introduction to
computer electronics

COMPUKIT 1

INSTRUCTIONS FOR ASSEMBLY AND USE

LIMROSE ELECTRONICS LYMM CHESHIRE

ERRATA

Page 3 Eq.(1.3)	should read as	$110 = 1x2^2 + 1x2^1 + 0x2^0$
Page 4 Line 1	should read as	.... numbers from 0 to 9 are,
Page 5 Line 6	should read as	Whereas in binary arithmetic 0 + 0 = 0, ...
Page 7 Fig.1.7		Basic OR should read as Basic AND Basic NOR should read as Basic NAND Basic AND should read as Basic OR Basic NAND should read as Basic NOR
Page 9 Line 15	should read as	$0 + 0 = 0$ $0.0 = 0$ $0 + X = X$ $X + \bar{X} = 1$
Page 9 Para 3.2 Line 2	should read as	.... The first law, equation (3.1),...
Page 13 Para 5.3 Line 2	should read as	bit number, except 000, using .....

HOW TO ASSEMBLE COMPUKIT 1

The assembly of Compukit 1 is very simple and should take you about an hour to complete. You will need a small soldering iron and a pair of cutting pliers to assemble the components onto the printed circuit board. The kit contains:

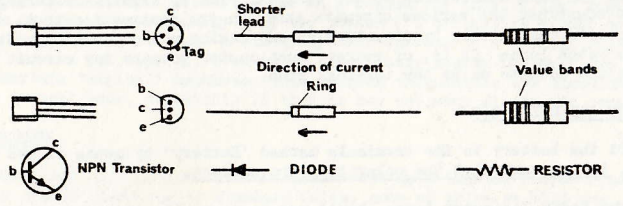
One Transistor pack with 14 NPN Silicon Transistors.
One Diode pack with 18 Gold-bonded Germanium diodes.
One Resistor pack I with 25 10 K Resistors.
One Resistor pack II with 15 1 K Resistors and 2 150 Ohms.
One Terminal pin pack
One Logic Indicator pack with 2 wire-ended bulbs,
2 plastic clips and
2 High gain Transistors.
One Printed Circuit, $5\frac{1}{2}$"x $9\frac{3}{4}$"
One $4\frac{1}{2}$ Volt Battery, Ever Ready Type 126.
One Instruction Book.
Wire, about 6 to 8 yds.
Solder, about 4 ft of pre-fluxed type.

Optional Extras: One Soldering Iron, 15-20 W, 250 Volt a.c.(31/- each)
Cutting Pliers (6/- each)
Solderless patch lead packs (27/6 for 5 double-ended leads)

All components supplied are in perfect condition, have been tested to our specifications and are ready to be assembled. The kit contains two extra transistors and two extra diodes to serve as spares in case you damage any during assembly. Most of you should have no difficulty in identifying the components. However, those of you not familiar with transistors, diodes, and resistors should find the following notes useful.

Transistors : A transistor usually consists of a small metal or plastic body having three wires sticking out of it at one end. Fig. A.1 shows the outline of two types of transistors, together with the symbol for the NPN type used in the kit. The three leads, marked e, b and c in the figure, can be easily identified on the transistors themselves by their relationship to the shape of the body, or the 'tag'.

Diodes: Each diode consists of a small tubular glass body, either with an 'identification' ring on one side or with one lead shorter than the other. Diodes conduct towards the ring, or towards the shorter lead, as the case may be, fig. A.2. (The latter coding is not generally used, but is a special coding peculiar to this kit only.)



Resistors : The resistors have an opaque body with colour coded bands to indicate their value, fig.A.3. The resistors supplied to you have coded rings as follows:
10 K- Brown, Black, Orange 1 K- Brown, Black, Red 150 Ohm- Brown, Green, Brown.

ASSEMBLY PROCEDURE

The Printed Circuit Board has tinned copper tracks on one side, and is screen printed on the other side with logic symbols and component identifications. On the printed side, the words 'Compukit 1' roughly divide the board into two halves. The upper half is the patching area and the terminal pins are assembled in this area. The lower half is marked where all the electronic components, except the bulbs, are to be mounted. The two transistors supplied with the logic indicator pack occupy the transistor positions on the extreme left of the board. Do not mix these transistors with the rest. Now proceed as follows:

1. Insert resistors in the lower half of the board as indicated, and push the leads in so that the resistor body comes into contact with the board. Solder the leads on the other side and cut off excess leads.

2. Spread and push the leads of the transistors through the holes marked with circles around them, being careful to 'align' the leads correctly so that the emmiter, base and collector leads go through the appropriate holes. Only two typical transistor positions have been marked with e, b and c lead positions. Solder the leads on the other side, using minimum heat and a small amount of solder. Cut off excess leads. Overheating may damage the transistors.

3. Align diode leads so that they conduct in the direction of the arrow printed on the board i.e. with the identification ring or the shorter lead upwards. Push in the leads after bending them gently, and carefully solder on the other side of the board using minimum heat and a small amount of solder. Overheating during soldering can damage the diodes if prolonged contact with the hot iron is maintained for more than a few seconds at a time.

4. The terminal pins should now be inserted from the copper side and gently pushed in as far as they will go. Solder the pins to the copper side lightly.

5. Now carefully assemble the wire-ended bulbs in the plastic clips, and push the clips in the $\frac{1}{8}$" dia. holes just above the area marked 'Logic Indicator'. Push the wire ends of the bulbs through the holes for them and solder on the copper side. Cut off any excess leads.

The assembly of the basic kit is now complete. If you have purchased the solderless patch lead packs as optional extras, you can assemble the patch leads from them by attaching a socket at each end of a small length of the flexible wire supplied. The bare wire end is soldered to the metal part, and the plastic part is pushed on top of it from behind to complete the assembly.

USING COMPUKIT 1

You may use the cardboard box in which the kit has been supplied as a base for your assembled board. The battery can be left in the box in its original position forming, together with the inserts, a platform upon which the assembled board can remain. Once the battery connections (see below) have been made, the board is ready for use.

You can make connections to the terminal pins by lightly soldering a wire to them to construct the various circuits shown in the instruction book. Alterations to the circuits can be made by de-soldering and making new connections. Using the solderless patch leads it is, of course, much easier to make any circuit simply by pushing the sockets on to the terminal pins.

TESTING THE ASSEMBLED KIT

1. Connect the battery to the terminals marked 'Battery' by means of two short leads, taking care that the polarities are correct.

