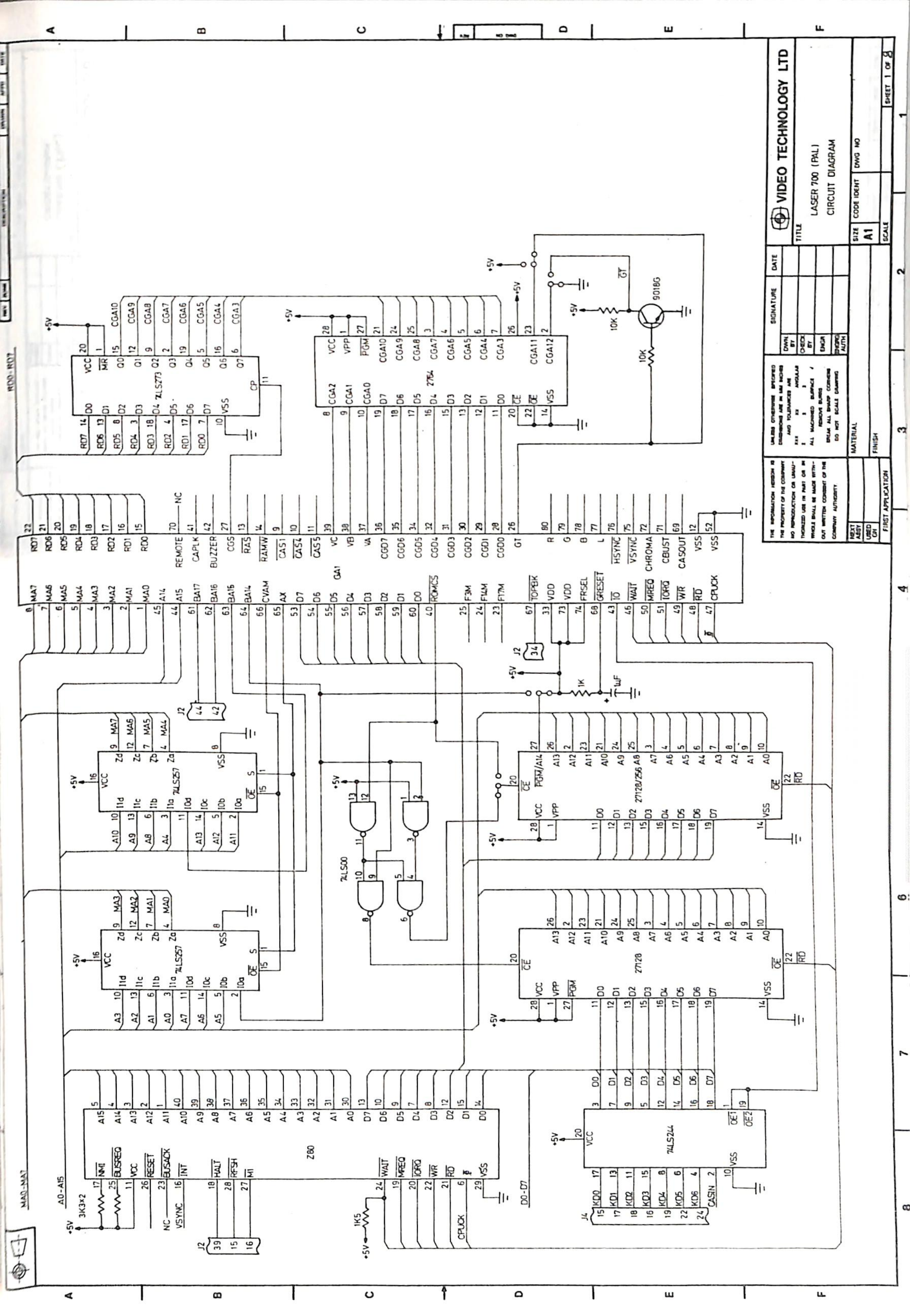
TITLE LASER 500 (PÁL)
CONNECTOR DIAGRAM

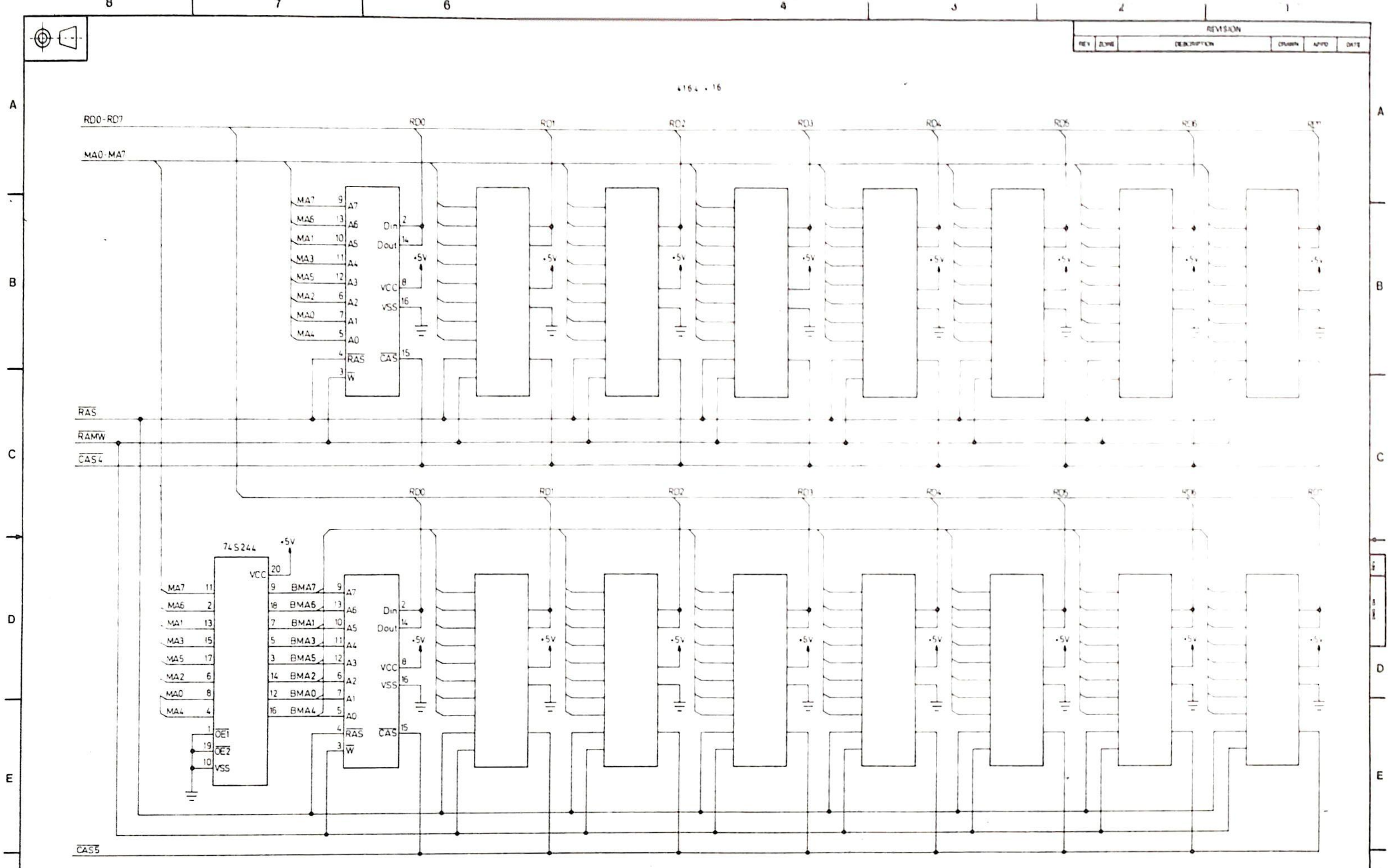
	SIGNATURE	DATE
DWN BY		
CHECK BY		
ENGR		
CHGRG AUTH		

MATERIAL

FINISH

SIZE A1 | CODE IDENT | DWG NO


VIDEO TECHNOLOGY LTD
TITLE
LASER 700 (PAL)
CIRCUIT DIAGRAM
SIZE A1
CODE IDENT
DWG NO
SCALE
SHEET 1 OF 8
THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
ALL MACHINED SURFACE
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING
SIGNATURE
DATE
DWN BY
CHECK BY
ENGR
PROD AUTH
MATERIAL
FINISH
NEXT ASSY
USED ON
FIRST APPLICATION
Z80
74LS244
74LS257
74LS257
74LS00
27128
27128/256
2764
74LS273
GA1
9018G
J2
J4
3K3×2
1K5
10K
10K
1K
1uF
+5V
A0-A15
D0-D7
MA0-MA7
RD0-RD7
NMI
BUSREQ
RESET
BUSACK
INT
HALT
RFSH
M1
WAIT
MREQ
IORQ
WR
RD
CPUCK
VSYNC
HSYNC
CHROMA
CBUST
CASOUT
REMOTE
CAPLK
BUZZER
CGS
RAS
RAMW
CAS1
CAS4
CAS5
ROMCS
TOPBK
FRSEL
GRESET
F3M
F14M
F17M
GT
NC
KD0
KD1
KD2
KD3
KD4
KD5
KD6
CASIN
CGA0
CGA1
CGA2
CGA3
CGA4
CGA5
CGA6
CGA7
CGA8
CGA9
CGA10
CGA11
CGA12
PGM/A14
A B C D E F
1 2 3 4 5 6 7 8



THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
.XXX .XX ANGULAR
± ± ±
ALL MACHINED SURFACE √
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING

