Control Instructions

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
IDLE	IDL	00	WAIT FOR DMA OR INTERRUPT; M(R(0))→BUS
NO OPERATION	NOP	C4	CONTINUE
SET P	SEP	DN	N→P
SET X	SEX	EN	N→X
SET Q	SEQ	7B	1→Q
RESET Q	REQ	7A	0→Q
SAVE	SAV	78	T→M(R(X))
PUSH X,P TO STACK	MARK	79	(X,P)→T; (X,P)→M(R(2)) THEN P→X; R(2)−1
RETURN	RET	70	M(R(X))→(X,P); R(X) +1 1→IE
DISABLE	DIS	71	M(R(X))→(X,P); R(X) +1 0→IE

Input–Output Byte Transfer

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
OUTPUT 1	OUT 1	61	M(R(X))→BUS; R(X) +1; N LINES = 1
OUTPUT 2	OUT 2	62	M(R(X))→BUS; R(X) +1; N LINES = 2
OUTPUT 3	OUT 3	63	M(R(X))→BUS; R(X) +1; N LINES = 3
OUTPUT 4	OUT 4	64	M(R(X))→BUS; R(X) +1; N LINES = 4
OUTPUT 5	CUT 5	65	M(R(X))→BUS; R(X) +1; N LINES = 5
OUTPUT 6	OUT 6	66	M(R(X))→BUS; R(X) +1; N LINES = 6
OUTPUT 7	OUT 7	67	M(R(X))→BUS; R(X) +1; N LINES = 7
INPUT 1	INP 1	69	BUS→M(R(X)); BUS→D; N LINES = 1
INPUT 2	INP 2	6A	BUS→M(R(X)); BUS→D; N LINES = 2
INPUT 3	INP 3	6B	BUS→M(R(X)); BUS→D; N LINES = 3
INPUT 4	INP 4	6C	BUS→M(R(X)); BUS→D; N LINES = 4
INPUT 5	INP 5	6D	BUS→M(R(X)); BUS→D; N LINES = 5
INPUT 6	INP 6	6E	BUS→M(R(X)); BUS→D; N LINES = 6
INPUT 7	INP 7	6F	BUS→M(R(X)); BUS→D; N LINES = 7

♦NOTE: THIS INSTRUCTION IS ASSOCIATED WITH MORE THAN ONE MNEMONIC. EACH MNEMONIC IS INDIVIDUALLY LISTED.
♦♦NOTE: THE ARITHMETIC OPERATIONS AND THE SHIFT INSTRUCTIONS ARE THE ONLY INSTRUCTIONS THAT CAN ALTER THE DF

THE EDUKIT MANUAL

by Dr. A.A.BERK

MODUS SYSTEMS Ltd
29A Eastcheap,
Letchworth,
Herts., SG6 3DA

Control Instructions

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
IDLE	IDL	00	WAIT FOR DMA OR INTERRUPT; M(R(0))→BUS
NO OPERATION	NOP	C4	CONTINUE
SET P	SEP	DN	N→P
SET X	SEX	EN	N→X
SET Q	SEQ	7B	1→Q
RESET Q	REQ	7A	0→Q
SAVE	SAV	78	T→M(R(X))
PUSH X,P TO STACK	MARK	79	(X,P)→T; (X,P)→M(R(2)) THEN P→X; R(2)−1
RETURN	RET	70	M(R(X))→(X,P); R(X) +1 1→IE
DISABLE	DIS	71	M(R(X))→(X,P); R(X) +1 0→IE

Input–Output Byte Transfer

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
OUTPUT 1	OUT 1	61	M(R(X))→BUS; R(X) +1; N LINES = 1
OUTPUT 2	OUT 2	62	M(R(X))→BUS; R(X) +1; N LINES = 2
OUTPUT 3	OUT 3	63	M(R(X))→BUS; R(X) +1; N LINES = 3
OUTPUT 4	OUT 4	64	M(R(X))→BUS; R(X) +1; N LINES = 4
OUTPUT 5	CUT 5	65	M(R(X))→BUS; R(X) +1; N LINES = 5
OUTPUT 6	OUT 6	66	M(R(X))→BUS; R(X) +1; N LINES = 6
OUTPUT 7	OUT 7	67	M(R(X))→BUS; R(X) +1; N LINES = 7
INPUT 1	INP 1	69	BUS→M(R(X)); BUS→D; N LINES = 1
INPUT 2	INP 2	6A	BUS→M(R(X)); BUS→D; N LINES = 2
INPUT 3	INP 3	6B	BUS→M(R(X)); BUS→D; N LINES = 3
INPUT 4	INP 4	6C	BUS→M(R(X)); BUS→D; N LINES = 4
INPUT 5	INP 5	6D	BUS→M(R(X)); BUS→D; N LINES = 5
INPUT 6	INP 6	6E	BUS→M(R(X)); BUS→D; N LINES = 6
INPUT 7	INP 7	6F	BUS→M(R(X)); BUS→D; N LINES = 7

♦NOTE: THIS INSTRUCTION IS ASSOCIATED WITH MORE THAN ONE MNEMONIC. EACH MNEMONIC IS INDIVIDUALLY LISTED.
♦♦NOTE: THE ARITHMETIC OPERATIONS AND THE SHIFT INSTRUCTIONS ARE THE ONLY INSTRUCTIONS THAT CAN ALTER THE DF

FOREWORD & IMPORTANT NOTES

This manual provides the beginner with a "once-over-lightly" account of the major topics of micro-computing. This field includes some of the most interesting and up-to the-moment ideas provided by modern technology. It is impossible in a short handbook of this type to cover all the material necessary for a complete understanding. Instead, an attempt is made to guide the owner through the subject in a practical manner by actual experience with a real machine. The main advantage, therefore, of the EDUKIT over a more complete and academic treatment of microcomputing, is its accent on "hands-on experience". The reader need no longer study abstract texts on the subject without being able to try the ideas out - instead, each stage of the learning process is accompanied by practice.

The method chosen to communicate ideas is one of example rather than lengthy explanation. The latter is left for the more complete and academic books which any EDUKIT user should consult as soon as possible. Indeed, the kit and manual must be viewed as a "primer" to allow more complex treatises to be understood if the user's interest is sufficiently aroused. If not, nothing is lost.

It is hoped, however, that the EDUKIT will st imulate an even greater curiosity and allow this interest to flourish towards greater things.

We wish you all the very best of luck!

Dr. A.A. Berk
MODUS SYSTEMS Ltd.,
29A Eastcheap,
Letchworth,
Herts.

NOTES

1. The kit includes a socket for the Microprocessor only - if you are very inexperienced or wish to expand the unit further, a set of sockets for the other chips is most desirable. A set of sockets is available from MODUS for £2.60 + VAT.

2. Ready assembled and tested units are available from MODUS - £39.95 + VAT + 80p PP.

3. MODUS is a recognised, Government recommended Software and Hardware Consultant house, and will be most happy to quote for your control and/or software problems in general.

4. Technical backup is available for our products, and if the kit will not work after assembly, MODUS will undertake to repair on a time-cost basis - minimum cost £10.

5. The 1802 User's Manual, and Tom Pittman's "Short Course" (see Appendix I) are available from MODUS - the Manual costs £3.95 + 50p PP - contact MODUS for the price of the "Short Course".

Sincerest thanks are due to Janet Morgan and Anne Noon for their extensive and valuable assistance with the production of this manual.

s (¼ Watt 5%)

1K (17 off)
2K2 (5 off)
100 (3 off)
10K (2 off)
18K

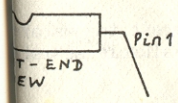
ransistors, LED's

N4001-4
N4148 (Typically) (7 off)
ed Led's (3 off)
N2926 (typically)
ND 500 (2 off)

ted and described below.

cleaning before
y and must not

egrated circuits (IC's)
es are of the "dual in
on each side of the



s. In order to
irst read the device
bination of numbers
ous other letters
d an LS in the middle.

Next, note that in addition to various dents in the upper surface, one end ha notch cut into its upper surface as shown above. This notch is used to ensure that package is inserted in position the correct way round. Failure to observe the correc orientation will totally destroy the IC's. The notch must tie up with the component overlay diagram.

Sometimes, instead of, or in addition to, the notch, a small hole or bump will appear on the upper surface of the IC to identify Pin 1. The pins are numbered, on all chips, anticlockwise around the chip, as shown, when looking at the top surface. The largest IC, the Microprocessor itself, is a 40-pin device, and a socket is included, for it, in the kit. Do not remove the IC from its packing until absolutely necessary.

CAPACITORS

There are just two types of capacitor :

a) Electrolytic : C 1, C 2, C 3 - all identical.

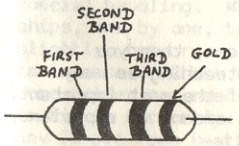
These devices have two pins - one of them is "+" and one "-", as shown on the body of the device. The + side is marked on the PCB so that these capacitors can be inserted the correct way round.

b) Ceramic : C 4, C 5, C 6, C 7.

These are flat devices with two pins and can be inserted any way round - C4, C5, C6 are marked "10n" and C7 is marked "100n" - perhaps with some other letters, depending upon the type supplied.

RESISTORS

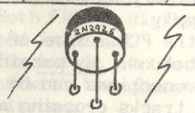
All resistors have red bodies and are similar, apart from coloured bands which identify the value of the resistor according to the following table:



Resistor	Value	Bands		
		First	Second	Third
R1-R17	1K	BROWN	BLACK	RED
R18-R22	2K2	RED	RED	RED
R23-R25	100	BROWN	BLACK	BROWN
R26, R27	10K	BROWN	BLACK	ORANGE
R28	18K	BROWN	GREY	ORANGE

Resistors may be inserted any way round.