2. Both the Logic Indicator lamps should now be partially lit.

3. Attach a wire, or patch lead, to the input pin of one of the logic indicator lamps, and connect the other end to the Logic 1 and Logic 0 buses alternately. Logic 1 should increase the brightness of the lamp filament, whereas logic 0 should cause no change in the brightness level.

4. Check NOR gates one by one. With no inputs connected, the output should be at logic 1. If any input is connected to logic 1, the output should be logic 0.

5. Check NAND gates. With no inputs connected, the output should be at logic 0. If any input is connected to logic 0, the output should be logic 1.

If the assembled kit passes these tests, the kit is now ready to be used. If however any of the tests fail, look for unsoldered or poorly soldered connections, broken copper tracks in the printed circuit board, or faulty components. Make sure that the diodes and transistors have been correctly connected.

THE DIGITAL COMPUTER

1.1

Someone once said that he could build complex structures if only he could think of the first thing that had to be done! Having overcome the initial hurdle, the remaining steps are likely to fall readily into place. Actually this is not far from the truth, and with luck and some effort on your part it should be possible for you to understand how digital computers work. Now that you have taken the first step, you may even eventually be able to build a digital computer for yourself. First of all, we will have a brief look at the general principles on which digital computers are based.

Fig.1.1 shows a block diagram of a digital computer, from which you can see the various parts and their functional relationships. A digital computer receives the problem and data through an Input Unit. This usually consists of a paper tape reader, a magnetic tape unit, or more recently introduced optical character reading units. A step by step procedure for working out the results, called a programme, is fed to the computer via the input unit. These instructions are stored in a Memory inside the computer, for rapid retrieval later on when required. The calculating function is performed by the Central Processing Unit, also called the Arithmetic Unit. However, no calculations can be really performed without the overall control and timing provided by the Cotrol unit. The results of computation are usually printed on an electric typewriter, or a high speed line printer, which constitutes the Output unit. This represents a simplified overall view of a digital computer.

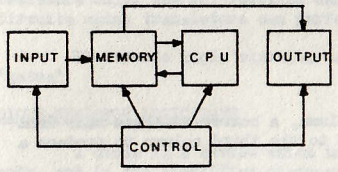


FIG 1.1

All signals in a digital computer have to be suitably coded in order to represent the numerous physical variables which are of interest to commercial, scientific or other users. Most digital computers use the Binary code for representing alpha-numeric data. We will now consider some aspects of the binary code.

BINARY CODE

1.1

Design of computers is greatly simplified by choosing certain codes, rather than others. We are all very familiar with the decimal number system, or the decimal code, which unfortunately is not entirely suitable for use in computers using electronic circuits. The main reason for this is that most electronic devices have basically only two states (a circuit is either on or off), forming a two-state i.e. a binary system. Whereas in a decimal system, there are numbers 0,1,2,3,4,5,6,7,8 and 9, the binary system has only two numbers, 0 and 1. Certain 'digital' machines, such as desk calculators are sometimes made using the decimal code, especially if they do not use many electronic components.

BINARY NUMBERS

1.2

Numerical calculations can be carried out in a digital computer using the binary system according to standard rules, some of which we will now examine. In the decimal notation, to express a number greater than 9 we use a system of 'weighted' positions using 10 as the base, thus

$$3127 = 3\times10^3 + 1\times10^2 + 2\times10^1 + 7\times10^0 \qquad (1.1)$$

In the binary system, large numbers are represented in a similar manner using 2 as the base. Thus, a large binary number is made up as follows,

$$101101 = 1\times2^5 + 0\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 \qquad (1.2)$$

$$110 = 1\times2^2 + 1\times2^1 + 1\times2^0 \qquad (1.3)$$

It is easy for you to work out that the binary number 101101 is equal to decimal 45, and that binary 110 is equal to decimal 6.

The binary equivalents of decimal numbers from 1 to 9 are,

0000 0001 0010 0011 0100 0101 0110 0111 1000 1001

BINARY ADDITION 1.3

Adding two binary numbers is very similar to adding two decimal numbers, except that in a binary system the highest number allowed is a 1 whereas in the decimal system the corresponding number is 9. In binary addition, when a 1 and a 0 are added together, the result is a 1 and there is no carry. If a binary 1 is added to another binary 1, the result is a 0 and there is a 1 carry. Thus, we add as follows,

101 (Decimal 5)	111 (Decimal 7)
010 (Decimal 2)	101 (Decimal 5)
111 (Decimal 7)	1100 (Decimal 12)

In the second example, adding 1 and 1 generates a carry in the first, second and third places to produce the 1's in the third and fourth places.

BINARY SUBTRACTION 1.4

There are several methods for performing binary subtraction, but the simplest of them all to understand is the 'direct method'. Example of binary subtraction using this method is,

1001 (Decimal 9)
101 (Decimal 5)
0100 (Decimal 4)

To subtract 1 from 0 in the third column, a borrow of 1 was made from the fourth column, which effectively added 2 to the third column to produce a 1 there and a 0 in the fourth column.

TWO'S COMPLEMENT ARITHMETIC 1.5

The method outlined above is, however, not very convenient for use in a digital computer. If this method were to be used, separate circuits would be required to perform binary addition and subtraction. For this reason, the subtraction in a digital computer is performed by adding the equivalent 'negative' number. Such a number is known as the 'complement' of the original number. Most digital computers use 'Two's Complement', both to represent negative numbers and to perform binary subtraction. The two's complement of a binary number is defined as the number which when added to the original number will result in a sum of all zeros except in the 'overflow' position. The easiest method of finding a two's complement for a binary number is to first obtain the 'one's complement' and adding a 1 to it. One's complement is formed simply by setting each bit to the opposite value. Thus,

0101 (Number)
1010 (One's complement)
0001 (Add 1)
1011 (Two's complement)

Subtraction is performed using the two's complement of the number to be subtracted. That is, to subtract A from B, we express A in its two's complement and add the value of B to it. Thus,

0101 (A)
1011 (Two's complement of A)
1001 (Add B)
1 0100 (Result B-A, ignoring the 'overflow')

BINARY LOGIC 1.6

In addition to using 0 and 1 for expressing numbers, they are also used in performing logic. Rules of binary logic form a Boolean algebra and are explained in a later section.

Boolean algebra is named after George Boole, a British philosopher. who was mainly interested in applying logic to man's intellect and thought process. Boole's logic is based upon the premise that a proposition is either 'true' or 'false'. True statements are given a logical value of 1, whilst false statements have a value of 0.