	SIGNATURE	DATE
DWN BY		
CHECK BY		
ENGR		
ENGRG AUTH		

VIDEO TECHNOLOGY LTD

TITLE

LASER 700 (PAL)
CIRCUIT DIAGRAM

NEXT ASSY

USED ON

FIRST APPLICATION

MATERIAL

FINISH

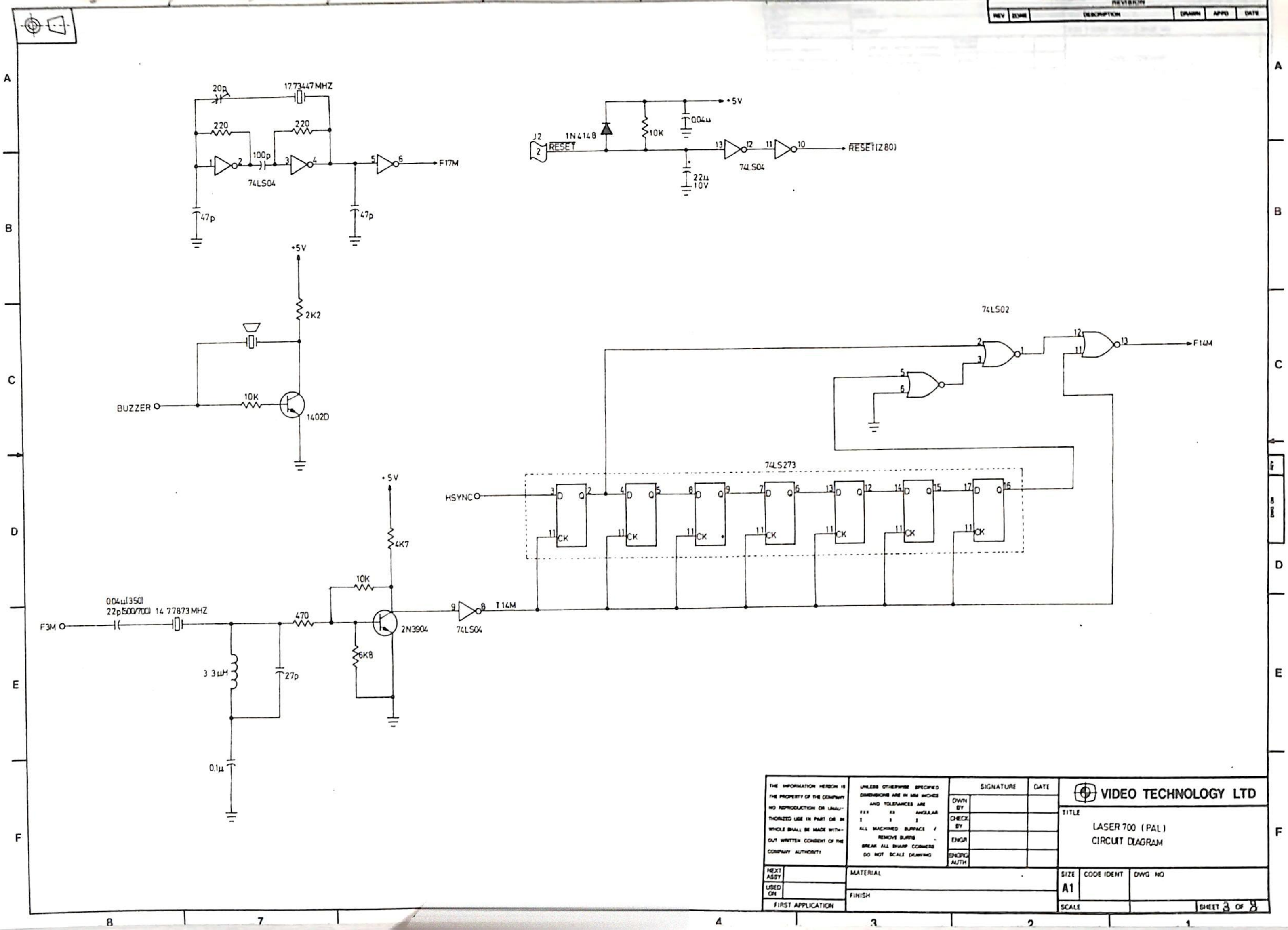
SIZE A1

CODE IDENT

DWG NO

SCALE

SHEET 2 OF 8



THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
.XXX .XX ANGULAR
± ± ±
ALL MACHINED SURFACE √
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING

	SIGNATURE	DATE
DWN BY		
CHECK BY		
ENGR		
ENG'G AUTH		

NEXT ASSY

USED ON

FIRST APPLICATION

MATERIAL

FINISH

VIDEO TECHNOLOGY LTD

TITLE
LASER 700 (PAL)
CIRCUIT DIAGRAM

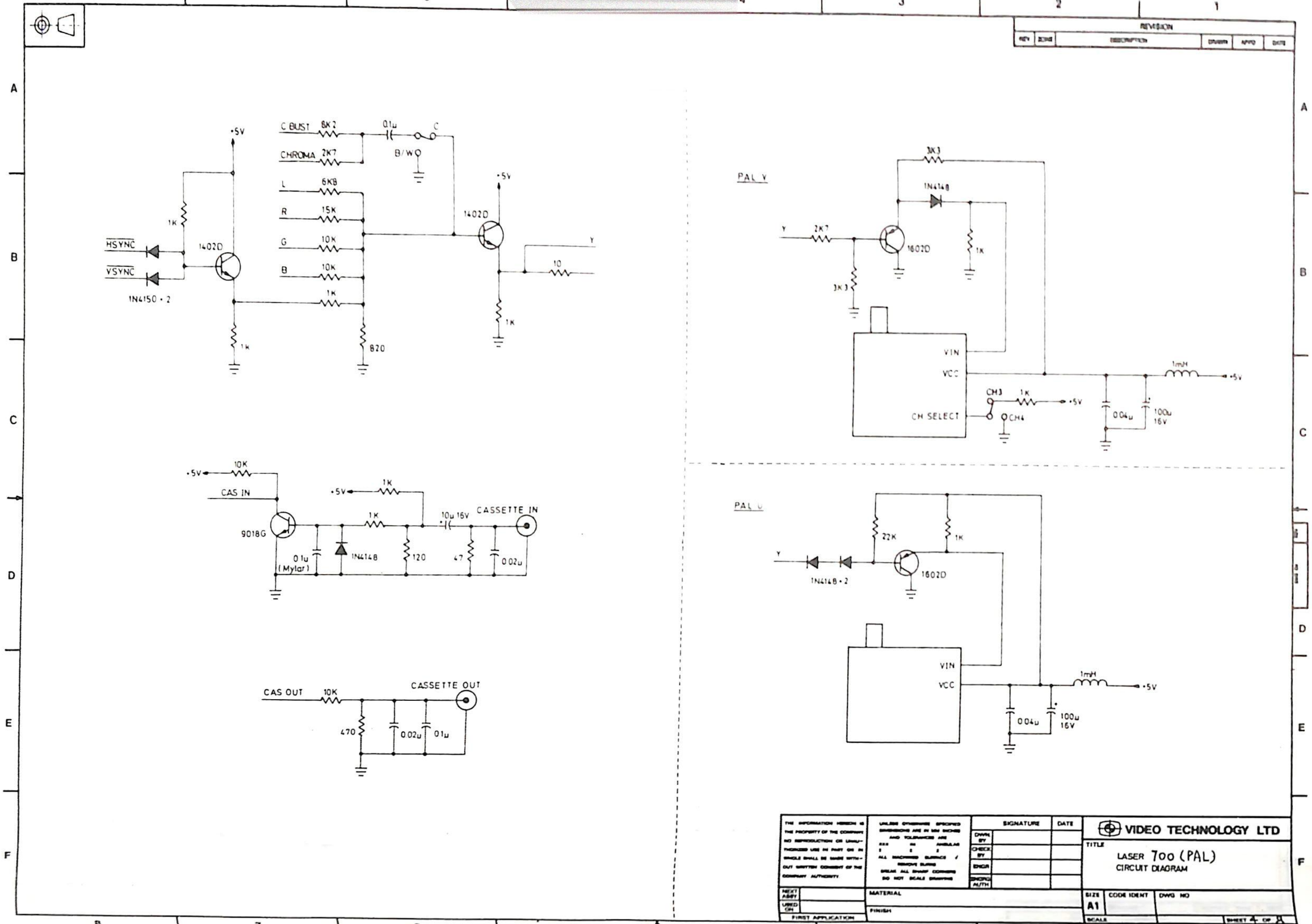
SIZE A1

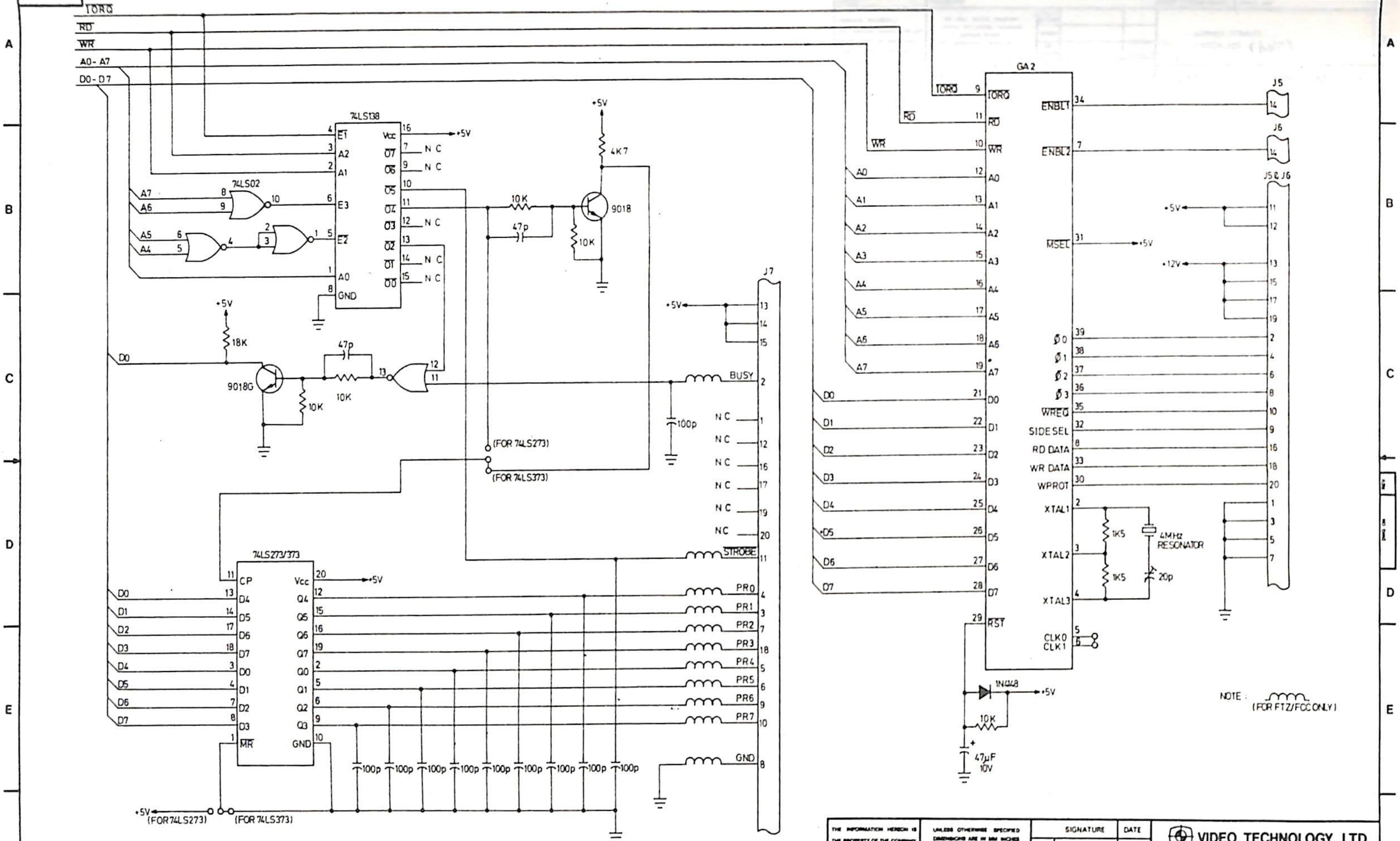
CODE IDENT

DWG NO

SCALE

SHEET 3 OF 8


REVISION
REV ZONE DESCRIPTION DRAWN APPD DATE
•5V
C BUST 8K2
CHROMA 2K7
0.1u
C
B/W
L 6K8
R 15K
G 10K
B 10K
1K
HSYNC
VSYNC
1N4150 • 2
1402D
1K
820
•5V
1402D
Y
10
1K
PAL Y
3K3
1N4148
Y 2K7
1602D
1K
3K3
VIN
VCC
CH SELECT
CH3 1K •5V
CH4
1mH •5V
0.04u
100u 16V
PAL U
22K
1K
Y
1N4148 • 2
1602D
VIN
VCC
1mH •5V
0.04u
100u 16V
•5V 10K
CAS IN
9018G
•5V 1K
1K
10u 16V
CASSETTE IN
0.1u (Mylar)
1N4148
120
47
0.02u
CAS OUT 10K
CASSETTE OUT
470
0.02u
0.1u
THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
ALL MACHINED SURFACE / REMOVE BURRS BREAK ALL SHARP CORNERS DO NOT SCALE DRAWING
SIGNATURE DATE
DWN BY
CHECK BY
ENGR
ENGRG AUTH
NEXT ASSY
USED ON
FIRST APPLICATION
MATERIAL
FINISH
VIDEO TECHNOLOGY LTD
TITLE
LASER 700 (PAL)
CIRCUIT DIAGRAM
SIZE A1
CODE IDENT
DWG NO
SCALE
SHEET 4 OF 8



THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
.XXX .XX ANGULAR
± ± ±
ALL MACHINED SURFACE √
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING

	SIGNATURE	DATE
DWN BY		
CHECK BY		
ENGR		
ENGRG AUTH		

NEXT ASSY

USED ON

FIRST APPLICATION

MATERIAL

FINISH

VIDEO TECHNOLOGY LTD

TITLE
LASER 700 (PAL)
CIRCUIT DIAGRAM

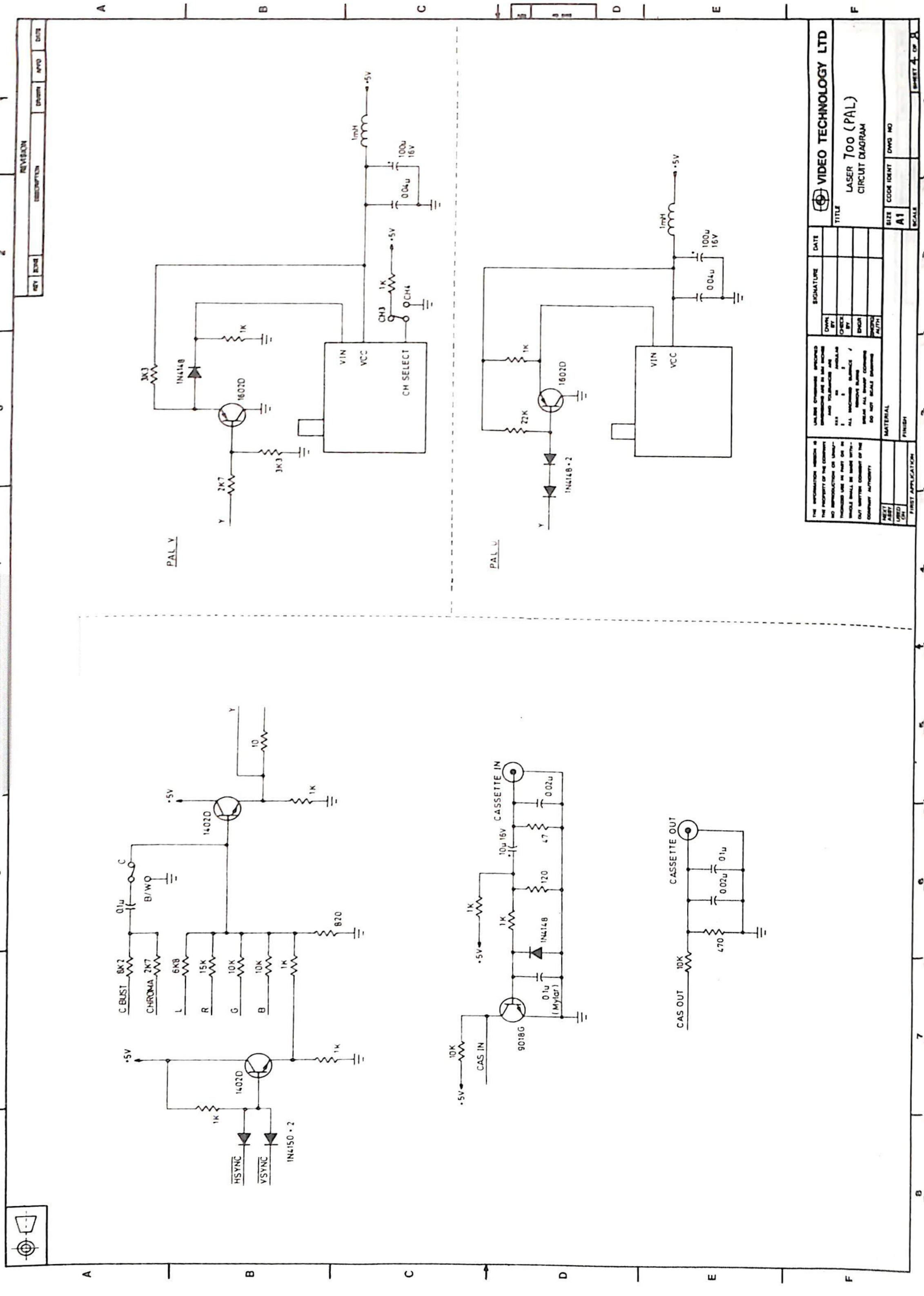
SIZE A1

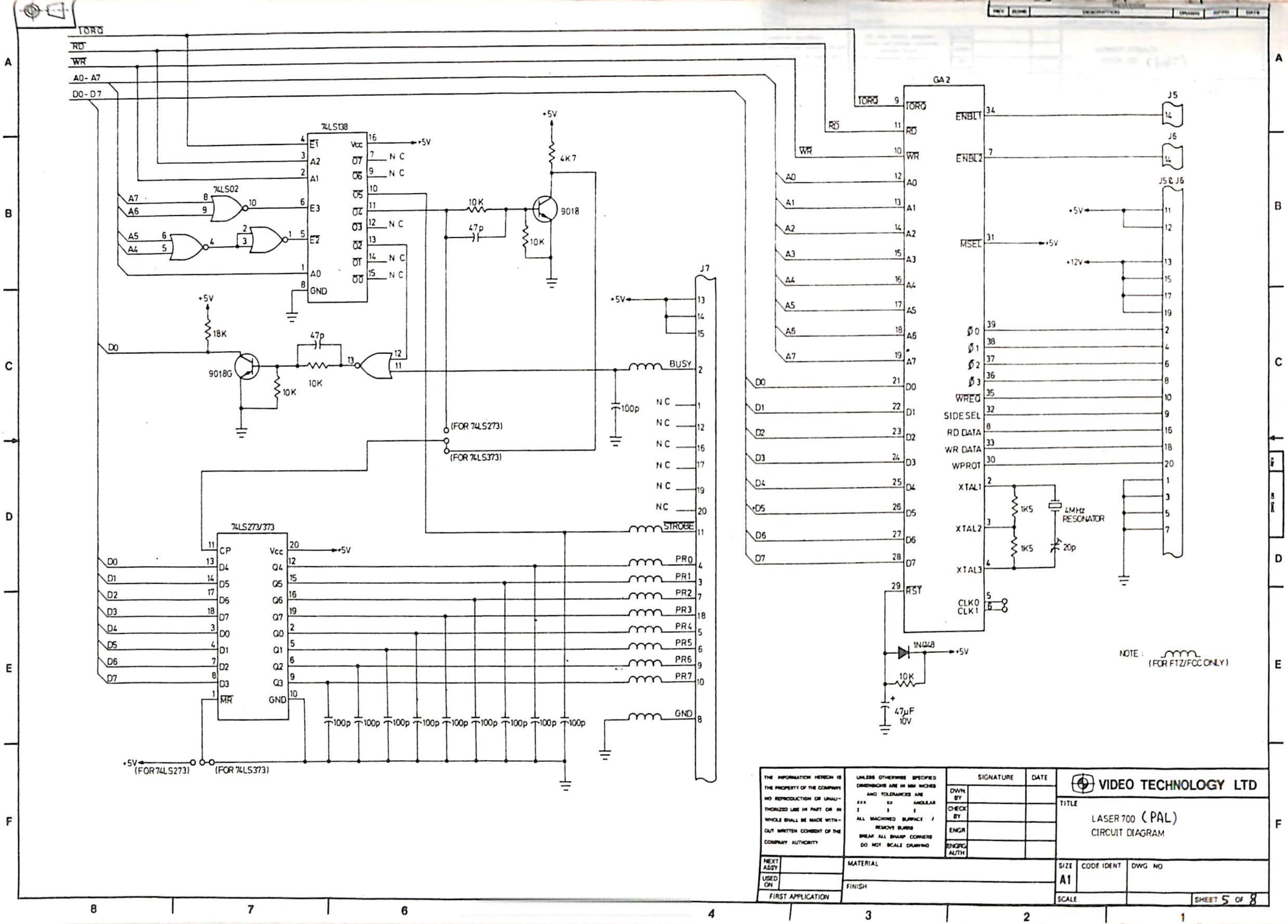
CODE IDENT

DWG NO

SCALE

SHEET 5 OF 8


VIDEO TECHNOLOGY LTD
LASER 700 (PAL)
CIRCUIT DIAGRAM
SIZE A1
SHEET 4 OF 8
PAL Y
PAL U
VIN
VCC
CH SELECT
CH3
CH4
1mH
100u 16V
0.04u
+5V
3K3
1N4148
1602D
1K
2K7
Y
22K
1N4148 · 2
C BUST
CHROMA
8K2
2K7
0.1u
C
B/W
L
R
G
B
6K8
15K
10K
10K
1K
820
1402D
10
HSYNC
VSYNC
1N4150 · 2
CASSETTE IN
0.02u
47
10u 16V
120
1N4148
0.1u (Mylar)
9018G
CAS IN
10K
CASSETTE OUT
0.1u
0.02u
470
CAS OUT



THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
.XXX .XX ANGULAR
± ± ±
ALL MACHINED SURFACE / REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING

	SIGNATURE	DATE
DWN BY		
CHECK BY		
ENGR		
ENGRG AUTH		

VIDEO TECHNOLOGY LTD

TITLE
LASER 700 (PAL)
CIRCUIT DIAGRAM

NEXT ASSY
USED ON
FIRST APPLICATION

MATERIAL

FINISH

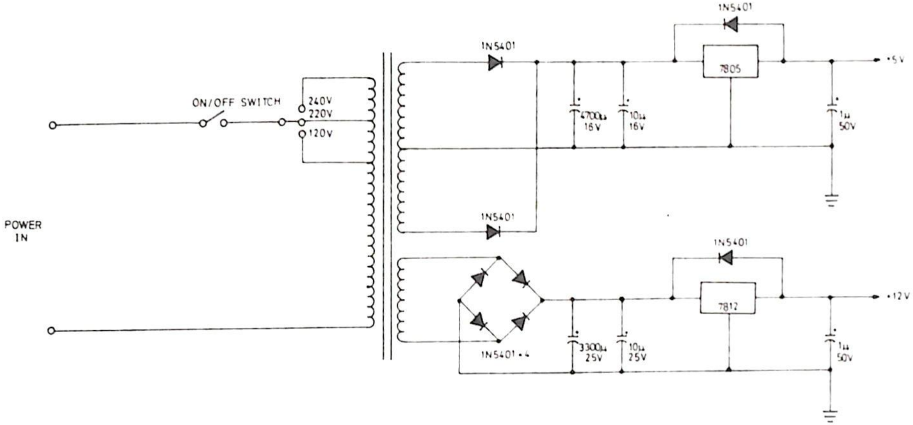
SIZE	CODE IDENT	DWG NO
A1		

SCALE

SHEET 5 OF 8

REVISION

REV	ZONE	DESCRIPTION	DRAWN	APPD	DATE



THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
.XX .X ANGULAR
± ± ±
ALL MACHINED SURFACE /
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING

	SIGNATURE	DATE
DWN BY		
CHECK BY		
ENGR		
ENGRG AUTH		

VIDEO TECHNOLOGY LTD

TITLE
LASER 700 (PAL)
POWER SUPPLY

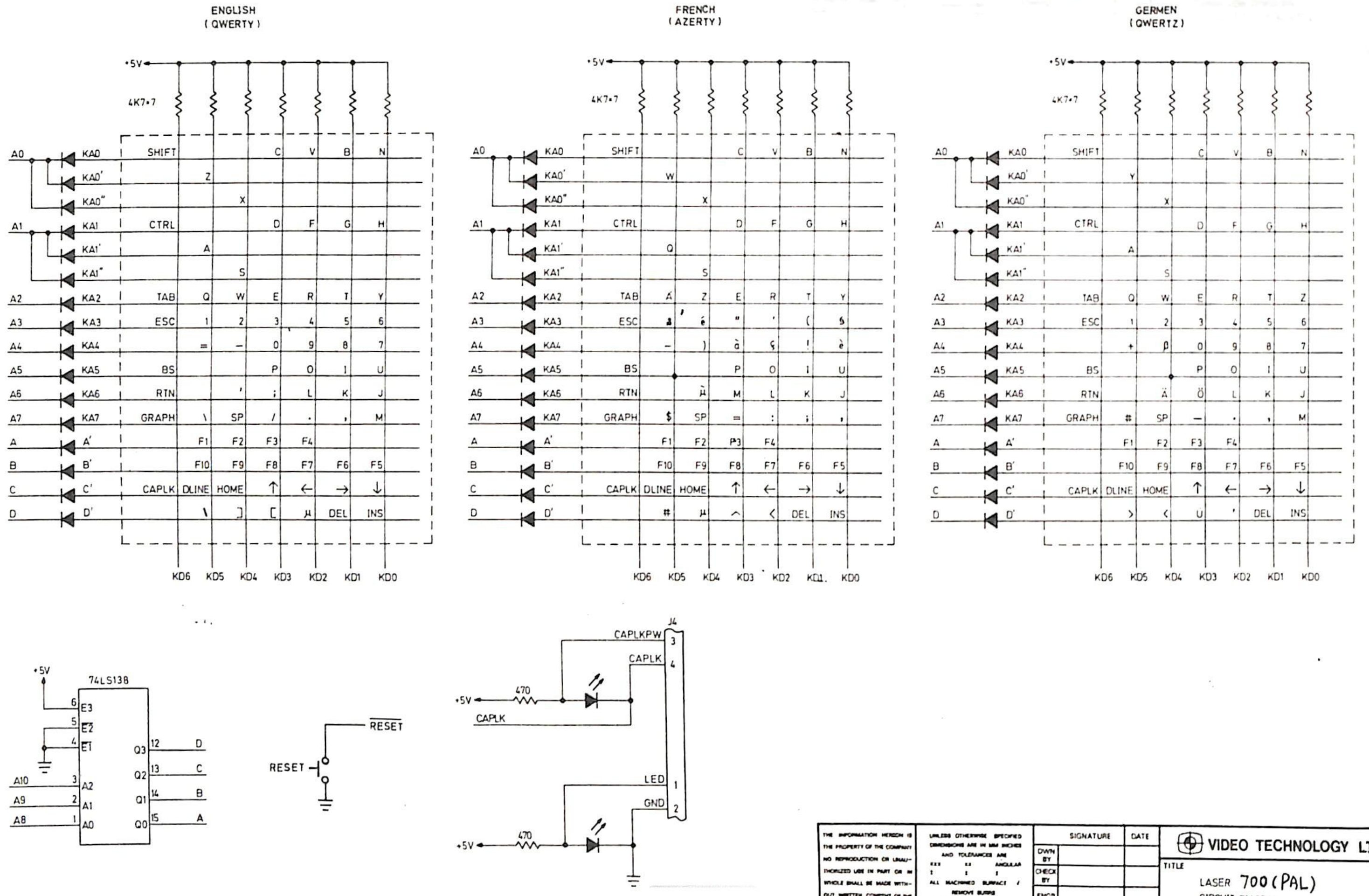
NEXT ASSY	
USED ON	
FIRST APPLICATION	

MATERIAL

FINISH

SIZE	CODE IDENT	DWG NO
A1		

SCALE
SHEET 6 OF 8



ENGLISH (QWERTY)

		KD6	KD5	KD4	KD3	KD2	KD1	KD0
A0	KA0	SHIFT			C	V	B	N
	KA0'		Z					
	KA0"			X				
A1	KA1	CTRL			D	F	G	H
	KA1'		A					
	KA1"			S				
A2	KA2	TAB	Q	W	E	R	T	Y
A3	KA3	ESC	1	2	3	4	5	6
A4	KA4		=	–	0	9	8	7
A5	KA5	BS			P	O	I	U
A6	KA6	RTN		'	;	L	K	J
A7	KA7	GRAPH	\	SP	/	.	,	M
A	A'		F1	F2	F3	F4		
B	B'		F10	F9	F8	F7	F6	F5
C	C'	CAPLK	DLINE	HOME	↑	←	→	↓
D	D'		\	]	[	µ	DEL	INS