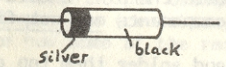
DIODES, TRANSISTORS, LED's.



The transistor (TR1) has 3 leads and 2N2926 written on it. It must be inserted, as shown here, with correct pins in correct holes.

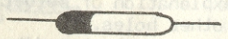
There are three types of diode. They must all be inserted the correct way around.

a) IN4001 (D1) has a black body and a silver ring around one end.

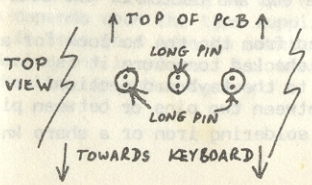


The ring end must tie up with the black stripe on the Component Overlay.

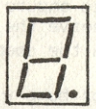
b) Small signal diodes: D2 - D8. Again, one end is striped - with Black or yellow - and must tie up with the striped end shown on the Component Overlay.



c) Light-Emitting Diodes (LED's) LD1, LD2, LD3.



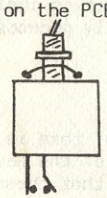
These are translucent red objects with two pins - one longer than the other. The long pins must be inserted in the holes as shown in the diagram, with PCB oriented accordingly.



If you inspect the upper surface of these devices the number 8 can be seen, with a decimal point showing. The devices must be inserted into the PCB with this point in the position shown on the Component Overlay.

ON/OFF SWITCH

The switch should be positioned the correct way round and soldered to two pins left upstanding on the PCB.



Disregard the words "On" and "Off" on the switch, and bend the pin as shown, to fit them in neatly. A piece of wire left over from the resistors can be stretched over the threaded portion of the switch and soldered to the pads below the PCB. The nut on the switch should be left in place to raise the switch body off the PCB and prevent the metal portion from shorting PCB tracks together.

CONSTRUCTION

You will need the following items:

a) A pair of side-cutters, capable of cutting the small pins supplied.
b) A watchmaker's glass or magnifying glass.
c) A soldering iron of around 15 watts power and fitted with a 3/32" bit.
d) Some flux-cored solder of thickness about equal to that of the wire leads on the resistors supplied. (22 s.w.g.)
e) A bottle of methylated spirit and a household paint brush.

To attempt assemby without these tools, or with tools of a different specification, will certainly court disaster.

The experienced contructor will not need to read the following in very great detail, but everyone should glance through at least once.

BEGINNERS START HERE

If you are not adept at soldering and construcing electronic devices, then you should read the following in very great detail. This kit is aimed at teaching as many aspects of the "new technology" as possible, and construction is one of the most important. In addition, by following the steps carefully, you will pick up a vast amount of experience and jargon concerning electronic components.

DESCRIPTION

The set of components in the kit will be assembled onto a single "board" and it is this board which is at the centre of the design.

Examine the fibre-glass board carefully. This is called a Printed Circuit Board (PCB). It has "tracks" on both sides, composed of "tinned copper" which conducts electricity. These tracks connect the components of the EDUKIT together, as well as conducting power throughout the system from the battery or power supply.

You will notice that there is a very large number of holes in the PCB. Some of them pass through tracks on the top of the board and tracks on the bottom. By passing metal pins through these holes, a complex pattern of interweaving connections can be made from top to bottom on the two-dimensional surface of the PCB without tracks crossing each other and shorting out. Some of these pin-through connections occur in the middle of wide PCB tracks, the rest occur as square "pads" with holes through the centre. You should inspect the board and check you can identify places where the pins should go through - if you put too many pins through the board, other components may not fit in later.

SOLDERING

Your first task, then, is to push pins through the PCB and solder them top and bottom. If you have never soldered before, this task is excellent for learning on before moving on to the more delicate components. For an explanation of soldering, see AppendixII . Do not, at this stage, pass pins through any other holes.

Having soldered the pins in, they should be clipped close to the board to neaten the appearance and prevent them fouling on other components. The final operation, before continuing, is to clean all the solder flux from the top and bottom of the board using a household paint brush and plenty of meths.

Having done this, the PCB should be carefully inspected from the top to look for any through-connections left unmade. Then, each pin should be checked to ensure it is soldered top and bottom. Do not forget the four pins next to the keyboard section! Check, with a glass, that there are no bridges of solder between two pins or between pins and nearby tracks. If any are found, remove them with the soldering iron or a sharp knife.

KEYBOARD

The switches come apart and are easily reassembled if necessary. The body of the switch is composed of a thermoplastic which easily melts at soldering temperatures and extreme care should be exercised when soldering them in place. If too much heat is applied the pins move into the body of the device and prevent correct operation. Locate the 20 switches in their positions on the board, place a flat piece of cardboard over the tops and invert the PCB, holding the board against the switches to prevent them from falling out Rest the assembly on the bench and take steps to ensure that all the switches are press ed firmly against the PCB. To do this, build up a pile of paper, for instance, under the top half of the PCB to the same thickness as the switches and place a heavy object in the middle of the PCB. This will level the PCB up with respect to the bench and push the switches firmly against the board.

Soldering of the switches should be done swiftly and carefully and without applying an end-on force to the pins or they will recede slightly into the plastic bodies of the switches. The pins can be cleaned very carefully beforehand but should not be tinned until in position - try not to touch the pins as any dirt or finger-grease impedes soldering and hence elongates the process and heats up the pins unnecessarily.

Care with this procedure produces a keyboard giving many hours of satsifactory use. Try the keys after soldering, each one should "click" under an even pressure to signify that contact has been made. If not, take the top off the switch, remove the metal disc inside, and check that none of the 3 metal pads inside has been pushed , even slightly, up into the plastic body. If one is slightly raised, apply the soldering iron to the pin beneath the board and gently push the metal back into place from the top pad itself. Any key not making proper contact should be dismantled and cleaned carefully.

Integrated Circuits, sockets, digital display

Identify each integrated circuit and, except for the 1802, place it (the right way round) on the Component overlay. These chips are very robust and do not require special handling. When you are satisfied that everything is quite correct, transfer the chips, one by one, to the PCB and push the pins through the holes. They may need bending slightly, but once a chip is in place it will usually remain there even when the board is inverted. Check that each IC is pressed firmly against the PCB's upper surface - some solder may need to be removed from one or two pin-through joints to allow all the IC's to fit in correctly. The IC's should then be soldered with care. Work quickly to prevent any IC pin from heating up too greatly. Use the minimum amount of solder to prevent solder-bridges from occurring between nearby pins and tracks. Examine the 40-pin socket and you will discover that one of the top "inside" corners is filled-in with a wedge to identify the position of Pin 1 - it will help you to insert the 1802 correctly if you line this wedge up as shown on the Component Overlay. The socket pins will not stand being bent unecessarily and overheating will melt the plastic body - solder this component carefully while pressing it firmly against the PCB. Do not insert the 1802 yet.

Both FND500 digital displays should also be pressed firmly against the PCB and soldered in place.

The bottom of the board should now be cleaned throughly with meths again and inspected with the glass for solder bridges. Take care not to allow the meths near the key switches.

The operation of cleaning and inspecting takes a minute or two but is one of the best investments of effort possible during the assembly.
Often, a possible solder bridge is washed away along with the flux.

Discrete Components (resistors, capacitors etc.)

Notice how some of the resistors are stood on end while others lie flat.

Push their leads through the board without straining the body of the Compoent, solder in place and clip the excess lead close to the board.

The same goes for the other components. Be careful not to force the capacitors and transistor too close to the PCB or the leads will be forced away from the device.
The LED's may fit close to the PCB or perhaps suspended slightly above it, on their leads. This depends upon the type supplied.

It is a good idea to identify all the Components with correct orientations before soldering any in place. All these Components are physically delicate and will not stand overheating or undue force.

The final soldering operations are for some extra pins and the ON/OFF Switch. Pins should be inserted, from the top, for the switch leads to solder too. These pins should be left standing up on the top of the board, but may be clipped beneath the board after soldering. Pins must them be inserted for 0 volt, + 5 volt and + 6 volt power supplies and soldered and clipped beneath the PCB only.

The only holes remaining unfilled are eight holes down the sides of the 1802 which are used for future expansions and should remain unfilled until needed. Any other holes are mistakes and should be checked carefully.

Keyboard Legends

The operation is to cut the card supplied into horizontal strips of keyboard legends and stick them in place as shown on the diagram. "Blue-Tack" or some similar material is ideal for this purpose as the key switches must be easily accessable for disassembly and cleaning. Use the minimum amount of adhesive possible to prevent it from being caught up in the switches' movement.

A spare set of legends is included on the card-just in case!

Inserting the 1802

This component may be damaged by excess static electricity. It is packed with pins shorted together in Conductive foam, aluminium foil or special antistatic tubes. You should Earth yourself by touching a cold water pipe while you handle this device. The pins will need bending before insertion into the socket. As you insert the pins, check that none are bent under the IC. For safety, some time should be spent on this operation.

When finished, a final check must be made with the glass for solder bridges - the bane of even the most experienced constructor!

SOME MICROCOMPUTER THEORY

A computer of any kind is, primarily, a machine cpable of executing a logical set of instructions. These instructions are stored in the machine's memory by the user and the machine is then forced to execute these commands. This list of commands is called a computer program. On the EDUKIT, programs are entered through the Keyboard in a special code which will be described later.