Whereas in binary arithmetic $0+0=1$, $0+1=1$, and $1+1=10$, in the binary logic of Boolean algebra, the symbol '+' is used to represent the 'OR' function. Thus in logic $0+0=0$, $0+1=1$, and $1+1=1$. In Boolean algebra, AB or A.B represents the 'AND' function. In terms of 1 and 0, the results are $0.0=0$, $0.1=0$ and $1.1=1$, which are similar to the rules of ordinary multiplication.

In addition to 'AND' and 'OR' functions, there is also a 'NOT' function in Boolean algebra, which is indicated by a 'bar' over the symbol. Thus $\overline{A}$ (NOT A), $\overline{X}$ (NOT X) etc. If A is 0, then $\overline{A}=1$, and if $A=1$ then $\overline{A}=0$.

Binary logic functions of NOT, OR and AND taken together are more than sufficient for carrying out all computation and control functions in a digital computer. These can also be used to form the memory. There are many electronic devices which can be used to perform these logic functions in a digital computer. Most practical electronic logic circuits utilise relays, diodes, transistors, resistors and magnetic cores. More recent additions to this range of components are tunnel diodes, integrated circuits and field-effect transistors. It is possible to perform the logical operations of OR and AND using diodes and resistors only. The NOT function cannot be directly performed in this manner. Circuits using transistors can perform the NOT operation directly.

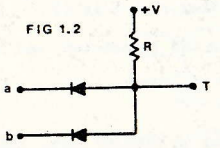
Electronic logic circuits used in a computer are generally known as 'gates'.

DIODE-RESISTOR LOGIC GATE 1.7

A diode is a device which ideally conducts current in one direction only, and in the simplified discussion which follows we will assume that the diodes have zero forward resistance and infinite reverse resistance.

There are two general diode-resistor configurations which can be used to perform logic operations, fig.1.2 and 1.3.

FIG 1.2



AND Gate

OR Gate

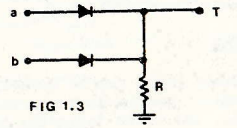


FIG 1.3

In fig.1.2, if either input is at logic 0 i.e. zero volts, a current would flow through the resistor R and the output at T will be logic 0. To obtain a logic 1 output at T, both inputs a and b must be at logic 1. Thus, in terms of Boolean notation for binary logic, we have,

$$T = a.b \quad \text{(i.e. T equals 1 if both a and b are 1)} \quad (1.4)$$

This is the basic diode-resistor AND gate.

For the circuit in fig. 1.3, output T will be at logic 1 if either of the inputs a or b is at logic 1. This is the basic OR function given by,

$$T = a + b \quad \text{(i.e. T equals 1 if either a or b is 1)} \quad (1.5)$$

TRANSISTOR-DIODE-RESISTOR GATE 1.8

The diode-resistor gate can perform the NOT operation if a transistor is added to the gate output. This transistor also provides useful power gain which can be used to drive more circuits from a single output. Because diodes

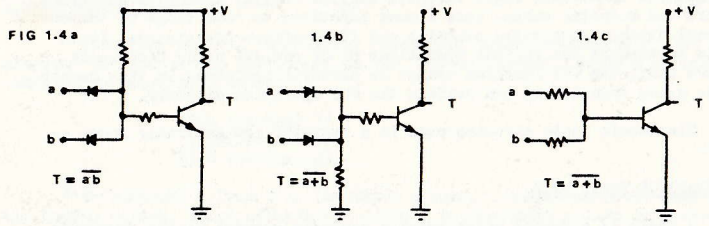
do not have a zero forward resistance, there is serious deterioration in voltage levels if multiple diode gates are used in series. Inclusion of the transistor provides the necessary amplification to counteract this deterioration.

Many of you are, perhaps, familiar with the operation of a transistor. In brief, the transistor acts as a 'current amplifier' and a small 'base current' can be used to control a larger current in the 'collector' circuit. The ratio between the controlled collector current and the controlling base current is known as the 'gain' of the transistor. Typical values for current gains in transistors are between 20 to 150.

Some schematic logic circuits using transistors are shown in fig.1.4a, b and c. In the configuration shown in fig.1.4a, if the inputs a and b are both at logic 1, the potential at the base of the transistor is sufficiently high to turn it 'on'. This results in T being pulled down to logic 0, thus,

$$T = \overline{a.b} \quad \text{(i.e. T equals 0 if both a and b are 1)} \qquad (1.6)$$

This operation is known as NAND, which stands for NOT AND taken together.



Circuits in figs.1.4b and c are used to perform the logic NOR (or NOT OR) operation. If either of the inputs a or b is at logic 1, the transistor is turned on and the output goes to logic 0. Thus,

$$T = \overline{a+b} \quad \text{(i.e. T equals 0 if either a or b is 1)} \qquad (1.7)$$

The circuit in fig.1.4c uses only resistors and transistor, and is known as 'Resistor-transistor logic'.

TRUTH TABLES 1.9

Binary logic can be conveniently represented in tabular form. This is known as a 'Truth table' and is commonly used to show the relationship between logic variables. For example, the truth table for NOT, OR and AND functions can be written as,

a	0	0	1	1	
b	0	1	0	1	
T = a + b	0	1	1	1	(logic OR)
T = a.b	0	0	0	1	(logic AND)
$\overline{a}$	1	1	0	0	(NOT a)
$\overline{b}$	1	0	1	0	(NOT b)

Similarly, the truth tables for the NOR and NAND functions are given as,

a	0	0	1	1
b	0	1	0	1
$T = \overline{a+b}$	1	0	0	0
$T = \overline{a.b}$	1	1	1	0

LOGIC IN COMPUKIT 1 1.10

You have been provided with logic gates in Compukit 1, similar to the ones which we have just discussed. The circuits used are shown in figs. 1.5a, b, c, d and e and have been specially selected for their simplicity and low cost. Fig. 1.5e, the logic indicator circuit, is merely a NOT gate driving a small lamp. This lamp brightens up when the input is at logic 1. The transistors used are NPN silicon type and the diodes are gold-bonded germanium types.

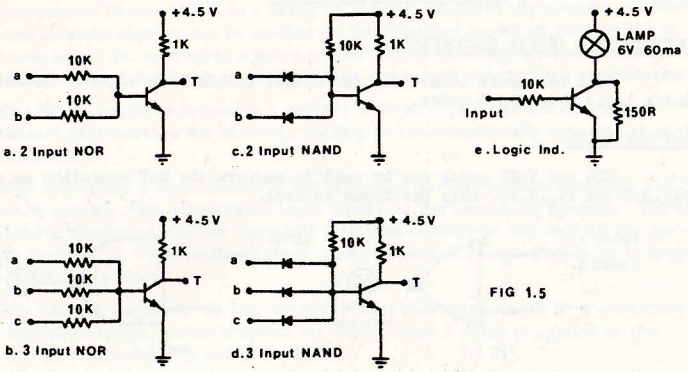


FIG 1.5

NOT LOGIC WITH NOR AND NAND GATES 1.11

It is obvious from the previous discussion, that the logic NOT operation can be performed using NOR and NAND gates, if only one of the inputs is used.