FRENCH (AZERTY)

		KD6	KD5	KD4	KD3	KD2	KD1	KD0
A0	KA0	SHIFT			C	V	B	N
	KA0'		W					
	KA0"			X				
A1	KA1	CTRL			D	F	G	H
	KA1'		Q					
	KA1"			S				
A2	KA2	TAB	Á	Z	E	R	T	Y
A3	KA3	ESC	&	é	"	'	(	§
A4	KA4		–	)	à	ç	!	è
A5	KA5	BS			P	O	I	U
A6	KA6	RTN		ù	M	L	K	J
A7	KA7	GRAPH	$	SP	=	:	;	,
A	A'		F1	F2	P3	F4		
B	B'		F10	F9	F8	F7	F6	F5
C	C'	CAPLK	DLINE	HOME	↑	←	→	↓
D	D'		#	µ	^	<	DEL	INS

GERMEN (QWERTZ)

		KD6	KD5	KD4	KD3	KD2	KD1	KD0
A0	KA0	SHIFT			C	V	B	N
	KA0'		Y					
	KA0"			X				
A1	KA1	CTRL			D	F	G	H
	KA1'		A					
	KA1"			S				
A2	KA2	TAB	Q	W	E	R	T	Z
A3	KA3	ESC	1	2	3	4	5	6
A4	KA4		+	β	0	9	8	7
A5	KA5	BS			P	O	I	U
A6	KA6	RTN		Ä	Ö	L	K	J
A7	KA7	GRAPH	#	SP	–	.	,	M
A	A'		F1	F2	F3	F4		
B	B'		F10	F9	F8	F7	F6	F5
C	C'	CAPLK	DLINE	HOME	↑	←	→	↓
D	D'		>	<	Ü	'	DEL	INS

THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE .XXX .XX ANGULAR ± ± ± ALL MACHINED SURFACE / REMOVE BURRS BREAK ALL SHARP CORNERS DO NOT SCALE DRAWING

	SIGNATURE	DATE
DWN BY		
CHECK BY		
ENGR		
ENGRG AUTH		

NEXT ASSY

USED ON

FIRST APPLICATION

MATERIAL

FINISH

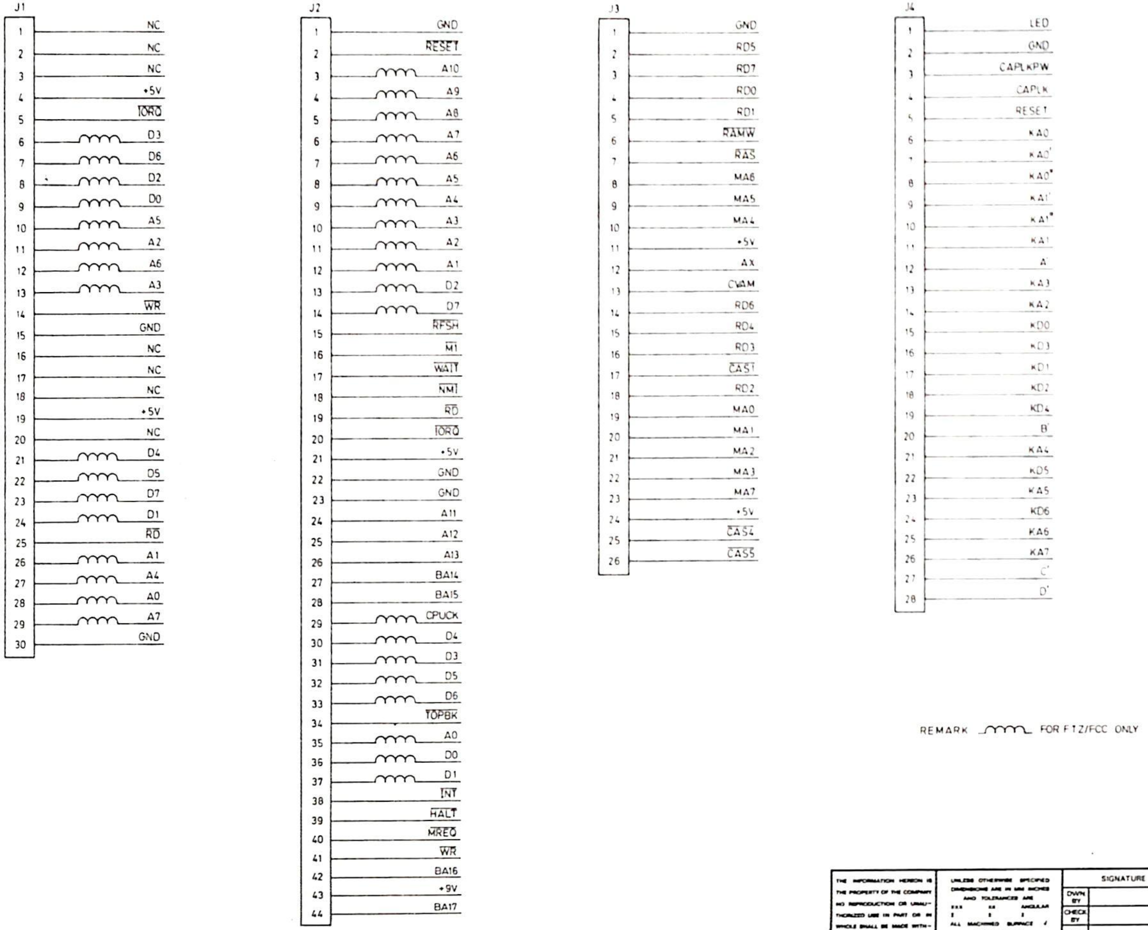
VIDEO TECHNOLOGY LTD

TITLE LASER 700 (PAL)
CIRCUIT DIAGRAM

SIZE A1 | CODE IDENT | DWG NO

SCALE | SHEET 7 OF 8

REVISION					
REV	ZONE	DESCRIPTION	DRAWN	APPD	DATE



THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
.XXX .XX ANGULAR
± ± ±
ALL MACHINED SURFACE √
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING

	SIGNATURE	DATE
DWN BY		
CHECK BY		
ENGR		
ENGRG AUTH		

VIDEO TECHNOLOGY LTD

TITLE
LASER 700 (PAL)
CONNECTOR DIAGRAM

NEXT ASSY	
USED ON	
FIRST APPLICATION	

MATERIAL

FINISH

SIZE	CODE IDENT	DWG NO
A1		
SCALE		SHEET 8 OF 8

APPENDIX M

VARIATION BETWEEN DIFFERENT MODELS

1. PERITEL MODEL WITH AZERTY KEYBOARD

1.1 The Cap-lock key is effective for the keys 'A'–'Z', '0'–'9' and [°;] . When the LED is ON, these keys will generate codes same as that when the shift key is pressed.

1.2 Pressing CTRL together with the following keys with geer-ate codes as shown belows:

CTRL-7	#
CTRL-8	··
CTRL-9	∧
CTRL-0	<
CTRL-)	>
CTRL-—	\

1.3 Some ASCII characters are substituted by French characters. The changes are as follows:

Code	**Original ASCII**	**French character**
40H	@	a
5BH	[	°
5CH	\	ç
5DH	]	§
7BH	{	ė
7CH	|	u̇
7DH	}	ė
7EH	~	··

1.4 There is no RF output for TV at the rear side. A 8 PIN DIN socket is used instead. A PERITEL cable is provided for connection to the PERITEL input of color TV.

1.5 There is no color defeat and channel select switch.

2. PAL MODEL WITH QWERTZ KEYBOARD (GERMAN)

2.1 Pressing CTRL together will the following keys will generate codes as follows:

CTRL-7	'
CTRL-8	<
CTRL-9	>
CTRL-0	Ü
CTRL-B	ü
CTRL-+	\

2.2 Some ASCII character are substituted by GERMAN characters. The changes are as follows:

Code	Original ASCII	German character
40H	@	§
5BH	[	A
5CH	\	O
5DH	]	U
7BH	{	a
7CH	|	o
7DH	}	u
7EH	~	

3. PAL MODEL WITH AZERTY KEYBOARD

1.1, 1.2 and 1.3 apply to this model also.