MICROPROCESSOR UNIT (MPU)

The heart of any computer is its Central Processing Unit (CPU). In a microcomputer this device is called a "Microprocessor Unit" (MPU) - the MPU is physically smaller than the CPU of a large computer installation. The CPU or MPU is wired in, electrically, to receive the program commands from memory, one by one, to interpret and execute them, while controlling all the devices in the system simultaneously.

MEMORY

The programs are stored in electronic components referred to as memory devices - the 2111 chips on the EDUKIT perform this function and the machine automatically stores the information input from the keyboard in these devices. The EDUKIT has 256 separate memory locations - each with its own electrical label or "ADDRESS".

COMMUNICATION

A set of microcomputer devices, no matter how sophisticated, is useless unless communication is allowed between members of the set. The block diagram (Chap5) shows the Communication path of the EDUKIT. The MPU "sits" at one end of a set of BUSES. There are so many interconnecting wires involved that they are "bundled" together, for convenience, into the collections of "Buses" of wires devoted to similar functions. The DATA bus (8 lines) is used by the MPU to fetch information from, and store information in the memory itself. These functions cannot be performed simultaneously, and it is up to the MPU to control the activities through the Control Bus.

When an electronic device of any kind is asked to perform an activity, the time it takes to react depends upon its nature - mechanical devices are slow, integrated circuits are very fast. Different circuits react at different speeds and hence it is essential to regulate the MPU's commands to its subordinates in a fashion which is slow enough for its slowest components but fast enough to use a computer rather than "do it by hand". To enable the MPU to act sequentially at the right speed a "clock" is always included in the system design. This clock gives out pulses used by the MPU to time its actions carefully.

Thus, if a program (set of instructions) is stored in memory, each instruction must be fetched and executed, in the correct time frame, before the next is fetched. To ensure that the right piece of data is fetched along the data bus, each piece of data is stored in a separate memory location in the memory block. To select the correct location uniquely, each memory location within the block has an address which sets it apart from all others. The address of a memory location is very similar to that of one's house. A letter is sent to your home by placing your address on the envelope. The address is recognised by the postal system - County first, Town next, road next and finally, the number. The letter arrives at its destination via a complicated set of routes.

In the same way, when the MPU wishes to communicate with a given memory location, it writes the appropriate locations onto the Address Bus lines and the memory block reads the Address Bus and switches the Data Bus to that location uniquely. It is up to the MPU, via the Control Bus, to say whether it wishes to STORE data contents at that location , or READ data onto the Data Bus from memory. If the latter, the MPU expects information back along the Data Bus.

The MPU, therefore, controls the system via the Buses using some built-in intelligence.

DATA AND ADDRESS NOTATION

So far, no system has been described for the format of either the Data or Address information which "flies" around the buses. A quick glance at the Circuit Diagram in the Hardware section of this manual will reveal a set of wires labelled D 0 - D 7 and another A 0 - A7. These are Data Bus and Address Bus respectively. A glance at the circuit diagram also emphasises the need for a block diagram to unravel the workings of a system!

Each line in a bus may carry only one piece or "BIT" of information at a time and in a typical Bus - 8 bits are sent simultaneously; larger Buses carry even more for each "tick" of the MPU's clock. A further restriction is placed on the BIT for a number of fundamental reasons. It may assume one of only two states: 0 volts or +5 volts. These states are labelled "0" and "1". A line is said to be in the "1" state, for instance, if a voltage measuring device on that line would read + 5 volts (in fact, less voltage would suffice - but that discussion belongs elsewhere). If 0 volts is read, the line is carrying a "0". The measuring device must be very fast, however, as Buses normally change state at speeds of more than one million times per second - quite a respectable

speed by Human standards, but relatively slow for most simple electronic logic circuits on the market today!

Thus, at any time instant, the Bus lines will contain a "pattern" of eight bits which, as far as the MPU is concerned, is one piece of data (or Address) - normally referred to as one BYTE.

Thus :

1 BIT = 0 or 1
8 BITS = 1 BYTE

Since any line has just two possible states, the number 2 is of fundamental importance in computing, just as the number 10 is in everyday affairs. We say that the base of our "normal" numbering system is 10. If a number is allowed to have two digits, it may represent any one of 100 different values - i.e. nos. 0, 1, 2,.....10, 11,.....,99. Notice how 0 is included to produce 100 numbers. i.e. 2 digits allow 10^2 (or 10 x 10) numbers. Similarly, 3 digits allow 10^3 = 1000 different values : 0→999.

A binary digit (BIT), however, is only allowed to take on one of 2 values. Thus a two-BIT number may only have value = 00, 01, 10 or 11 - i.e. 4 = 2^2 different values. Similarly, 3 bits give 2^3 = 8 values. Thus, 8 lines, each having just one of 2 values (0 & 1) has 2^8 = 256 different possible patterns of 1's & 0's - remember this number, it is very important! In binary, therefore, we say we are counting to BASE 2.

Normally, a digit communicated by humans may have one of the states 0,1,2,3,4,5,6, 7,8,9 - notice again how the system starts at 0 and not at 1. More complex numbers are built up as follows :

number 387 = 7 ones
+8 tens
+3 hundreds

and everything is 'based' on powers of ten.

I.e. one hundred = 10^2 (or 10 x 10)
one ten = 10^1
one = 10^0

The latter may be unfamiliar and is a mathematical law which must be learnt. Thus, 387 may be shown on a diagram as follows:-

10^2	10^1	10^0
3	8	7

where the lower row of numbers is the number of 10^n units to be included in the final number - this table, therefore, implies that 387 contains three "ten squareds", eight "ten to one's" and seven "ten to zero's".

Similarily, 1594 contains:

10^3	10^2	10^1	10^0
1	5	9	4

one thousand, five hundreds, nine tens and four ones.

This system of describing numbers is called the "decimal system" as it is based on ten. The system originated, presumably, because of the set of ten fingers and thumbs we all carry, and use, even today, as primitive counting devices. Luckily, early man did not include his toes or we should have twenty digits in our counting system! Electronic computers, however, were notably absent from prehistoric man's environment, or we could now be using the BINARY COUNTING SYSTEM. Any large number is built up from 1 and 0 in precisely the same way as that described above.

Consider a Bus of 8 lines, D7, D6, D5, D4, D3, D2, D1 and D0, containing the following pattern of 1's and 0's: 10010111 - how are we to interpret this number in our everyday system? The answer is found by drawing a 2's table as we did above for ten:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	1	0	1	1	1

(here, as before, 2^0 = 1)

The trick is to build up the final interpretation of the number as we did for 387 above. The lower numbers denote the number of 2^n's found in the final value. I.e., the table says to add up the following:

$1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

which equals:

128 + 0 + 0 + 16 + 0 + 4 + 2 + 1,

Which is 151 in our normal base-ten language. We indicate this by the notation:

$(10010111)_2 = (151)_{10}$

Forgetting this notation can give rise to confusions : e.g. 101 could be binary, and equal $1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$ = five (in decimal), or (in decimal) equal to one hundred and one.

The data on the Data Bus or Address Bus is not always to be interpreted simply as a number - sometimes it is a numerical code for an instruction to be executed by the MPU - sometimes a series of 1's and 0's to switch a bank of lights on or off; but always the pattern of 1's and 0's can be written down in decimal numerical form, as above, more conveniently than in the original binary. I.e., 151 is a more convenient notation for the pattern on the Data Bus than 10010111. The problem is that the latter is far more descriptive of what is actually happening on each Bus line - the decimal equivalent is rather hard to convert back to 1's and 0's!

To solve this problem, an alternative numbering system is used which is easily converted to binary, yet more compact to write down. This system is based, in microcomputers, upon sixteen instead of ten or two. The system is called HEXADECIMAL (six + ten) or HEX, for short. Its convenience, described below, arises from the simple fact that 16 = 2^4. (Any base, incidentally, which is a power of two would be applicable to the following process. E.g., using 8 = 2^3 gives rise to the OCTAL system not considered here)

To define the numbering system, a set of digits must be agreed upon.

0-9 are chosen for DECIMAL (10 digits)
0 and 1 " " BINARY (2 digits)
0-9 and A-F " " HEX (16 digits)

The digits 0 - 9 are supplemented by the first six letters of the alphabet for this process. If one had fifteen fingers, counting would proceed as follows:-

0 = 'no fingers'
1 = 'one finger'
etc.
9 = 'nine fingers'
A = 'ten fingers'
B = 'eleven fingers'
C = 'twelve fingers'
etc.
F = 'fifteen fingers'

Larger numbers are built up as follows:-

4B3 is interpreted as

16^2	16^1	16^0
4	B	3

(again, $16^0 = 1$)

Thus, $(4B3)_{16} = (4 \times 256 + 11 \times 16 + 3 \times 1)_{10}$
$= (1203)_{10}$

Hex is even more compact than decimal for describing a binary number and the following table shows how to convert between Binary, Hex and Decimal for the sixteen Hex digits.

BINARY	HEX	DEC
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Each Hex digit is "equivalent" to 4 Bits.

Patterns of 1's and zero's on any 8-bitbus may be conveniently written down at any instant by just two Hex digits. The convention as to the order of the lines is described in the following diagram for a typical Bus state:

Address Bus:	A7	A6	A5	A4	A3	A2	A1	A0
State :	1	0	0	1	1	1	0	1
Hex :	9				D			

Notice how the least significant BIT (binary digit), A0, is always written over to the right - as in our normal numbering system : where eight, for instance, is the least significant, digit of 378.