LOGIC SYMBOLS 1.12

In designing large digital systems, such as digital computers, it is convenient to use symbols to represent different operations. Unfortunately, there is no universally accepted set of symbols and many different types are in common use. The two types most commonly used in this country are specified in the British Standards B.S. 350- Graphic Symbols for Telecommunications, and the American Military Standards, MIL-STD-806 B Logic Symbols. Some of the most important of these are shown in figs.1.6 and 1.7 for your reference. Throughout these notes, however, we have used the more popular MIL-STD-806 B Logic Symbols.

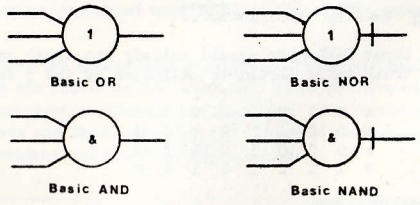


FIG 1.6 B.S.350 GRAPHIC SYMBOLS FOR TELECOMMUNICATIONS

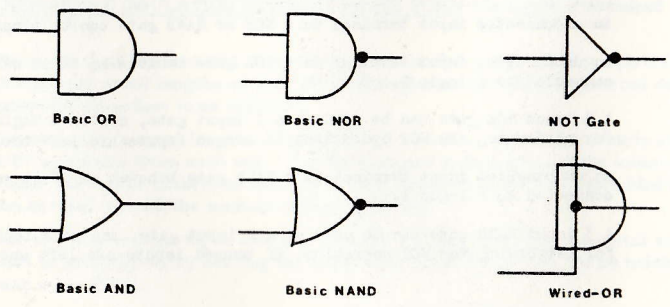


FIG 1.7 MIL-STD-806B LOGIC SYMBOLS

LOGIC WITH COMPUKIT GATES

OUTPUTS WITH INPUTS DISCONNECTED 2.1

With no inputs connected, the output should be a logic 1 for NOR gates and a logic 0 for NAND gates.

NOT OPERATION 2.2

NOR and NAND gates can be used to perform the NOT operation as shown in fig.2.1 and fig.2.2. Only one input is used.

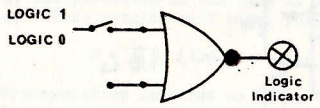


FIG 2.1

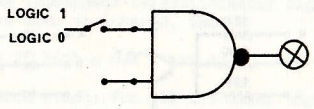


FIG 2.2

Output logic level should be the complement of the input, i.e. the logic indicator lamp should brighten up when the gate input is connected to logic 0 and should be off when it is connected to logic 1.

NOR FUNCTIONS $T = \overline{a+b}$, and $T = \overline{a+b+c}$ 2.3

Connect the output of a 2 input NOR gate to a logic indicator lamp. Note that the lamp goes dim if any one of the inputs is connected to logic 1. Verify that this gate satisfies the truth table given in section 1.9.

Repeat with a 3 input NOR gate, and verify that the following truth table is satisfied,

a	0	1	0	0	1	0	1	1
b	0	0	1	0	1	1	0	1
c	0	0	0	1	0	1	1	1
T	1	0	0	0	0	0	0	0

NAND FUNCTIONS $T = \overline{a.b}$, and $T = \overline{a.b.c}$ 2.4

The 2 input NAND gate should satisfy the truth table given in section 1.9. The following truth table should be satisfied by the 3 input NAND gates,

a	0	1	0	0	1	0	1	1
b	0	0	1	0	1	1	0	1
c	0	0	0	1	0	1	1	1
T	1	1	1	1	1	1	1	0

SUMMARY OF GATE OPERATION 2.5

From these simple experiments with Compukit gates, you will note that:

An unconnected input terminal in a NOR or NAND gate can be ignored.

An unconnected input terminal in a NOR gate behaves as if it were connected to a logic 0.

A 3 input NOR gate can be used as a 2 input gate, and as a logic inverter for performing the NOT operation, if unused inputs are left unconnected.

An unconnected input terminal in a NAND gate behaves as if it were connected to a logic 1.

A 3 input NAND gate can be used as a 2 input gate, and as a logic inverter for performing the NOT operation, if unused inputs are left unconnected.

FLOATING INPUTS IN LOGIC CIRCUITS 2.6

Some confusion may be caused if any active inputs are left 'floating' in logic circuits shown in the later sections, and incorrect results will be obtained when verifying truth tables. In all such cases, the inputs must be appropriately connected to the logic 0 or logic 1 bus.

BOOLEAN ALGEBRA 3.1

You have now seen how logic equations can be written using the symbols of Boolean algebra. This algebra, although confusing at first glance, is an extremely simple form of algebra and can be used for designing complex systems such as digital computers. We will now tabulate some important relationships and laws of Boolean algebra, many of which will be obvious and many of which may be easily derived.

Fundamental		Special Properties	
$\overline{0} = 1$			
$0+0=0$	$0.0=0$	$0+X=X$	$X+X=X$
$0+1=1$	$0.1=0$	$1+X=1$	$X+\overline{X}=X$
$1+0=1$	$1.0=0$	$0.X=0$	$X.X=X$
$1+1=1$	$1.1=1$	$1.X=X$	$X.\overline{X}=0$
$\overline{(\overline{X})}=X$			

Commutative law: $X+Y = Y+X$
$X.Y = Y.X$

Associative law $X(YZ) = (XY)Z$
$X+(Y+Z) = (X+Y)+Z$

Distributive law $X(Y+Z) = X.Y + X.Z$

De Morgan's Laws (i) "The complement of the sum of two functions is the product of their complements."

$$\overline{X+Y} = \overline{X}.\overline{Y} \qquad (3.1)$$

(ii) "The complement of the product of two functions is the sum of their complements."

$$\overline{X.Y} = \overline{X}+\overline{Y} \qquad (3.2)$$

Useful relationships $a+a.b=a$
$a+\overline{a}.b=a+b$

Many of these relationships have been mentioned previously in some form or another, although some will be new to you. You will find these relationships extremely useful in simplification of complex Boolean expressions.