APPENDIX N

BASIC APPLICATION PROGRAMMS

1. **SUM & AVERAGE**

   This program computes the total sum and average of a group of numbers. Can you tell the logic behind the computer?

```
10 REM SUM & AVERAGE
20 CLS
30 PRINT "SUM AND AVERAGE"
40 INPUT "ENTER HOW MANY NOS.";A
50 FOR I=1 TO A
60 PRINT "NOS." ;I;"=";
70 INPUT B
80 C=C+B:NEXT
90 PRINT "SUM =";C
100 PRINT "AVERAGE =";C/A
110 END
```

```
RUN
```

```
SUM AND AVERAGE
ENTER HOW MANY NOS.? 5
NOS. 1=? 10
NOS. 2=? 20
NOS. 3=? 30
NOS. 4=? 40
NOS. 5=? 50
SUM = 150
AVERAGE = 30
Ready
```

2. **PERMUTATION & COMBINATION**

   Permutation and combination are 2 popular subjects in modern mathematics. By using this program, you can get the answers quickly. Can you beat the computer in speed and accuracy?

```
10 REM PERMUTATION & COMBINATION
20 CLS
30 PRINT "PERMUTATION & COMBINATON"
40 INPUT "ENTER TOTAL NOS.";A
50 INPUT "ENTER SUBSET NOS.";B
60 C=1:D=1
70 IF B>A THEN 30
80 FOR I=A-B+1 TO A
90 IF C*I>1E+36 THEN 200
100 C=C*I:NEXT
110 FOR I=2 TO B
120 D=D*I:NEXT
130 PRINT "PERMUTATION =";C
140 PRINT "COMBINATION =";C/D
150 END
200 PRINT "OVERFLOW":GOTO 60
```

```
RUN

PERMUTATION & COMBINATION
ENTER TOTAL NOS.? 5
SUBSET NOS.? 4
PERMUTATION = 120
COMBINATION = 5
Ready
```

3. HIGHEST COMMON FACTOR (H.C.F.)

Just input 2 numbers and this program will tell you the Highest Common Factor.

```
10 REM FIND HCF
20 CLS
30 PRINT "FIND H.C.F."
40 INPUT "ENTER 2 NUMBERS";A,B
50 IF A=0 OR B=0 THEN 100
60 IF A>B THEN A=A-B
70 IF A<B THEN B=B-A
80 IF A<>B THEN 60
90 PRINT "H.C.F. =";A
100 END
```

```
RUN


FIND H.C.F.
ENTER 2 NUMBERS? 20,10
H.C.F. = 10
Ready
```

4. **LOWEST COMMON MULTIPLE (L.C.M.)**

   Similar to (H.C.F.) but will give you the Lowest Common Multiple instead of the Highest Common Factor.

```
10 REM FIND LCM
20 CLS
30 PRINT "FIND L.C.M."
40 INPUT "ENTER 2 NUMBERS";A,B
50 IF A=0 OR B=0 THEN 110
60 IF A>B THEN C=A-1 ELSE C=B-1
70 C=C+1
80 IF INT(C/A)<>C/A THEN 70
90 IF INT(C/B)<>C/B THEN 70
 100 PRINT "L.C.M. =";C
 110 END
```

```
RUN


FIND L.C.M.
ENTER 2 NUMBER? 11,13
L.C.M. = 143
Ready
```

5. **PRIME FACTOR**

   This program identifies all the prime factors hidden in any number.

```
10 REM PRIME FACTORS
20 CLS
30 PRINT "PRIME FACTORS"
40 INPUT "ENTER A NUMBER";A
50 IF A=0 THEN 130
60 PRINT SGN(A);:A=ABS(A)
70 FOR I= 2 TO A:B=0
80 IF A/I<>INT(A/I)   THEN 100
90 A=A/I:B=B+1:GOTO 80
100 IF B=0 THEN 120
110 PRINT I;"^";B;
120 NEXT
130 END


RUN


PRIME FACTORS
ENTER A NUMBER? 240
1  2 ^ 4  3 ^ 1  5 ^ 1
Ready
```

6. **ROOTS OF QUADRATIC EQUATION**

Generally speaking, Quadratic Equations are in the form of $ax^2 +bx+c=0$, where a, b and c are the constant coefficients and x is the unknown variable. This program can find out the roots (values of x) for you easily.

```
10 REM ROOTS OF QUADRATIC EQUATION
20 CLS
30 PRINT "QUADRATIC EQUATION"
40 PRINT "A*X^2+B*X+C=0"
50 PRINT "ENTER COEFFICIENTS ";
60 PRINT "A,B,C"
70 INPUT A,B,C
80 D=B^2-4*A*C
90 IF D<0 THEN 160
100 D=SQR(D)
110 PRINT "THE ROOTS ARE :"
120 PRINT (-B-D)/(2*A);
130 PRINT (-B+D)/(2*A)
140 GOTO 170
160 PRINT "NO REAL ROOTS"
170 END
```

RUN

```
QUADRATIC EQUATION
A*X^2+B*X+C=0
ENTER COEFFICIENTS A,B,C
? 1,1,-12
THE ROOTS ARE :
-4  3
```

7. AREA OF TRIANGLE

The area of a triangle can be determined once the three sides are fixed. Can you write a program to find out the area of a circle if I can give you the radius.

```
10 REM AREA OF TRIANGLE
20 CLS
30 PRINT "AREA OF TRIANGLE"
40 PRINT "ENTER 3 SIDES"
50 INPUT A,B,C
60 D=.5*(A+B+C)
70 E=D*(D-A)*(D-B)*(D-C)
80 PRINT "AREA IS";SQR(E)
90 END
```

RUN

```
AREA OF TRIANGLE
ENTER 3 SIDES
? 6,8,10
AREA IS 24
Ready
```

8. AREA OF POLYGON

In this program, the area of a regular polygon can be computed. All you have to do is to input the number of sides and its corresponding length.

```
10  REM AREA OR POLYGON
20 CLS:PI=3.1416
30 PRINT "AREA OF REGULAR ";
40 PRINT "POLYGON"
50 INPUT "ENTER NOS. OF SIDES";A
60 INPUT "ENTER LENGTH";B
70 C=PI*(.5*A-1)/A
80 D=A*B*B*TAN(C)/4
90 PRINT "AREA IS";D
100 END
```

```
RUN


AREA OF REGULAR POLYGON
ENTER NOS. OF SIDES? 5
ENTER LENGTH? 4
AREA IS 27.5278
Ready
```

9. RADIAN & DEGREE

This program converts any value in radians to degrees, and vice versa.

```
10 REM RADIAN & DEGREE
20 CLS
30 INPUT "FIND RADIAN(1) OR DEGREE(2)";S
40 IF S=1 THEN 140
50 INPUT "RADIAN";B
60 C=B*180/3.1416
70 IF C>360 THEN C=C-360:GOTO 70
80 PRINT INT(C);"DEGREES"
90 D=(C-INT(C))*60
100 PRINT INT(D);"MINUTES"
110 E=(D-INT(D))*60
120 PRINT INT(E);"SECONDS"
130 END
140 INPUT "DEGREES";A
150 INPUT "MINUTES";B
160 INPUT "SECONDS";C
```

```
170 PRINT
180 D=A+B/60+C/3600
190 IF D>360 THEN D=D-360:GOTO 190
200 D=D*3.1416/180
210 PRINT D;"RADIANS"
220 END
```

```
RUN


FIND RADIAN(1) OR DEGREE(2)? 1
DEGREES? 1
MINUTES? 1
SECONDS? 1

 .0177491 RADIANS
Ready

RUN


FIND RADIAN(1) OR DEGREE(2)? 2
RADIAN? 1
 57 DEGREES
 17 MINUTES
 44 SECONDS
Ready
```

10. FAHRENHEIT & CELSIUS

Similar to Radian & Degree, except to tell you the conversion in temperature.

```
10 REM DEGREE FAHRENHEIT & CELSIUS
20 CLS
30 INPUT "FIND DEGREE-F(1) OR DEGREE-C(2)";A
40 IF A=2 THEN 90
50 INPUT "DEGREE-C";B
60 PRINT B;"DEGREE-C =";
70 PRINT B*9/5+32;"DEGREE-F"
80 END
90 INPUT "DEGREE-F";B
100 PRINT B;"DEGREE-F =";
110 PRINT (B-32)*5/9;"DEGREE-C"
120 END
```

```
RUN


FIND DEGREE-F(1) OR DEGREE-C(2)? 1
DEGREE-C? 0
 0 DEGREE-C = 32 DEGREE-F
Ready


RUN


FIND DEGREE-F(1) OR DEGREE-C(2)? 2
DEGREE-F? 32
 32 DEGREE-F = 0 DEGREE-C
Ready
```

11. FOOT & METRE

Similar to Radian & Degree, except the subjects are Foot & Metre.

```
10 REM FOOT & METRE
20 CLS
30 INPUT "FIND FOOT(1) OR METRE(2)";A
40 IF A=1 THEN 90
50 INPUT "FEET";B
60 PRINT B;"FEET =";
70 PRINT .3048*B;"METRES"
80 END
90 INPUT "METRES";B
100 PRINT B;"METRES =";
110 PRINT B/.3048;"FEET"
120 END
```

```
RUN


FIND FOOT(1) OR METRE(2)? 1
METRES? 1
 1 METRES = 3.28084 FEET
Ready
```

```
RUN
```

```
FIND FOOT(1) OR METRE(2)? 2
FEET? 1
 1 FEET = .3048 METRES
Ready
```

12. POUND & KILOGRAM

Similar to Radian & Degree, except that Pound & Kilogram are being converted.

```
10 REM POUND & KILOGRAM
20 CLS
30 INPUT "FIND POUND(1) OR KILOGRAM(2)";A
40 IF A=1 THEN 90
50 INPUT "POUNDS =";B
60 PRINT B;"POUND =";
70 PRINT .4536*B;"KILOGRAMS"
80 END
90 INPUT "KILOGRAMS";B
100 PRINT B;"KILOGRAMS =";
110 PRINT B/.4536;"POUNDS"
120 END
```

```
RUN
```

```
FIND POUND(1) OR KILOGRAM(2)? 1
KILOGRAMS? 1
 1 KILOGRAMS = 2.20459 POUNDS
Ready
```

```
RUN
```

```
FIND POUND(1) OR KILOGRAM(2)? 2
POUNDS =? 1
 1 POUND = .4536 KILOGRAMS
Ready
```

13. GALLON & LITRE

Similar to Radian & Degree, except that Gallon & Litre are used.

```
10 REM GALLON & LITRE
20 CLS
30 INPUT "FIND GALLON(1) OR LITRE(2)";A
40 IF A=1 THEN 90
50 INPUT"GALLONS";B
60 PRINT B;"GALLONS =";
70 PRINT 3.785*B;"LITRES"
80 END
90 INPUT "LITRE";B
100 PRINT B;"LITRE =";
110 PRINT B/3.785;"GALLONS"
120 END
```

```
RUN


FIND GALLON(1) OR LITRE(2)? 1
LITRE? 1
 1 LITRE = .264201 GALLONS
Ready


RUN


FIND GALLON(1) OR LITRE(2)? 2
GALLONS? 1
 1 GALLONS = 3.785 LITRES
Ready
```