Thus we say the Address Bus above, contains the "ADDRESS" 9D. This will set a unique memory location. If the Data Bus then contains (for example) :

Data:	D7	D6	D5	D4	D3	D2	D1	D0
Binary:	0	0	1	0	1	0	0	0
Hex :	2				8			

we say that data 28 has been stored in or read from Address 9D. Be careful not to confuse 28 with "twenty-eight". Address and Data are always in HEX except where otherwise stated and the reader should develop the habit of saying, for instance, 28 as "two eight" rather than "twenty eight". This will prevent automatic confusions from arising - "two eight" in Hex is equal to forty in decimal!

This describes the system used for counting and labelling memory locations in microcomputers. The ADDRESSABLE memory locations in the EDUKIT can be described in the following map:

FF	8 bits
FE	8 bits
.	.
.	.
.	.
.	.
.	.
02	8 bits
01	8 bits
00	8 bits

There are 256 locations in the EDUKIT'S Addressable Memory - each labelled by a pair of hex digits and each containing 8 bits of information.

To understand the map fully, the method of counting must first be explained. In the previous table, the first 16 Hex numbers are described. Thus, counting in Hex begins at 0, goes through 1,2,3, etc, to 9, just as in our normal Decimal system. Then comes A - F. "What comes after 'F' ?" is the main question. The answer is found from our Decimal system:

0, 1, 2, 3, ...,9, 10, 11,.....,20, 21, etc.

After the last single digit, comes 10 (ten). I.e., the least significant digit (as it is called) is set to 0 and the next digit (to the left) to 1. Then comes 11, 12, etc up to 19. Again, the least significant digit is set to 0 and the next digit incremented by one to give 20 (twenty). In Hex, the system is similar, but the most significant digit is incremented (by one) after F instead of 9. I.e.:

0, 1, 2, 3, 4, ...,9, A, B, C, D, E, F, 10, 11, 12, 13, 14,...,1F, 20,21, 22,..etc.

The 10, for instance, is not "ten" but the next number after "F". The next number after 2F is 30 and so on. The next number after FF is 100 and is too big to be represented by 8 bits. Thus, if only 8 bits are available for presenting data, the "1" in the "100" is discarded and the next number after FF is 00 - back to the beginning. This is exactly what happens in the EDUKIT. If you try to go above FF, you simply start back at the bottom of the Address Map again. We say that the memory block "wraps around" on itself after 256 bytes.

Addition in Hex is similar to our normal Decimal Addition. To add two numbers, e.g. 567 and 850, (in Decimal) we perform the following calculation:

```
  5 6 7
+ 8 5 0
-------
    7
    0
```

we add 7 and 0 together first (the least significant digits) and write down any "carry", which is zero here.

```
  5 6 7
+ 8 5 0
-------
1 4 1 7
-------
  1 1 0
```

Then 6, 5 and any "carry" are added and the carry, "1" here, written down. Finally, the process is completed with the final carry (again "one") appearing in the most significant position (far left). Note that the maximum carry possible during addition of two numbers is one - even if you add 999 to 999.

Addition in Hex is similar - the prime difference is that "carries" are not performed after 9 has been reached in any column, but after F instead. For example : 3 + F = D, in Hex, because D = A + 1 + 1 + 1. Similarly, 5 + A = F. However, if the result is above F, a "carry" is generated. E.G., 6 + A = 10, 8 + A = 12 etc. You will need to be able to add any two Hex digits to understand the next step.

Try checking that:

9 + 3 = C, 6 + E = 14 (do not pronounce this as "fourteen"! use "one four")

and E + E = 1C.

Use your fingers if you must - most people do! Again, the maximum possible "carry", in Addition, is a "one".

To add the multidigit numbers 10463 and FAB68, proceed as follows:

```
  1 0 4 C 3      8 + 3 = B, carry 0.
  F A B 6 8      C + 6 + carry = 12 - i.e. carry 1, 4+B+1=10,
-----------      0 + A + 1 = B (carry 0), and finally,
1 0 B 0 2 B      1 + F + 0 = 10 and the final carry appears
-----------      in the most significant position.
  1 0 1 1 0
```

This should be practiced by trying the following excercises, the answers to which are given after the Summary to this chapter.

a) 7 6 7 2 + 2 9 9 9 b) A B C D + 6 9

c) The answer to: E 7 9 7 + 8 7 6 is not given - you'll know the right answer if you can eat it!

The point of learning this arithmetic is, firstly, to be able to use the EDUKIT's Memory Map correctly and, secondly, because one of the MPU instructions is Binary Additon of 8-Bit numbers - i.e. 2-Hex-digit numbers may be added together and displayed by the computer. By "ganging" such additions together and "carrying" from one to another, as above (a maximum of "1", remember), numbers with any number ofdigits may be added. Multiplication is the next step and simply consists of many additions - e.g. 3 x 6 is merely a question of adding together three 6's - a reasonably straightforward routine.

To return to the Memory Map and Programming : Instructions, in general, are nothing more than the contents of one, two or three successive memory locations, and any program has all or part of its first instruction stored in location 00, next part or instruction in 01 etc., etc. The MPU fetches the contents of these locations one after the other and executes them. The electronics of the EDUKIT allows the user to set up this program for the MPU through the keyboard and then Command the MPU to execute the program. The program must be self-terminating or end with an instruction to tell the MPU to run the program again from the beginning rather than ploughing straight through memory, trying to execute indiscriminately,the contents of every available memory location.

The instructions, themselves, are stored in memory using a special code - each instruction has one such OPERATION CODE (OP code) defined for it by the MPU manufacturer. For instance, if the user wishes to switch on the Q LED, there is a special instruction called SET Q having Hex Op-code 7B. This represents an 8-bit pattern of 1's and 0's which may be stored in address 00, for instance. When the RUN state is entered, the MPU places 00 automatically on the Address Bus, along with the necessary Bits on the Control Bus, and 7B (contents of 00) appears, from memory, on the Data Bus. This is read by the MPU, and executed, and the Q light comes on. The MPU then automatically fetches the contents of 01 and executes it etc. The list of instructions available is printed in Appendix III, and described in detail later.

SUMMARY

a) Buses have lines, the states of which are described by 1's and zero's or by Hex digits
b) Each memory location contains 8 BITs (binary digits) called 1 BYTE. This requires 2 Hex digits because each HEX digit represents 4 BITS.
c) Bytes are stored in memory and may be interpreted by the MPU as program commands.
d) Each memory location has an 8 Bit (2 Hex digit) address, and this allows 2^8 = 256 different possible addresses.
e) Addition in Hex involves the same principles as those in decimal, with the "carry" (0 or 1 only) occurring after F instead of 9.

Answers to exercises : a) A0 0B b) AC 36.

INITIAL USE

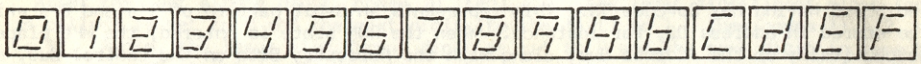
Having soldered the EDUKIT together, its operation must be checked. If you refer to the Components Diagram, you will notice three power supply pins. If you are using a 6 volt battery or supply, the positive(+) side must be connected to the + 6 volt pin, and the other side (-) to the 0 volt pin. If a + 5 volt supply is being used the + 5v pin is used instead of the + 6 volt pin. These pins are also marked on the PCB surface. The device takes between a third and half an Amp during operation, and thus a very small battery is of little use. Two large three volt batteries are satisfactory if connected in series - i.e. plus to minus to form a chain of batteries with one positive and one negative end. Alternatively, a power supply is the long term solution, and the comp onents for a suitable supply may be purchased from MODUS.

A word of caution is necessary at this point. The power must be connected the correct way around or permanent damage will result. In addition, no more than + 5 volts or + 6 volts must be connected to the respective pins. As a 6 volt battery wears down, however, its voltage drops sufficiently to allow it to be connected to the + 5 volt pin. If the battery has not been in use for an hour or two, it will "rejuvenate" itself slightly and return to a normal voltage for a short while ; thus, a 6 volt battery should be tried on the 6 volt terminal first until it stops supplying enough power, before swopping to the + 5 volt pin.

Having connected the EDUKIT to a suitable power source, some of the LED (Light Emmitting Diode) lights should be glowing red. If not, refer to the section on trouble-shooting.

Assuming normal operation, try pressing the R and L buttons on the keyboard and note the effect on the associated lights. Pressing R should change the state of R light - if it was off, it should light and vice versa; similarily for L. By this means, the R and L lights may be lit in any pattern - i.e. both on, both off or just one on. Check this is possible and practice doing so before reading further.

The next step is to switch the toggle-switch to M/P (i.e. to right). This protects the memory of the computer from being overwritten and is called MEMORY PROTECT. Now, press R and L until both LED's are alight - this RESETS the machine. It is now ready for you to command in some way. Press R until the R light is out leaving just the L light on. This puts the computer into a state called the LOAD mode. In this state, the computer's memory contents may be loaded with a program, or examined, for checking purposes. You are going to READ the memory's contents - all 256 locations, if you wish. Press the I (or In) key once. The two-digit display just above the keyboard will now be showing you the contents of the Address00. Each digit shows one of the 16 Hexadecimal digits 0-9 or A-F in the following format - some of the letters are upper case and some lower case - make sure you can recognise them - take careful note of the difference between the letter B and the number 6.