VERIFICATION OF DE MORGAN'S LAWS 3.2

De Morgan's laws can be verified using the NOR and NAND gates available in Compukit 1. The first law, equation(2.1),requires that

$$\overline{a+b} = \overline{a}.\overline{b}$$

which can be implemented by using NOR and NAND gates in Compukit 1. The left hand side requires a NOR gate, and the right hand side can be implemented by first inverting a and b and then using a NAND gate as shown in fig.3.1

The second law, equation (3.2), is in fact a corollary of equation (3.1) and need not be proved separately. However, the second law can be verified by using a very similar circuit shown in fig.3.2 .

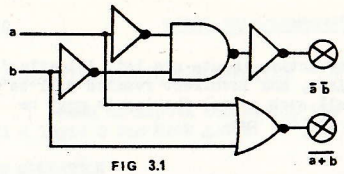


FIG 3.1

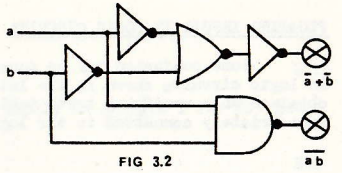


FIG 3.2

APPLICATION OF BOOLEAN ALGEBRA TO LOGIC SIMPLIFICATION 3.3

Now we shall see how Boolean algebra can be applied to logic design, in particular to the problem of simplification of logic circuits. This problem occurs very frequently and can best be explained by an example.

(i) Simplify the switching network shown in fig.3.3. This network uses four switching elements and can be expressed in terms of binary logic as follows:

$$T = (a+b)(a+c) \qquad (3.3)$$

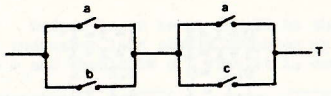


FIG 3.3

i.e. the output T is logic 1 if switches a or b and a or c are closed. Using Boolean algebra, we can show that this circuit is equivalent to,

$$T = a + bc \qquad (3.4)$$

Equation (3.4) can be implemented by the arrangement shown in fig.3.4, and uses fewer switches. It can also be implemented using basic gates as shown in fig.3.5.

Proof: that $(a+b)(a+c) = a+bc$

We have
$$\begin{aligned}(a+b)(a+c) &= aa + ac + ab + bc \\ &= a(1+c+b) + bc \\ &= a + bc \end{aligned} \qquad (3.5)$$

Equation (3.5) cannot be, however, easily implemeted using NOR, NAND and NOT gates in Compukit 1 without further manipulation. Again, we can use Boolean algebra for this purpose and convert the equation to a suitable form for its implementation by gates of the given type only.

De Morgan's Laws enable us to convert from a "sum form" to a "product form" i.e. from OR (or NOR) implementation to NAND (or AND) implementation. The circuit in fig.3.5, for example, can be implemented using NAND or NOR gates alone if necessary. The NAND gate implementation is obtained by further manipulation of equation (3.5) as follows,

$$T = a + bc = \overline{\bar{a}\,.\,\overline{b.c}} \text{ , using De Morgan's laws .}$$

This last form can be implemented using NAND gates as shown in fig.3.6.

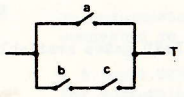


FIG 3.4

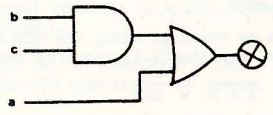


FIG 3.5

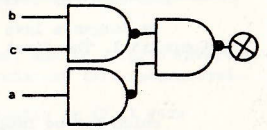


FIG 3.6

(ii) Let us now try to simplify a logic function given by,

$$T = (a+b)c + \bar{a}(\bar{b}+c) + \bar{b} \qquad (3.6)$$

This function can be represented by the switching network shown in fig.3.7

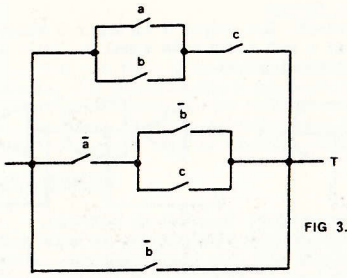


FIG 3.7

We can simplify eq.(3.6) as follows:

$$\begin{aligned} T &= ac + bc + \bar{a}\bar{b} + \bar{a}c + \bar{b} \\ &= (ac + \bar{a}c) + bc + (\bar{a}\bar{b} + \bar{b}) \\ &= c(a+\bar{a}) + bc + \bar{b}(\bar{a}+1) \\ &= c + bc + \bar{b} \\ &= c(1+b) + \bar{b} \\ &= c + \bar{b} \end{aligned} \qquad (3.7)$$

The expression in eq.(3.7) can be implemented by the switching network in fig.3.8, which is obviously a great simplification of fig.3.7. Without the help of Boolean algebra, it would be extremely difficult to arrive at the network in fig.3.8 from the network in fig.3.7. Equation (3.7) is however as yet not suitable for implementation using NAND gates alone. We apply the De Morgan's Laws to eq. (3.7) as follows,

$$T = c + \bar{b} = \overline{(\bar{c}.b)} \qquad (3.8)$$

Eq.(3.8) can now be implemented using NAND gates as shown in fig.3.9.

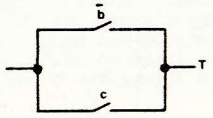


FIG 3.8

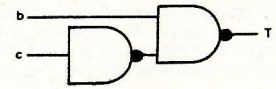


FIG 3.9

(iii) Simplify the following logic function,

$$\begin{aligned} T &= a + abc + \bar{a}bc + a\bar{b} + ad + a\bar{d} \\ &= a(1+bc) + \bar{a}b(c+1) + a(d+\bar{d}) \\ &= a + \bar{a}b + a \\ &= a + \bar{a}b \\ &= a + b \qquad (3.9) \\ &= \overline{\bar{a}.\bar{b}} \qquad (3.10) \end{aligned}$$

Equations (3.9) and (3.10) are the simplest forms to which we can reduce the original equation. Eq.(3.9) is suitable for implementation using NOR gates, and eq.(3.10) is suitable for implementation using NAND gates.

There are many other techniques for simplification of Boolean functions. The most important of these are the mapping methods due to Venn and Karnaugh, and the tabulation method developed by Quine and McCluskey. Techniques such as these can be found in more advanced texts on Boolean algebra and switching network theory, but are not necessary for beginners.

WIRED- OR LOGIC 4.1

Outputs of any two, or more, gates may be tied together to perform a Wired-OR . The maximum number of gates which may be tied together in this manner depends on the design of particular logic gates, and for the gates in Compukit 1 this is limited to four for NOR gates, and two for NAND gates. The output from the Wired-OR NAND gates must further be fed to a NOR gate only. With Wired-OR, the output is at logic 1 if all the outputs tied together are at logic 1, otherwise the output is logic 0. Fig.4.1 shows a Wired-OR implementation of the logic function,

$$T = \overline{ab + cd} \qquad (4.1)$$

Inputs a, b and c, d are applied to two 2 input NAND gates, the outputs of

which are tied together to perform the Wired-OR. The output T is logic 0 whenever either a and b are both equal to logic 1, or c and d are both equal to logic 1. Without the Wired-OR, we would require additional gates.