14. DEPRECIATION

The value of most commodities will decrease after a certain period of time. This program calculates the depreciation value (the difference) once you have input the original price, the depreciation are and the timing involved.

```
10 REM DEPRECIATION
20 CLS
30 INPUT "ORIGINAL PRICE";A
40 INPUT "DEPRECIATION RATE(%)";B
50 INPUT "NO. OF YEARS";C
60 PRINT "DEPRECIATION =";
70 B=B/100
80 D=A*B*(1-B)^(C-1)
90 D=INT(D*10+.5)/10
100 PRINT D:END
```

```
RUN
```

```
ORIGINAL PRICE? 1000
DEPRECIATION RATE(%)? 10
DEPRECIATION = 65.6
Ready
```

15. SORTING NUMBERS

If you input a group of numbers (from 2 to 20), this program will sort the numbers in an ascending order. Can you modify the program in the way that it can sort the numbers in a descending order?

```
10 REM SORTING NOS. IN ASCENDING ORDER
20 CLS
30 PRINT "SORTING NOS.(2-10)"
40 INPUT "HOW MANY NOS.";A
50 DIM A(19)
60 FOR I= 1 TO A
70 PRINT "NO.";I;:INPUT A(I-1)
80 NEXT I
90 FOR J= 0 TO A-2
100 FOR I= 0 TO A-2
110 IF A(I)<A(I+1) THEN 130
120 B=A(I):A(I)=A(I+1):A(I+1)=B
```

```
130 NEXT:NEXT
140 FOR I=0 TO A-1
150 PRINT A(I);
160 NEXT
170 END


RUN


SORTING NOS.(2-10)
HOW MANY NOS.? 6
NO. 1 ? 6
NO. 2 ? 5
NO. 3 ? 4
NO. 4 ? 3
NO. 5 ? 2
NO. 6 ? 1
 1  2  3  4  5  6
Ready
```

16. SORTING WORDS

This program sorts a group of words (from 2 to 10) in an alphabetic order.

```
10 REM SORTING WORDS IN ALPHABETIC ORDER
20 CLS
30 PRINT "SORTING WORDS(2-10)"
40 INPUT "HOW MANY WORDS";A
50 DIM A$(9)
60 FOR I= 1 TO A
70 PRINT "WORD";I;:INPUT A$(I-1)
80 NEXT I
90 FOR J= 0 TO A-2
100 FOR I= 0 TO A-2
110 IF A$(I)<A$(I+1) THEN 130
120 B$=A$(I):A$(I)=A$(I+1):A$(I+1)=B$
130 NEXT:NEXT
140 FOR I=0 TO A-1
150 PRINT A$(I);"  ";
160 NEXT
170 END
```

```
RUN


SORTING WORDS(2-10)
HOW MANY WORDS? 6
WORD 1 ? ZOO
WORD 2 ? FAST
WORD 3 ? LAZY
WORD 4 ? EAT
WORD 5 ? EAR
WORD 6 ? HELLO
EAR EAT FAST HELLO LAZY ZOO
Ready
```

17. NUMBER GUESSING

The computer will generate a number in random (from 1 to 1000) and you will have to guess what is the pre-selected number. How many trials you have to attempt?

```
10 REM GUESS A NUMBER
20 CLS:C=1:RANDOMIZE
30 A=INT(RND*1000)+1
40 PRINT"GUESS A NUMBER"
50 INPUT "(1-1000)";B
60 IF B>A THEN PRINT "SMALLER"
70 IF B<A THEN PRINT "LARGER"
80 IF B=A THEN 100
90 C=C+1:GOTO 40
100 PRINT "YOU ARE RIGHT"
110 PRINT "YOU HAVE TRIED";C;
120 PRINT "TIMES"
130 END
```

```
RUN


GUESS A NUMBER
(1-1000)? 500
SMALLER
```

```
GUESS A NUMBER
(1-1000)? 250
LARGER
GUESS A NUMBER
(1-1000)? 300
YOU ARE RIGHT
YOU HAVE TRIED 3 TIMES
Ready
```

18. WORD GUESSING

This time you have to guess a 4 letter word. The method of playing is similar to the Number Guessing.

```
10 REM GUESS A WORD
20 CLS:RANDOMIZE
30 C$="FISHRUSHRESTSIDETALKDIRTWORKGIRLJUMPMOOD"
40 I=(INT(RND*10))*4+1
50 A$=MID$(C$,I,4):S=1
60 PRINT "GUESS A WORD"
70 INPUT "(4 LETTERS)";B$
80 FOR J=1 TO 4
90 IF MID$(A$,1,J)=MID$(B$,1,J) THEN NEXT
100 PRINT "YOU HAVE";J-1;
110 PRINT "LETTERS RIGHT"
120 IF J<>5 THEN S=S+1:GOTO 60
130 PRINT "YOU HAVE TRIED";S;
140 PRINT "TIMES"
150 END
```

RUN

```
GUESS A WORD
(4 LETTERS)? R
YOU HAVE 0 LETTERS RIGHT
GUESS A WORD
(4 LETTERS)? S
YOU HAVE 1 LETTERS RIGHT
GUESS A WORD
(4 LETTERS)? SIDE
YOU HAVE 4 LETTERS RIGHT
YOU HAVE TRIED 3 TIMES
Ready
```

19. RANDOM GRAPHICS

Random graphic is generated by making use of the pre-defined graphic characters.

```
10 REM GRAPHIC
20 CLS:RANDOMIZE
30 COLOR INT(RND*15)
40 N=INT(RND*959)
50 Y=INT(N/40):X=N-INT(N/40)*40
60 PRINT CHR$(27);CHR$(161);CHR$(X+32);
CHR$(Y+32);"*";
70 GOTO 30
```

20. MELODY

You can write and play your own song. All you have to do is to select the frequency code and the duration code of each note. However, the maximum number of notes that you can play at one time will depend on the memory size of your computer.

```
10 REM SONG
20 CLS
30 INPUT "ENTER NO. OF NOTES";N
40 PRINT "ENTER YOUR NOTES"
50 DIM A%(2*N-1)
60 FOR I=0 TO N-1
70 INPUT "FREQUENCY CODE";A%(I*2)
80 INPUT "DURATION CODE";A%(I*2+1)
90 NEXT
100 FOR I = 0 TO N-1
110 SOUND A%(I*2),A%(I*2+1)
120 NEXT

RUN
```

```
ENTER NO. OF NOTES? 8
ENTER YOUR NOTES
FREQUENCY CODE? 26
DURATION CODE? 3
FREQUENCY CODE? 30
DURATION CODE? 3
```

```
FREQUENCY CODE? 28
DURATION CODE? 3
FREQUENCY CODE? 21
DURATION CODE? 5
FREQUENCY CODE? 26
DURATION CODE? 3
FREQUENCY CODE? 28
DURATION CODE? 3
FREQUENCY CODE? 30
DURATION CODE? 3
FREQUENCY CODE? 26
DURATION CODE? 7
Ready
```

21. MARK SIX

It will generate 6 random numbers with one extra special number.

```
10 REM MARK SIX
20 CLS:RANDOMIZE
30 FOR I = 1 TO 7
40 A(I)=INT(RND*36)+1
50 IF I = 1 THEN 90
60 FOR J=1 TO I-1
70 IF A(I) = A(J) THEN 40
80 NEXT
90 NEXT
100 PRINT "THE NOS. ARE :"
110 FOR I = 1 TO 5
120 FOR J = 1 TO 5
130 IF A(J)<A(J+1) THEN 150
140 B=A(J):A(J)=A(J+1):A(J+1)=B
150 NEXT:NEXT
160 FOR J = 1 TO 6
170 PRINT A(J);
180 NEXT
190 PRINT
200 PRINT "SPECIAL NO. IS :"
210 PRINT A(7)
220 END
```

```
RUN


THE NOS. ARE :
 4  17  23  30  33  34
SPECIAL NO. IS :
 9
Ready
```

22. RAM DISK

The following program demonstrates how to use the extra memory that are not used by the BASIC interpreter as a RAM disk. After the RAM disk is installed, the user can type 'SAVE' to save his program in the RAM disk and retrieve it with the 'LOAD' command.

The BASIC program is used to poke an machine code program into the RAM. The assembly language program listing is also included to demonstrate how to perform bank-switching.