Two Hex digits will have appeared, compare them with those in the diagram above. Try pressing I again, the contents of 01 will be displayed. If you continue pressing I, the contents of each memory location from 00 to FF (256 locations) will be displayed. The next location after FF will be 00 again and the process will repeat. Normally, switching on the machine will store a random set of numbers in memory which will be displayed, byte by byte, by the above process. If you are unfamiliar with Hex displays, step through memory, as above, checking you can read the displayed characters accurately each time. To start displaying from the beginning again, first RESET the machine back to the start by pressing R, thus lighting both lights again. To read the contents of 00, press R again to leave just the L light on and press I. Again, further pressings of I will display contents of 01, 02, 03 etc. This should be practiced now.

ENTERING DATA AND PROGRAMS

The above describes how to view memory - an action you will always have to perform after loading your own memory contents.

To load some data, Reset by lighting R and L lights and switch R off with the R button as above. The LOAD mode is thus entered again. This time, however, set the toggle switch at the top of the board OUT of the M/P position - i.e. to the left. This removes Memory Protection (M/P) and allows the memory's contents to be changed. Whatever you type into the keyboard can be stored at locations 00 at this point by the following method:-

Suppose 7B is to be stored in location 00. Press 7 key, then press B, then press I. This is represented by :- (7)(B)(I) The Hex display should show 7B at this point.

If not, press the AM (Amend) key and, still keeping it down, press (7) (B) - the keys are sometimes sticky and (7) (B) may have to be pressed several times for the required effect. Watch the display because whatever you press while the AM key is down will appear on the Hex display and be stored in the current memory location at the same time. 00 is the curren location until I is pressed again.

If 7B appears and remains there with your fingers off the keys, it is successfully stored. Do not press I after the AM key or the next location will also contain a copy of the corrected data and two copies will thus have been written to memory.

To pass to the next location try entering 7 A by

(7) (A) (I)

then (3) (0) (I) and (0) (0) (I) .

These four bytes represent a program and when the RUN mode is entered, the program will be executed byte by byte. First, however, the contents of the first four addresses in memory will be reviewed, to check that they contain the information just keyed in.

To view the locations :-

a) Press R until both lights are on (RESET)
b) Press R to extinguish the light and leave L light alone (LOAD mode)
   This resets a "pointer" in the MPU to looking at Address 00.
c) Set M/P on (to the right)
d) Press I.

This should display 7 B. If not, press 7B with the AM key pressed - this will correct the mistake: press L again. This should display 7 A and so on. 30 and 00 should follow with each I. The Memory Map thus contains

00 : [7 B] 01 : [7 A] 02 : [3 0] 03 : [0 0]

The program, for which 7B, 7A, 30 and 00 are the OP CODES, will have the following effects. 7B sets the "Q" Flag to a "1" (described later) and pin 4 of the MPU will then show + 5 volts. This is connected to a transistor on the EDUKIT and then to a pin on the PCB. Another pin, nearby, completes the circuit through an LED. If the two "Q" pins are bridged by a conductor, when the Q flag is set to a "1", the LED lights up. Thus, when the program is run, as described later, 7B is fetched and executed by the MPU, whcih lights the LED.

Then 7A is fetched. This is the code for an instruction to reset the Q flag to zero. The LED goes out again very quickly. 30 is an instruction to tell the computer to Jump somewhere else for the next instruction - 00 tells it where to Jump. I.e. 30 00 is a two-byte instruction to tell the computer to jump back to address 00 and fetch and execute the 7B. This is fetched again and the LED lights again. Then 7 A turns it out again, and 30 00 Jumps back to repeat the "LOOP", as it is termed, over and over again. The LED switches on and off many times a second, and should glow faintly, as far as the Human eye is concerned.

Now switch the L and R lights on, thus resetting the machine to 00. When L is pressed to switch the L light out, the R light will remain on. This puts the EDUKIT into RUN mode and the program is executed from 00 onwards. If you have a small loud-speaker handy, remove the bridge between the pins and connect the loudspeaker to the pins. The current flowing on and off through the LED is thus forced to pass through the loud-speaker. These cycles occur too fast for the Human ear to perceive as separate sounds and it merges them together into a continuous "musical" note.

The pitch, or frequency of the note, is determined by the speed with which the program runs - i.e. how fast the MPU can fetch and execute. This, in turn, depends upon the frequency of the "clock" from which all instruction-timing comes. The clock frequency depends upon power supply voltage and a fresh battery or stabilised power supply will give a higher note than an old battery.

The pitch of the note may be lowered by programming as follows. This uses the "NO OPERATION" instruction or "NOP". It has the OP-Code C4. If placed in a program, it is fetched and executed, but literally causes nothing to happen. Howver, it takes time to fetch and "execute" and hence can be used to "waste" time. If the program: 7B C4 7A C4 30 00 is loaded and run, a slightly lower note is produced. The program sets the Q flag on, leaves it on while C4 is executed, then resets Q again and wastes some time before returning to set Q again.

If twenty or thirty C4's are put after 7B and 7A, the speed of operation is slow enough to allow the observation of the Q light flickering on and off - try it !

The machine can be made to "PAUSE" by pressing R to turn both LED's off. The machine's state is frozen and pressing R, again, causes the program to continue running.

SUMMARY

a) Programs are sequences of OP CODES stored, normally, from 00 onwards. The MPU interprets the OP CODE as an instruction when it is fetched.

b) There are four "Command" buttons and one toggle switch. The toggle switch is switched to M/P when viewing data loaded into memory locations.

c) As R and L are pressed, four Commands are given, depending upon the state of the LED lights.

Both on .....RESET back to 00
R on alone.....Running a program from 00
L on alone.....Loads information from 00
Both off .....Operations frozen.

The "IN" key is used to step to the next location during loading (M/P off) or viewing (M/P on) of memory contents.

"Am" is used to amend the current memory location during loading or viewing.

d) Fetching and executing takes time, and time may be wasted without affecting the machine's state by using C4 - the OP CODE for NOP or NO OPERATION.

e) A note may be produced through a loudspeaker by operating the Q Flag many times a second.

CHAPTER 4

MACHINE CODE PROGRAMMING

The Ideas introduced in this chapter are entirely dependent upon an understanding of the previous material in this manual.

In the last chapter, you were shown how to load, run and check a program on the EDUKIT. In this chapter, the set of instructions available for inclusion in a program is introduced. For the purpose of programming, the MPU is not considered electronically in any way. Instead, it is regarded as a machine capable of manipulating memory locations. There are two types of memory location - one of which, Addressable Memory, has already been introduced. The block diagram of the EDUKIT contains a block labelled "memory" - this is where Addressable Memory resides and is physically provided by the "2111" IC's on the PCB. The other type of memory contained in a microcomputer is not addressable by Hex address codes and, in fact, resides within the MPU itself. These locations are called "REGISTERS" and are so important that each has a separate name as opposed to a Hex address. In addition, many of them have very specific functions and all have special OP Codes associated with their use.

There are over forty Registers for use during a program - to store intermediate results etc.

All MPU operations involve at least one of these registers, and it is fair to say that all an MPU does is to "shuffle" register contents around and control the Bus lines. The MPU knows nothing of the various IC's making up the rest of the microcomputer - the "outside world" exists only insofar as it accepts register contents placed on Bus lines and provides Bus-contents to be stored in registers at the appropriate time.

The set of registers provided on the EDUKIT is decided by the manufacturers of the MPU - the RCA COSMAC 1802, to give its full name. The set of forty registers is given in the following two tables with name, number of bits stored and purpose.

High Order 8 - bits	Low Order 8 - bits
R(0).1	R(0).0
R(1).1	R(1).0
R(2).1	R(2).0
"	"
"	"
"	"
"	"
R(F).1	R(F).0

Sixteen 16 — Bit General Purpose Registers - each split into two bytes - a HIGH order and a LOW order byte.

Name	Bits	Purpose
D	8	Data register (Accumulator)
DF	1	Data Flag (ALU carry)
R	4	Pointer to one of General Purpose Registers.
P	4	Points to program Counter
X	4	Points to Data Pointer
N	4	Holds low order instruction digit
I	4	Holds high order instruction digit
Q	1	Q flag

(there are actually two more registers labelled "IE" and "T" which are not discussed in this chapter).

The 32 general purpose 8-bit registers are not labelled from 1 - 32; instead, they are ganged together in pairs - one called the "High Order byte" labelled with a "1" and one "Low Order byte" labelled with "0".

Note that R(0) (and indeed, any R-register) has 16 bits, whereasaddresses, so far, have always been 8 bits long. This is because only 256 locations are available as $256 = 2^8$. The MPU itself is capable of addressing 65,536 locations, in fact, or 2^{16} - which requires 16 bits. However, in order to simplify matters, the EDUKIT's electronics ignores the upper (or High Order) byte of this 16-bit addressing capability and uses just the Low Order byte. The result is that when an R-register is used for addressing (e.g. when used as the PC) only the Low Order half is significant. Thus, when we say that R(0) is the PC, we will mean that R(0).0 is used. In referring to addresses, we should really use 4 Hex digits for the 16-bits. However, the upper 8 bits are redundant and hence we shall stick to two Hex digits as usual.

The next most important register is the 8-bit D-register. This is a typical computer Accumulator and is best described by reference to the Human brain. Try multiplying 3 by 651 with your eyes closed. You will find yourself using your own "Accumulator" often referred to as the "Mind's Eye". As you perform the calculation, intermediate results are stored in yourown D- register. The situation is very similar in an MPU and many of the operations which your programs will perform will go via the D-register. A good example of the process is provided by Addition. This is described below, along with some other instructions to complete the definition of the Register Set, and introduce some important example-programs.