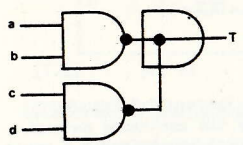


FIG 4.1

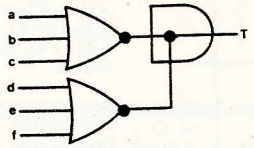


FIG 4.2

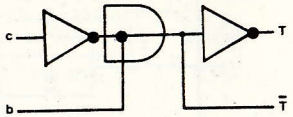


FIG 4.3

Fig.4.2 shows the Wired-OR method for implementing a 6 input NOR function using two 3 input NOR gates. In this case, the Wired-OR acts as an expander for the NOR gate. The logic function implemented in fig.4.2 is given by,

$$T = \overline{a+b+c+d+e+f} \qquad (4.2)$$

Any type of gate may be used, and the Wired-OR can also be achieved between NOR and NAND gates. The 'fan-out' from a Wired-OR output is considerably less than for ordinary gates. In certain logic designs, such as many types of integrated circuits, the Wired-OR is permitted on only certain types of gates. In this case, it is usual to have the collector resistances of all but one stage removed. Using this technique, you can Wired-OR upto eight gates in Compukit 1. However, it is not recommended for you to do this.

The Wired-OR approach simplifies the design and reduces the number of gates required. Thus, the circuit in fig.3.9 can be implemented using this approach as shown in fig.4.3 and requires only two gates to provide both T and $\bar{T}$. We will now look at a number of other logic functions implemented using this versatile technique.

COINCIDENCE CIRCUIT (Logic Comparator) 4.2

The circuit shown in fig.4.4 detects the coincidence of signals a and b. The output is at logic 1 if either both a and b are at logic 1, or both are at logic 0. This circuit can be used for comparing two binary bits.

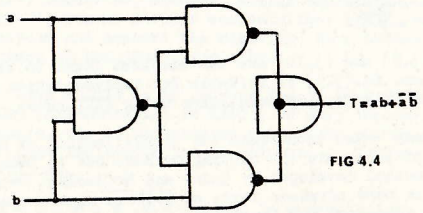


FIG 4.4

ANTI-COINCIDENCE CIRCUIT (Exclusive-OR) 4.3

In this circuit, the output T is at logic 1 if only one of the two inputs is at logic 1. If bits a and b are equal the output is logic 0. Using Boolean algebra, you can show that this circuit is a logical complement of the previous one. The exclusive-OR function is commonly used in a number of situations, and forms the basis of many useful circuits.

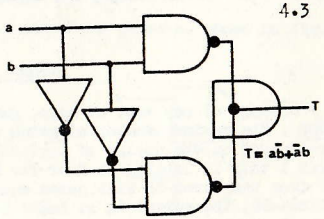


FIG 4.5

PARALLEL COMPARATOR 4.4

It is obvious that by using multiple coincidence circuits, we can compare binary words of as many bits as we like.

BINARY ARITHMETIC

Now you have already learnt many techniques commonly used in digital computers. The circuits which you have learnt can be used for 'decision making' and control functions in the computer. Some of them, for example the exclusive-OR, form the basis of other circuits used in binary arithmetic. We will now examine the basic approach used in digital computers to perform the arithmetic.

BINARY ADDITION 5.1

Addition is the most fundamental arithmetic operation, which can be in turn used to perform more complex operations such as subtraction, multiplication, division, exponentiation etc. The simplest of the adding circuits is a 'half-adder' shown in fig.5.1. This circuit is used to add two one-bit binary numbers. It has two inputs a and b, and there are two outputs which form the sum S and the carry C.

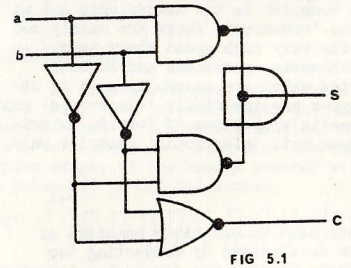


FIG 5.1

You can easily show that the following truth table applies to binary addition of bits a and b,

a	0	1	0	1
b	0	0	1	1
S	0	1	1	0
C	0	0	0	1

The logical equations can be written as,

$$S = \bar{a}b + a\bar{b} = \overline{\overline{a}\overline{b} + ab} \qquad (5.1)$$
$$C = ab = \overline{\bar{a}+\bar{b}} \qquad (5.2)$$

which can be implemented by using the exclusive-OR circuit, together with a 2 input NOR gate as shown in fig.5.1.

FULL ADDER 5.2

For binary addition of numbers containing more than one bit each, carries from the previous stages must be taken into account. The circuit used to perform this is known as the 'Full Adder'. A full adder has three inputs and two outputs. The inputs are bits a and b and the carry C from the previous stage. The outputs are the sum S and the carry D to the next stage. The truth table for a full adder is as follows,

a	0	0	0	1	1	0	1	1
b	0	0	1	0	1	1	0	1
C	0	1	0	0	0	1	1	1
S	0	1	1	1	0	0	0	1
D	0	0	0	0	1	1	1	1

The logical equations are given by,

$$S = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc$$
$$D = ab\bar{c} + \bar{a}bc + a\bar{b}c + abc$$

which can be simplified and regrouped as follows,

$$S = \overline{(\bar{a}b + a\bar{b})}C + (\overline{\bar{a}b + a\bar{b}})\bar{C}$$
$$D = ab + (\bar{a}b + a\bar{b})C$$

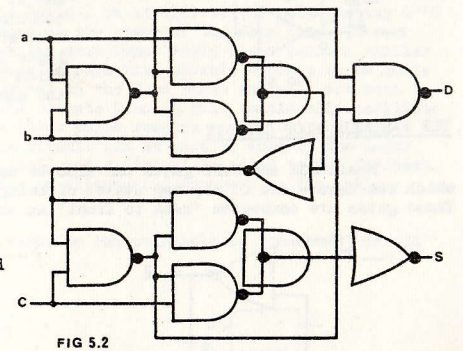


FIG 5.2

The sum S and carry D are now expressed in terms of the outputs of a half adder, and can be implemented using the alternative circuits shown in fig.5.2 .