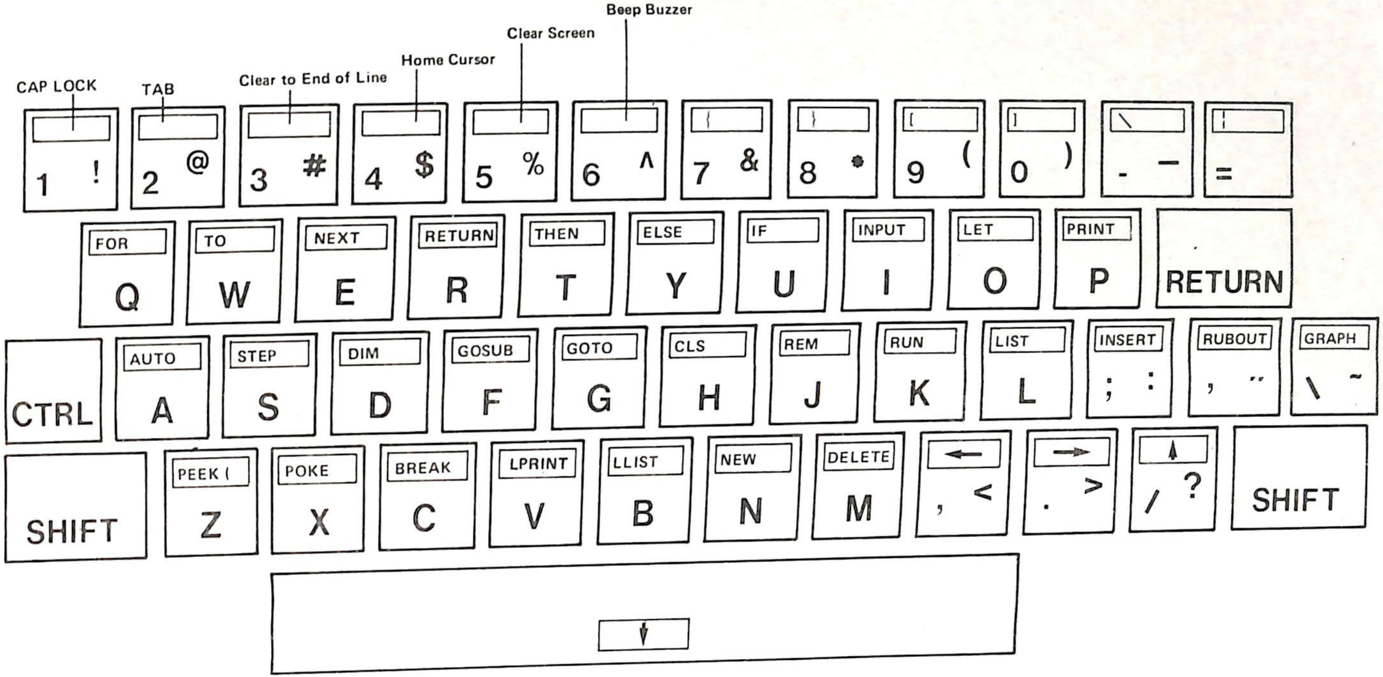
```
10 REM SET UP LOAD COMMAND VECTOR
20 POKE &H859E,&HB0
30 POKE &H859F,&H86
40 REM SET UP SAVE COMMAND VECTOR
50 POKE &H85B3,&HB3
60 POKE &H85B4,&H86
70 REM POKE MACHINE CODE PROGRAM INTO
RAM
80 FOR AD=&H86B0 TO &H8720
90 READ OP
100 POKE AD,OP
110 NEXT
120 PRINT "RAM DISK INSTALLED"
130 END
140 DATA &HAF,&H18,&H02,&H3E,&H01,&HF5
,&H2A,&HE9,&H83
150 DATA &HED,&H5B,&H41,&H80,&HB7,&HED
,&H52
160 DATA &H44,&H4D,&H11,&HFF,&H21,&H7A
,&H94,&H38,&H3A
170 DATA &H20,&H04,&H7B,&H95,&H38,&H34
,&H3E,&H06,&H32,&H66,&H86
```

```
180 DATA &HD3,&H41,&HF1,&HB7,&H28,&H16
.&HED,&H43,&H21,&H87,&H11,&H00,&H5E
190 DATA &H2A,&H41,&H80,&HED,&HB0,&H3E
,&H01,&H32,&H66,&H86,&HD3,&H41
200 DATA &HC3,&H59,&H00,&HED,&H4B,&H21
,&H87,&HED,&H5B,&H41,&H80
210 DATA &H62,&H6B,&H09,&H22,&HE9,&H83
,&H21,&H00,&H5E,&H18,&HE1,&HF1
220 DATA &H21,&H12,&H87,&H7E,&HB7,&HCA
,&H59,&H00,&HCD,&H2B,&H00,&H23,&H18,&H
F5
230 DATA &H53,&H49,&H5A,&H45,&H20,&H54
,&H4F,&H4F,&H20,&H4C,&H41,&H52,&H47,&H
45,&H00
```

```
;**************************************************
;
;   USE EXTRA RAM AS RAM DISK FOR LASER 350/500/700
;
;   ADDRESS 15E00-17FFF ARE NOT USED BY THE BASIC
;   INTEPRETER AND CAN BE USE BY USER
;
;**************************************************
;
VARTAB  EQU   83E9H
TXTTAB  EQU   8041H
BANK1   EQU   8666H
READY   EQU   0059H          ;ENTRY POINT OF BASIC
OUTDO   EQU   002BH          ;OUTPUT DRIVER

        ORG   86B0H          ;USE DEFAULT BUFFER TO SAVE P
                             ROGRAM
LOAD:
        XOR   A              ;[A]=0 FOR LOAD
        JR    SAVE1
SAVE
        LD    A,1
SAVE1
        PUSH  AF
        LD    HL,(VARTAB)    ;CALCULATE WHETHER SIZE OF P
                             ROGRAM IS TOO LARGE
        LD    DE,(TXTTAB)
        OR    A
        SBC   HL,DE
        LD    B,H
        LD    C,L
        LD    DE,21FFH       ;MAX SIZE FOR RAM DISK
        LD    A,D
        SUB   H
        JR    C,NORAM
        JR    NZ,RAMOK
        LD    A,E
        SUB   L
```

```
              JR    C,NORAM
   RAMOK
              LD    A,6            ;PUT PHYSICAL BANK 6 TO LOGIC
                                   AL BANK 1
              LD    (BANK1),A      ;UPDATE BANK1 IMAGE
              OUT   (41H),A        ;CHANGE THE BANK
              POP   AF
              OR    A              ;SEE IF SAVE OR LOAD
              JR    Z,LOAD1        ;LOAD
              LD    (LENGTH),BC
              LD    DE,5E00H       ;STARTING ADDRESS OF RAM DISK
              LD    HL,(TXTTAB)     ;GET PROGRAM STARTING ADDRES
                                   S
   SAVE3
              LDIR                 ;SAVE PROGRAM IN RAMDISK
              LD    A,1            ;RESTORE PHYSICAL BANK 1 TO L
                                   OGICAL BANK 1
              LD    (BANK1),A
              OUT   (41H),A
              JP    READY
   LOAD1                          ;ACTUAL CODE FOR LOAD
              LD    BC,(LENGTH)   ;GET LENGTH OF PROGRAM IN RA
                                  MDISK
              LD    DE,(TXTTAB)
              LD    H,D
              LD    L,E
              ADD   HL,BC
              LD    (VARTAB),HL   ;UPDATE END OF PROGRAM ADDRE
                                  SS
              LD    HL,5E00H
              JR    SAVE3
   NORAM
              POP   AF            ;CLEAR STACK
              LD    HL,MSG
   NORAM1
              LD    A,(HL)        ;PRINT 'SIZE TOO LARGE' MESSA
                                  GE
              OR    A
              JP    Z,READY
              CALL  OUTDO         ;OUTPUT THE CHAR
              INC   HL
              JR    NORAM1
   MSG        DEFM  'SIZE TOO LARGE'


              DEFB  0H
   LENGTH     DEFS  2             ;STORE SIZE OF PROGRAM
              END
```

LASER 350 / 500 / 700 Keyboard—Control Keys



NON-DISCLOSURE AGREEMENT
AND
REGISTRATION FORM

The party below agrees that it is receiving a copy of Microsoft BASIC for use on a single computer only, as designated on this non-disclosure agreement. The party agrees that all copies will be strictly safeguarded against disclosure to or use by persons not authorized by Video Technology Computers, Ltd. to use Microsoft BASIC, and that the location of all copies will be reported to Video Technology Computers, Ltd at Video Technology Computers, Ltd's request. The party agrees that copying or unauthorized disclosure will cause great damage to Video Technology Computers, Ltd. and the damage is far greater than the value of the copies involved. The party agrees that this agreement shall inure to the benefit of any third party holding any right, title or interest in Microsoft BASIC or any software from which it was derived.

Purchased From:

Company

Address

Phone

For Use On:

Model

Serial #

Software Product

Purchased By: (Distributor)

Name

Company

Address

Phone

Date

Purchased By: (Dealer)

Name

Company

Address

Phone

Date

Purchased By: (End-User)

Name

Company

Address

Phone

Date

NOTE: The Non-Disclosure Agreement MUST be signed by Party purchasing product directly from Video Technology Computers Ltd.. No Product will be shipped without signed agreement. It is the responsibility of Distributor and/or Dealer to transfer ownership to appropriate party.

Completed form should be returned to the following address:

VIDEO TECHNOLOGY COMPUTERS LTD.
23/F., Tai Ping Ind. Centre, Blk. 1,
57 Ting Kok Rd., Nam Hang, Tai Po,
N.T., Hong Kong.

NON-DISCLOSURE AGREEMENT
AND
REGISTRATION FORM

The party below agrees that it is receiving a copy of Microsoft BASIC for use on a single computer only, as designated on this non-disclosure agreement. The party agrees that all copies will be strictly safeguarded against disclosure to or use by persons not authorized by Video Technology Computers, Ltd. to use Microsoft BASIC, and that the location of all copies will be reported to Video Technology Computers, Ltd at Video Technology Computers, Ltd's request. The party agrees that copying or unauthorized disclosure will cause great damage to Video Technology Computers, Ltd. and the damage is far greater than the value of the copies involved. The party agrees that this agreement shall inure to the benefit of any third party holding any right, title or interest in Microsoft BASIC or any software from which it was derived.

Purchased From:

Company

Address

Phone

For Use On:

Model

Serial #

Software Product

Purchased By: (Distributor)

Name

Company

Address

Phone

Date

Purchased By: (Dealer)

Name

Company

Address

Phone

Date

Purchased By: (End-User)

Name

Company

Address

Phone

Date

NOTE: The Non-Disclosure Agreement MUST be signed by Party purchasing product directly from Video Technology Computers Ltd.. No Product will be shipped without signed agreement. It is the responsibility of Distributor and/or Dealer to transfer ownership to appropriate party.

Completed form should be returned to the following address:

VIDEO TECHNOLOGY COMPUTERS LTD.
23/F., Tai Ping Ind. Centre, Blk. 1,
57 Ting Kok Rd., Nam Hang, Tai Po,
N.T., Hong Kong.

LASER

COLOR COMPUTER

350 / 500 / 700