ADDITION

Chapter II described the process of adding in Hex, it will be assumed,here,that this is fully understood.

The op-code used for the Addition instruction depends upon the type of addition and where the two numbers to be added are stored. The "ADD IMMEDIATE" instruction (with short form, or MNEMONIC, ADI) is a 2 - byte instruction, just as the JUMP instruction (30) must be followed by a second byte to give an address to JUMP to.

The R-registers are thus 16-bit registers where each 8-bit half may be "contacted" separately. R(A).1, for instance, is the high order byte of register number A (Hex for ten, remember).

In the last chapter, a small program to set and reset the Q light was introduced. The Q LED is connected (via a transistor) to a pin on the MPU itself. This pin connects internally to a 1-bit register called the Q flag (see the above tables). The word "flag" is often used to denote a 1-bit register. A single-bit memory device is sometimes referred to, also, as a FLIP-FLOP. The Q flip-flop may be read by the MPU and some action taken, depending upon whether it is set or reset - this is described later.

During the execution of a program, it is essential for the MPU to know from where the next instruction is to be fetched. A special register called the PROGRAM COUNTER (PC) is used by the MPU to store the address of the next instruction. Thus, in program : 7B 7A 30 00, which sets and resets the Q flag, execution begins at 7B (stored in address 00), proceeds to 7A (stored in 01),then to 30 00, which tells the MPU to go back to 7B in address 00 and begin again.

During this process, the PC starts at 00, increments to 01, then to 02 and 03 and back to 00. To make the MPU execute a program, all that is needed is to set the PC to the address of the first instruction in that program and the MPU carries on from there. There are instructions devoted to changing the PC at will.

The Program Counter is one of the R-registers, and the P-Register (see the table) tells the MPU which R-register it must use for the PC. P is a 4-bit register. If it contains 0101, which is Hex "5", then R(5) is the PC. Very often, P contains (Hex) 0, and thus R(0) is the PC - this is always true, in fact, just after RESET (R and L lights on).

ADI needs 2 bytes, the first of which is always FC. This instruction simply takes the second byte and adds it to the contents of the D register, leaving the result in D. Thus, if D contains 26 (always in Hex) the instruction FC 30 produces the result 27 + 30 = 57 in D. If you look in the "Arithmetic Operations" section of Appendix III, you will see the ADI instruction described briefly. In the description, you will notice the notation M(R(P)) + D → DF,D;R(P) + 1 to briefly define the operation.

R(P) is the general purpose register being used as the Program Counter (PC), and M(R(P)) means the contents of the memory location addressed by the PC. When the first byte of the FC 30 is fetched, the PC stored by R(P), is set to point at the next byte - i.e. 30.

Thus M(R(P)) = 30 and hence FC 30 causes M(R(P)) + D to be executed. The arrow which follows implies that the result is stored in DF and D. DF is the 1-bit register (see register table) which stores the final "carry" (only 1 or 0, remember) if there is any. Finally, the Program Counter must be incremented to point to the beginning of the next instruction and hence R(P) + 1 appears. In this manner, any pair of 2-hex-digit numbers may be added and the carry retained for further additions. To add 4-figure numbers, such as 12FF & FB24, first the FF and 24 are added to give 23 in D and 1 in DF (i.e. FF + 24 = 123), and then 12 + FB + Data Flag is performed which equals 12 + FB + 1 = 10E. This leaves 0E in D and DF = 1. Thus 23 is the least significant part of the answer and 10E is the most significant. It was necessary to "add with carry" on the second addition.

There are two instructions in Appendix III which allow this type of addition. You will notice, therefore, that there are four addition instructions. Two of which add without "carry" and two with.

The difference between ADD and ADI, in the Arithmetic Operations section of Appendix III, is in where the byte to be added to D is found. The Immediate instruction always includes the data byte to be added in its second byte, as explained above. If you look at ADD you will see that the definition is : M(R(X))+D → DF, D. This implies that the byte to be added to D is found in M(R(X)). X is another 4-bit register, like P, which is used to point to one of the R - registers. X may be set by the SETX instruction (see "Control Instructions" in Appendix III). In this way, the byte to be added to D may be stored in any memory location and the address of that location stored in R(N) (where N is a Hex digit). X is then set to N & an ADD Instruction is executed. To set X to N, Appendix III shows that the op-code EN is used, this is a 1-byte instruction where N is any hex digit to be loaded into X.

No example-programs of the above can be given before a few more instructions are introduced.

SOME OTHER INSTRUCTIONS

Before continuing, glance through the sections in Appendix III devoted to : Memory Reference, Register Operations, Logic Operations, Arithmetic Operations and Control Instructions. Notice that several of these instructions come in an "Immediate" form - in each case, the instruction has two bytes, with the data stored in the second byte. Also, many of the instructions in these sections rely on the notations M(R(X)), M(R(N)) and M(R(P)). M(R(X)) has been explained - it depends upon X pointing to an R-register containing an address.

When an instruction byte is fetched by the MPU using the address in the PC, that byte is stored in two halves - one in the I register and one in N. Thus, when the op-code E5 is fetched, E is stored in I & 5 is stored in N. Thus R(N) is an R - register pointed to by the Low Order 4-bits of the instruction just fetched. See, for instance, the INC instruction at the beginning of the Register Operations Section in Appendix III. 13 would be fetched and N set to 3, this would be used to tell the MPU to increment R(3) by 1.

M(R(P)) is simply the Contents of the next memory location to be fetched during a program. Thus M(R(X)), M(R(N)) and M(R(P)) are contents of address locations pointed to by R-registers. The particular R-register implied depends upon the value of X, N, P registers respectively. This is called INDIRECT ADDRESSING. Rather than directly telling the MPU to fetch some data from a specific memory location, it is given the number of a register which contains the address of the data instead.

If two numbers are to be added, D must first be set to one of them. This requires a LOAD instruction - see "Memory Reference" Instructions. The final answer must be accessible to the user and thus a STORE instruction is used to store the contents of D in some memory location. This may be read by the user by stepping through memory with the In key or, alternatively, displayed automatically on the digital display using another instruction.

LOAD

"Load Immediate" is a two byte instruction with mnemonic LDI and op-code F8 which simply loads the second byte into D. Thus F8 31 sets D to 31. If the byte to be loaded is somewhere else in memory, then the address of this byte's location is stored in R(N) (for some N) and LDN instruction used instead. If a number of bytes are stored consecutively in a table, and these bytes are to be loaded and used one after another, it is wasteful to have to increment the contents of R(N) to point to the next byte each time, and the LOAD ADVANCE (LDA) instruction is used,which automatically increments R(N)'s content

In addition, X, instead of N, may be used to point to the appropriate R-register (R(X)) and then LDX or LDXA is used with effects as shown in the table.

STORE

There are only two STORE instructions available - one via N, and one via X with an automatic decrement of R(X)'s contents.

In order to set R(N) to a given value, the GET and PUT instructions in the "Register Operations" sections must be used. R(N).1 and R(N).0 are set separately and all such setting is performed through D;i.e., to set R(5).1 to 3F, it is necessary to LOAD D with 3F - perhaps using F8 3F - and then PUT HIGH R(5), with op-code B5 (see table of op-codes). Thus setting R(5).1 to 3F is effectively a 3-byte instruction : F8 3F B5, though "officially" it is a two byte instruction followed by a one byte instruction.

DISPLAY

To display a byte on the digital display, an OUTPUT instruction must be executed - see the INPUT-OUTPUT BYTE TRANSFER section in Appendix III. These instructions transfer bytes directly to and from the DATA BUS.

There are three N lines : N2, N1, N0 which come out to pins on the MPU. When Output Instruction 61 (OUT 1) is executed, the N lines take up the pattern:

N2	N1	N0
0	0	1

Some external electronics is meant to recognise this pattern and route the Data Bus Contents to the appropriate external device. In the EDUKIT, OUT 4 is used to "contact" the digital display. If 64 is executed the n-lines take up the pattern:

N2	N1	N0
1	0	0

which is the binary for 4. Some electronic logic recognises this and activates the digital display which then takes the Data Bus Contents (from location pointed to by R(X)) and converts this to a pair of Hex digits for the operator to read. This is extremely useful for displaying the results of a program.

IDLE

Finally, before considering some examples, it is helpful to know how to end a program. The Idle instruction "IDL" (see Control Instructions) tells the Computer to stop execution - its op-code is 00. This condition is maintained until the machine is RESET (or electronically interrupted in a manner to be described later).

EXAMPLES

Now we can apply some of the theory described in this chapter. It is assumed that you have read and assimilated Chapter III on using the EDUKIT, and can Load & Run programs, inspect memory locations and use the Amend key.

A constant format for writing down programs will be maintained throughout, for consistency, with addresses, op codes, mnemonics and comments included as below.

TO DISPLAY THE CONTENTS OF A MEMORY LOCATION

Start by loading any byte (3F will be used below) into any location (08 is used here). To do this, RESET (both lights on), press R to leave the LOAD light on and press the In key NINE times (i.e. step through locations 00, 01, 02, 03, 04, 05, 06, 07, 08). Press the "Am" key, this will ammend the contents of 08 to 3F.