TWO'S COMPLEMENT OF A 3 BIT NUMBER 5.3

Fig.5.3 shows a circuit for computing the two's complement of a three bit number using Wired-OR. The logic equations are,

$$A = a$$
$$B = \overline{(ab + \bar{a}\bar{b})}$$
$$C = \bar{c} + \bar{a}\bar{b}c$$

which can be implemented by the circuits shown in fig.5.3.

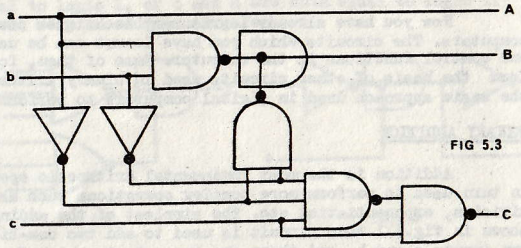


FIG 5.3

A very important part of the digital computer is the memory unit and we shall now look at the basic circuits which can 'remember'. There are mainly two types of memories used in modern computers. The very high speed short term memories such as those used in performing arithmetic operations and control functions, and the medium to long term memories which are generally used in the storage of data and programmes. The former types are invariably 'electronic' units, whilst the latter depend upon the electromagnetic properties of ferrite materials for their operation. Here we shall be concerned with 'electronic' memories only.

A SIMPLE MEMORY 6.1

The simplest form of electronic memory used in computers consists of 'latching' circuits. One such circuit can be constructed by connecting two logic inverters as shown in fig.6.1. The outputs Q and $\bar{Q}$ are always complementary to each other, and the 'memory' can be set by applying a logic 0 through the 'set' and 'reset' push buttons as shown. When the logic 0 is applied through the set terminal, Q goes to 1 and $\bar{Q}$ goes to 0. The reset push button makes Q equal to 0 and $\bar{Q}$ equal to 1 again when depressed.

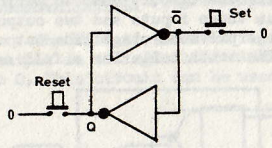


FIG 6.1

Using gates in Compukit 1, you can build a 'seven bit' memory using this principle. The push buttons are not really necessary and logic 0 can be applied to set or reset the memory by momentarily touching the appropriate terminal by a wire whose other end is connected to the logic 0 bus. These circuits will remember which button was last depressed, hence the name 'memory'.

THE R-S FLIP FLOP CIRCUIT 6.2

Basic NOR and NAND gates can also be used to form 'latching' circuits which can assume one of the two states of being 'set' or 'reset' as required. These gates are connected 'back to front' as shown in figs.6.2 and 6.3.

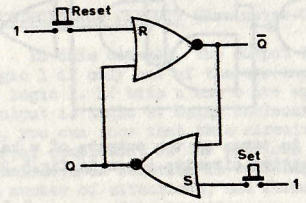


FIG 6.2

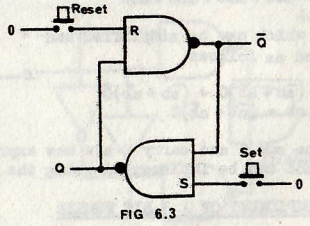


FIG 6.3

In fig.6.2, two 2 input NOR gates are connected to form a 1-set R-S Flip Flop. The two inputs, S and R are the set and reset inputs and there are two outputs Q and $\bar{Q}$ as before. Operation of this circuit is very similar to the previous one, and it 'remembers' which button was last depressed. This type of memory is, in fact, the simplest type used in digital computers.

The logical equations for an R-S flip flop can be written thus,

R	1	0	0	1
S	0	1	0	1
Q	1	0	$\bar{Q}$	0
$\bar{Q}$	0	1	Q	0

R-S Flip flop using two NOR gates

R	1	0	1	0
S	0	1	1	0
Q	1	0	$\bar{Q}$	1
$\bar{Q}$	0	1	Q	1

R-S Flip flop using two NAND gates

You will notice that in the last column of both these tables, Q and $\bar{Q}$ have been listed as having the same logical states. This is, obviously, contrary to our definition and is consequently not allowed. Thus, for the flip flop in fig.6.2 the R and S inputs should never be allowed to have logical states 1 at the same time, and for the flip flop in fig.6.3 they should never be at logic 0 simultaneously. This restriction severely limits the application of R-S flip flop and to overcome this difficulty a 'clocked' flip flip known as the J-K type has been developed.

BINARY COUNTING 6.3

In binary counting, the first digit has only to change alternately from 1 to 0 and 0 to 1 on each count. This will be obvious to you by looking at the binary numbers in a previous section. The second stage, also, changes alternately from 1 to 0 and 0 to 1 each time the first stage changes from 1 to 0, and so on. Various stages of the binary counter will assume states as shown in fig.6.4 as the pulses to be counted arrive.

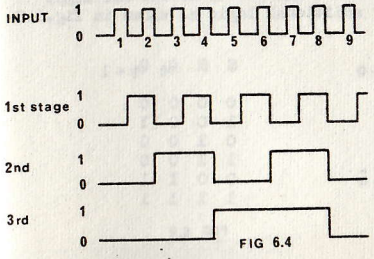


FIG 6.4

It is quite clear that the subsequent stages function in exactly the same manner as the first stage. The first stage can be easily designed if the pulses to be counted could be alternately routed to the R and S inputs of the flip flop shown in fig.6.2. Consider the circuit shown in fig.6.5. The concept behind this circuit is that if an input is not present, the points R and S will both be at logic 0 and Q and $\bar{Q}$ will remain in the last state in which they had been set. Thus, if Q and $\bar{Q}$ were at logic 1 and 0 respectively before the input pulse arrived, R would go to logic 1 on arrival of the pulse making Q = 0 and $\bar{Q}$ = 1. The input pulse has thus changed the state of the flip flop. We can pursue a similar argument to show that the next input would again cause a similar change making it a binary counting stage. Though this circuit appears to be quite feasible at first sight, it is impractical in the form shown and will not work ! A little thought will show you that the outputs Q and $\bar{Q}$ will continually oscillate from 1 to 0 and 0 to 1 as long as the input pulse remains at logic 1, making it thus useless as a binary counter. This circuit can be made to work if the input pulse duration is made sufficiently small and a 'delay' is added to the feedback loops in fig.6.5. We shall now consider an alternative approach to the design of a binary counter.