The program to display the contents of 08 is as follows:

Address	Op-Code	Mnemonic	Comments
00	E5	SEX 5	Set X to 5
01	F8 08	LDI 08	Load D with 08
03	A5	PLO 5	R(5). 0 set to 08
04	64	OUT 4	Display M(R(5))
05	00	IDL	End.

To load this program, only the op-code Column is loaded. I.e., Reset the machine and load E5, F8, 08 etc. Then Reset and run the program. The MPU fetches E5 and stores E in register I, 5 in register N and sets register X equal to 5. Then the D-register is loaded with 08 and this stored in R(5).0 for use as an address.

Only the lower 8 bits of an address are used by the EDUKIT, therefore R(5).1 is left untouched. Finally, the OUT instruction uses X = 5 to choose R(5) to address the memory location 08. Here the byte to be displayed is held, and 3F should have appeared - the program then stops, leaving 3F on display.

The above format for writing programs down is almost self-explanatory. Notice that the Address column contains the address of the first byte in each row, and thus E5 is stored in 00, F8 in 01, 08 in 02, A5 in 03 etc.

ADDITION OF TWO NUMBERS

This program uses a few more instructions and is used to add any two numbers stored in memory locations 02 and 03. The answer comes up on the display, and is stored in location 03.

00	30 04	BR 04	Branch into program
02	x y	- -	Two bytes to be added
04	F8 02 A5	LDI02 & PL05	Set R(5).0 to 02
07	45	LDA	Load via R(5) and advance
08	E5	SEX 5	X set to 5
09	F4	ADD	Add D to M(R(X))
0A	73	STXD	Store D at M(R(X)) & decrement R(X)
0B	60	IRX	Increment R(X) back again
0C	64	OUT 4	Display M(R(X))
0D	00	IDL	End

Before reading the explanation of this program, type it in with two simple numbers for x and y. Try 12 and 34 to give 46 and check that the answer appears. After loading the program, it will only work with M/P in the off condition. This is necessary since otherwise the memory is protected against being overwritten, and the STORE instruction, in location 0A above, cannot perform its task. You should check your program through before running it as typing errors are easy to miss.

While writing or explaining a program, a "dry run" table is most beneficial to its understanding. The registers and locations affected by the program are written in a table and their contents entered in columns as the program is "mentally" or "dry" run.

row address	D	R(5).0	X	02	03
04	02	02	-	12	34
07	12	03	-	12	34
09	46	03	5	12	34
0A	46	02	5	12	46
0B	46	03	5	12	46

Refer to this table for the following explanation.

This table holds one complete "dry run" of the above program. At address 00, the 30 instruction followed by 04 tells the MPU (via the PC) to Jump over the data in 02 and 03 and execute from 04 onwards. It is essential that 00 contains an instruction and not data, but it is useful to have small amounts of data near to 00 where they are easily accessible by the user.

Location 04 contains F8 followed by 02 and A5. This loads 02 into D, and then 02 (from D) into R(5).0. The state of the memory locations and registers after this row of instructions is written in the above table, along with the address of the first byte in that row. X's contents are indeterminate at this point. Then 07's instruction is fetched and executed - this causes D to be loaded from the address 02 - pointed to by R(5).0.

In addition, R(5).0 is advanced to 03 to point to the next piece of data. 08's instruction just sets X to 5 and this is included in the next line of the dry run table. Then F4 is executed which takes X's value and uses R(X) to find an address (03) whose contents (34) are added to D. R(X) is still pointing to 03, and this is used in the next instruction, 73, to store the answer (in D) in memory. This instruction decrements R(X) at the same time. Note that storing D does not change its contents - D is just copied into memory.

R(X) is then incremented back to 03 and the number stored at this address is the answer. This is then displayed using 64, and the program ends with an IDL instruction. If you follow the dry run table and program at the same time, a little effort will make the process clear and your knowledge and understanding of computer programming will be greatly enhanced.

After running the program, set M/P on and examine the program to see that location 02 still contains 12, and 03 contains 46. Reset and set M/P off again and Run the program without changing anything - can you explain why the display contains 58? Before resetting & running again, check that you can predict the next number.

The problem comes if x and y generate a "carry". The DF register would be set but not displayed above. The Q light can be used to display its value. To achieve this, a "Conditional Short Branch" instruction is introduced.

After the ADD instruction is executed in location 09, the Q light is to be lit if DF is set. If we just insert a SEQ (set Q) instruction after F4, the Q light will always be set whether DF is 1 or 0.

Instead, a further instruction is inserted, which skips over the SEQ instruction if DF is zero, thereby leaving Q at 0 if DF is at 0, and 1 otherwise.

The BNF or "Short Branch if DF = 0",in the "Short Branch" section of Appendix III, is appropriate. Short Branch instructions are two-byte instructions whose second byte provides an address to branch to.

30 has been used before, and is an unconditional branch. BNF, however, only branches to the indicated address if DF = 0. The program may nowbe written (slightly condensed) as :

```
00   30   04        REQ        Reset Q to zero
02   x    y         BNF 0E     branch if "carry" = 0
04   F8   02        SEQ        set Q if "carry" = 1
07   45              STXD
08   E5
09   F4
0A   7A
0C   3B
0D   7B
0E   73
0F   60
10   64
11   00
```

To run this program, the Q pins (next to the Q light) must be bridged by an electrical conductor - also M/P must be off again.

Load the program and try some numbers in x and y. If you choose 12 and 34 again the result is 46 with the Q light off indicating no carry. Try F3 and 42 - the result should be 135. The display shows 35 and the 1 appears on the Q light. Try resetting and running again. You can add multidigit Hex numbers by adding them in pairs and carry ing from pair to pair - try using this, along with a pencil and paper, for the excercises in Chapter 2.

The program works as before up to the "Addition" in location 09. Q is then reset and the Data Flag examined for a "carry". If DF = 0, the branch occurs and SEQ is not executed - the Q light remains off. If a carry has occurred, the branch does not take place and SEQ turns the Q light on to indicate this condition.

BRANCH AND SKIP INSTRUCTIONS

The 1802 Microprocesser is quite rich in branches as can be seen from Appendix III. The Long Branch group will not be considered here as these 3 — byte instructions contain a 16 — bit address which the EDUKIT is not set up to recognise.

Branches are used for decision processes as illustrated in the previous section. They simply change the order of execution of a set of instructions stored in consecutive memory locations. There is a variety of conditions upon which a branch can occur as can be seen from the instruction table. BZ branches if D is zero, BNZ if D not zero etc. All the arithmetic branches refer to the state of D. In addition, the state of the Q flag may cause a branch to occur. An example of the use of the Arithmetic Conditional Branches is in producing a time delay. The following program illustrates this idea.

```
00   7B          SEQ          Q light on
01   F8 00       LDI 00       D = 0
03   FC 01       ADI 01       Increment D
05   3A 03       BNZ          Repeat until zero
07   7A          REQ          Q light off
08   FC 01       ADI 01       delay again
0A   3A 08       BNZ
0C   30 00       BRA          Branch back and repeat.
```

Load and Run this program with the Q pins shorted together - the Q light should switch on and off about once a second. Q is set and remains On while D, which starts off at zero, is incremented. Eventually, D reaches FE then FF then 00, with a "carry" which is ignored. The "Branch if D not zero", in location 05, no longer branches back at this point, and Q switches off - a condition which lasts during the next delay. Finally, the program repeats by branching, unconditionally, back to the start.

There are many different ways to achieve this effect, see if you can produce some others.

There are four pins on the 1802 called EF1, EF2, EF3 and EF4 whose state may be read by the MPU and a branch effected depending on their condition. These lines are perfect for accepting switch-state information from a device which the EDUKIT is controlling. This is taken up again later.

Another type of jump is the SKIP instruction. These instructions are one-byte instructions which simply skip over the following two bytes if the Conditions shown in the table are satisfied. The Short Skip is the only unconditional Skip instruction and is the exception in that it skips just one byte. The following program illustrates the use of Skip to switch Q on and off repeatedly as above.

```
00   7B          SEQ          Q light on
01   F8 80       LDI 80       Start D at 80
03   FC 01       ADI 01
05   3A 03       BNZ 03       } Delay
07   CD          LSQ
08   30 00       BR 00        Skip over branch if Q set.
0A   7A          REQ
0B   30 01       BR 01
```

This program sets Q at the start, delays, and then asks whether Q is set. The first time around, Q is set and the Skip is performed causing Q to reset. The process is repeated missing out SEQ, and thus next time LSQ is encountered, Q is reset and the Skip not performed - the program restarts at the beginning and sets Q again. This saves just one byte of program over the last method. The following program is even shorter and its working is left as an excercise.

```
00   FC 01       ADI 01
02   3A 00       BNZ 00
04   CD          LSQ
05   7B          SEQ
06   38          SKP          Short Skip
07   7A          REQ
08   30 00       BR 00
```

This routine provides an illustration of the use of Short (unconditional) Skip.

SUBTRACTION INSTRUCTIONS

As with addition, the process of Hex subtraction is best seen by analogy with Decimal. Consider 566 — 389 in Decimal :

	hundreds	tens	units	
	5	16	16	10 is borrowed from the 10's and 16 - 9 = 7.
—	4	9	9	The 80 is then incremented by 10 to make up
	1	7	7	for the borrow, and becomes 90.

100 is borrowed from the 100's to convert the 60 to 160 and 16 - 9 = 7. Again the borrow is added back turning 3 to 4. 5 — 4 = 1 and no further borrow is needed. If 389 — 566 was performed, there would be a final borrow. To see how this works 389 — 566 is written as:

	1000's	100's	tens	units	
	0	13	8	9	The subtraction goes as normal, but 3 - 5
—	1	5	6	6	needs a borrow from the 1000's column.
	−1000 +	8	2	3	Then 13 - 5 = 8, but this leaves 0 —1000 at the left hand side.