The circuit shown in fig.6.7a works on the principle of 'remember and add'

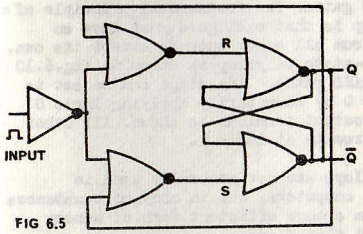


FIG 6.5

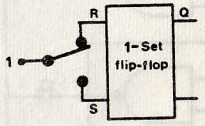


FIG 6.6

and can be used as a binary counter. When the input pulse is at logic 0, the flip flops A and B 'copy' the logic states of $\bar{Q}$ and Q respectively, and remain in this state till the input pulse changes again. When the input pulse goes to logic 1, outputs Q and $\bar{Q}$ are then set to the same logic states as A and B. Thus the

sequence of operations can be illustrated as in fig.6.7b. Due to the 'cross-over' in the information exchange loop, the logic states of Q and $\bar{Q}$ are changed each time an input pulse is received.

For testing the binary counter, it is necessary to generate a 'clean' pulse. If a change-over switch is used to control an R-S flip flop, fig.6.6, the output from the latter is free from 'contact-bounce' thus producing a clean pulse.

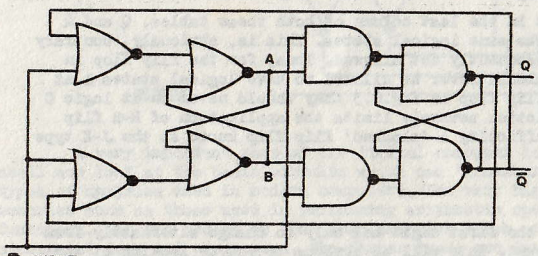


FIG 6.7a

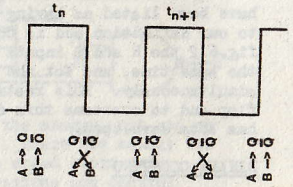


FIG 6.7b

A MODIFIED R-S FLIP FLOP

6.4

The R-S flip flop shown in fig.6.2 can be modified to accept two logic 1's at the input terminals simultaneously by additional logic as shown in fig.6.8.

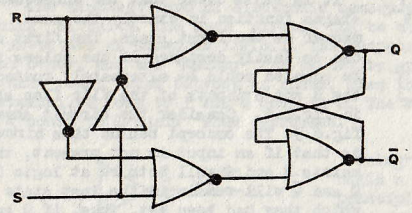


FIG 6.8

S	R	Q_t	Q_{t+1}
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	0
0	0	1	1
1	1	1	1

FIG 6.9

The corresponding truth table is shown in fig.6.9. This circuit is somewhat similar in operation to the J-K flip flop mentioned before, except that it is not 'clocked'. For description of the J-K type, you are asked to refer to more advanced texts.

POLYFLOP

6.5

A polyflop is a logic circuit that has three, or more, states and will 'remember' the last of the several states to which it had been set. A flip flop is a special case of a polyflop, where the number of possible states has been reduced to only two.

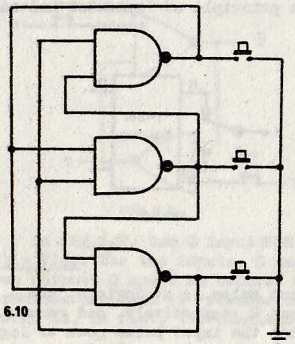


FIG 6.10

Poly flops can be constructed using NOR or NAND gates. The fundamental principle of a polyflop is that each gate must have an input from all other outputs except its own. A three-state polyflop is shown in fig.6.10 using NAND gates. Any stage can be set to a logic 0 by momentarily applying logic 0 to the output terminal as shown. All other stages remain at logic 1.

Polyflops are not generally used in digital computers, but in many circumstances can form a more efficient form of memory than R-S flip flops.

Addendum

NOTES ON COMPUKIT 1 DELUXE MODEL AND HOW TO USE IT

Compukit 1 Deluxe Model is a teaching aid for computer electronics, digital logic and Boolean algebra and is similar to the standard model of Compukit 1 in many respects. It is however a more sophisticated teaching aid and is mainly intended for use in schools, technical colleges and industrial training organisations.

The deluxe model is housed in a cabinet with an on/off switch on one side. The cabinet also carries the battery, making it an exceptionally easy to use portable instrument.

This model has a five switch 'Input Register' on the top left hand side which is used to provide five independent logic inputs to the remaining system. These logic inputs appear on the two terminal pins immediately to the left of the particular switch. A 'logic 0' is obtained when the switch is up and a 'logic 1' is obtained when it is down.

The 'Output Register' on the top right hand side consists of four miniature logic indicator lamps, which brighten up when a logic 1 input is applied to the terminal pins immediately under the bulbs.

The power is switched on by pulling the on/off switch lever towards you. The filaments of the bulbs in the output register will now glow slightly. The power source used in this model is the 4½ volt battery, Ever Ready Type 126, or equivalent, and can be replaced by removing the back of the cabinet, if necessary. If an external power source is used, it should provide filtered direct current at no greater than 6 volts. The current required by the deluxe model is about 0.3 amperes. It is advisable that the power supply should be short circuit proof in order to avoid accidental damage.

The electronics in Compukit 1 Deluxe Model is 'student proof' and will not normally get damaged due to accidental incorrect patching, provided that the recommended 4½ volt battery, or equivalent, is used.

The logic on the deluxe model consists of two 3-input Nor gates, six 2-input Nor gates, two 3-input Nand gates and six 2-input Nand gates. The logic used is 'positive logic' with logic 1 equal to 4 volts nominal and logic 0 equal to 0 volts nominal. The maximum fan out is about ten. For simulating larger systems, two or more units may be connected together by patching the battery terminal pins.

All circuits given in the instruction book for the standard model of Compukit 1 can also be made on the deluxe model, and many more circuits can of course be constructed by the imaginative student and teacher. Use of associated accessories, such as the task sheets and transparencies for overhead projection, is strongly recommended for classrooms and teaching laboratories.

ASSEMBLY PROCEDURE FOR SOLDERLESS PATCH LEADS WITH HEAT SHRINKABLE INSULATION SUPPLIED IN KIT FORM (Compukit 1 standard model only):

The patch lead kit contains about 24 feet of flexible wire, 48 silver plated socket ends and small lengths of heat shrinkable plastic sleeving in assorted colours. The assembly procedure is as follows:

1. Cut the wire in equal lengths of about 10-12 inches as required and strip about 3/8" insulation from each end. Tin the stripped ends using a little solder and solder the metal sockets to each end. The terminal pins on an assembled kit can be used to hold the sockets during soldering.
2. Cut the plastic sleeving in about ¾" lengths and shrink it over the metal socket ends to grip tightly by heating for a few seconds using hot air, or by holding near a low flame.

The patch leads are now ready to be used to make solderless connections.