The answer is thus : -1000 + 823 = -177 which is correct. 823 is an important intermediate stage.

The final result is found by subtracting 823 from the next higher power of 10 (i.e. 1000) and adding a "minus" sign. 823 is called the "TEN'S COMPLEMENT" of the answer. There is a simpler method of converting 823 into the final answer called 9's Complementation. Each digit of 823 is subtracted from 9, and 1 added to the final total. A "minus" sign is then attached as before. I.e. 823→176, and 176 + 1 = 177 giving -177 as the answer! A similar process is used below.

Subtraction in Hex proceeds as follows:

	16's	units	
	9	1A	10 (Hex) must be borrowed from the 16's to turn A
			into 1A, and then 1A-C = E.
—	7	C	The borrowed 10 is then given back to the 6,
	2	E	and 9 - 7 = 2.

If 6C - 9A is performed instead, it is again easier to write:

	256's	16's	units
	0	16	C
−	0	9	A
	-100 +	D	2

C - A = 2 and 16 - 9 = D, with the borrow left at the end to give :
-100 + D2 as the final answer.

A Hex subtraction still has to be done to finish! However, as for Decimal, an easier process exists. D2 is called the "Sixteen's Complement" or (more normally) "TWO's COMPLEMENT" form of the answer.

To turn D2 into the final answer, it is best written in binary as : D2 = 1101 0010.

Each "1" is then turned into a "0" and each "0" into a "1" - the result is 0010 1101. "1" is added as follows:

```
    00 10  11 01
+              1
    ------------
    00 10  11 10
    ------------
           1
```

with the usual rules of carrying.

This becomes the final answer when a "minus" sign is attached. i.e. - 0010 1110 = -2E which is 6C - 9A. The process of changing 1's to 0's, and vice versa, is called COMPLEMENTATION. One of the "Logic Operations" in Appendix III may be employed to perform this automatically. The operation is XRI with FF, this is explained in the next section. The DF register signifies if a final borrow is needed. DF = 1 if not, and 0 otherwise. Thus, DF = 0 signifies that the answer is in 2's Complement form, and needs XRI FF followed by ADI 01 to give the answer.

The following program illustrates the subtraction instruction. The program is similar to the Addition programs in the previous section.

```
00    30 04      BR 04
02    x  y       data
04    F8 02 A5   LDI02 & PL05
07    45         LDA
08    E5         SEX 5
09    F5         SD          M(R(X)) - D
0A    7A         REQ
0B    33 0F      BDF 0F      Branch if DF = 1
0D    30 13      BR 13       Branch down to complement
0F    73         STXD                and add.
10    60         IRX
11    64         OUT 4       Display
12    00         IDL         End
13    FB FF      XRI FF      Complement
15    FC 01      ADI 01      Add one
17    7B         SEQ         Set Q light on
18    30 0F      BR 0F       Return to display
```

Here, y - x is calculated, and, if negative, automatically Complemented etc., and the Q-light lit to indicate that the result is negative. To understand this program, the addition programs in the previous sections must be assimilated. Here ADD is replaced by SD in location 09. If the subtraction has a positive answer, then DF = 1 and BDF in location 0B is executed causing the contents of 03, where the answer resides, to be displayed. If the subtraction is negative, the program branches down to a routine, starting at location 13, which changes 1's to 0's & vice-versa, then adds one - thus turning the 2's complement form into the final (negative) result. The Q light is lit to signify that the displayed result is negative.

As for addition, multidigit numbers may be subtracted in pairs with the Borrow transferred automatically by "subtract with borrow" instructions - this is left for your experimentation and more advanced treatis es. Multiplication, Division and all the other arithmetic processes may be synthesised from Addition and Subtraction, and the reader should obtain a book incorporating Binary Arithmetic if this is of interest.

LOGIC OPERATIONS

The basic logic operations provided on the 1802 are : AND, OR, EXCLUSIVE OR and SHIFT. These are operations performed on the BITs themselves. The following defines AND:

```
1 AND 1  = 1
1 AND 0  = 0
0 AND 1  = 0
0 AND 0  = 0
```

Thus, the answer to the AND operation is "1" only if both operands are "1."

OR AND EXCLUSIVE OR (XOR) are defined by:

```
1 OR 1  = 1      1 XOR 1  = 0
1 OR 0  = 1      1 XOR 0  = 1
0 OR 1  = 1      0 XOR 1  = 1
0 OR 0  = 0      0 XOR 0  = 0
```

OR only gives the answer zero when both operands are zero - i.e. if one or other of the operands is "1" then the answer is "1". (XOR) is the same as OR except that 1 XOR 1 = 0.

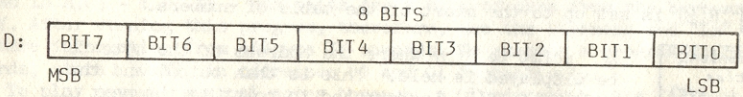
These 3 operations apply to Hex numbers by writing them in binary and applying the above rules to them bit by bit. Thus 3D OR 23 goes as follows: 3D = 00 11 11 01, 23 = 00 10 00 11

```
        00 11  11 01
OR      00 10  00 11
        -----------
=       00 11  11 11   = 3F
```

and similarly for the other operations - check 3D XOR 23 =1E and that 3D AND 23 = 21.

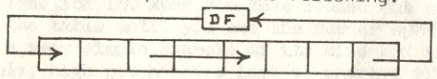
XOR with FF is a useful way to Complement a number - that is change 1's to zero's and vice versa. Check that the Complement of 23 is DC using 23 XOR FF, as an ex ercise. You should then write some small programs on the lines of the Addition and Subtraction programs above to work out the results of "OR", "AND" etc. on x and y.

The SHIFT operation allows the byte stored in D to be examined bit by bit through the DF register. D is best considered as follows for this operation:



D has 8 bits, Bit 0 to Bit 7 from right to left. Bit 7 is called the Most Significant Bit (MSB) and Bit 0 is the Least (LSB).

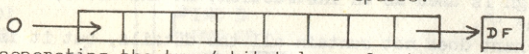
SHIFT RIGHT WITH CARRY (SHRC) performs the following:



All the bits shift to the right one place, with DF shifting into MSB and LSB shifting into DF. If Bit 3 is to be examined, and a Branch executed upon its being a "1", say, the shift is performed four times. This places the required bit in DF and "Branch if DF = 1" can be executed. This is of utmost importance for Control of external devices where the contents of D could contain the states of 8 switches and sensors. Each BIT can be treated separately by shifting it into DF and making a decision depending upon DF's state.

SHIFT LEFT With CARRY is similar but the other way around.

SHIFT LEFT (SHL) AND SHIFT RIGHT (SHR) are different only in that as the bits of D are shifted out, zeros are shifted in to fill the spaces:



This is useful in separating the two 4-bit halves of a byte and storing them in separate locations. For instance, if D = 5C, we might want location 03 to store 05 and location 02 to store 0C - and similarly for any other D—contents. See if you can write a program to do this using Shifts. The clue is that if you Shift D Right 4 times, you are left with 05. However, the C is lost - therefore you must store D first.

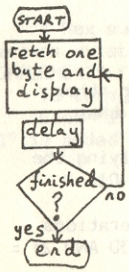
Another useful excercise is to display the bits in D on the Q light, one after the other, as the digital display shows 00, 01, 02, 03, 04, 05, 06, 07 to indicate which BIT is on the Q light at any time. A delay should be inserted between each display to allow the condition to be read.

NOTE

There are one or two instructions left for more comprehensive books on 1802 Software. See Appendix I . The TOM PITTMAN book gives a very easy to read guide to 1802 Machine Code, the 1802 Manual, however, is more complete.

General Programming Techniques

In order to write a computer program to perform some activity, a logical set of steps must be written down to achieve the end. Sometimes, it is easier to write this stage on the form of a chart. For instance, suppose we wish to show 16 bytes on the display, with a time delay between each byte. The 16 bytes must be stored somewhere in memory then fetched and displayed one after the other. A means of terminating the process must be decided upon. The following shows a "high level" chart of this process.

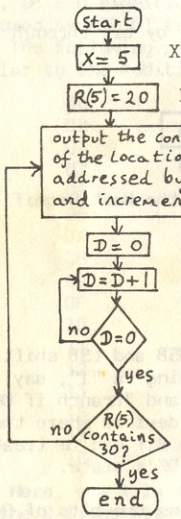


Here, some standard "flow-chart" symbols are used. ⬭ is used for start and end; ▭ is used for statements; ◇ is used for decision boxes.

Thus , after start, a byte is fetched from the number store and displayed for a time. Then, a decision is made as to whether all the bytes have been displayed yet. If so, the program ends, if not, the next byte is fetched. This process of repeating the same activity several times is called a"loop" . Here the loop is performed 16 times. If you run a program with a mistake in it, there is a chance that your program will enter an infinite loop from which a RESET is necessary for escape.

This chart orders the essential parts of the program without actually indicating how those parts are to be achieved. To help in writing the program itself, a more detailed chart can be drawn. However, before this is done, some details must be given. The sixteen numbers must be allotted specific addresses. We shall choose addresses: 20, 21, . . . , 2F. In addition, R(5) will be used to store the address of the current byte on display.

The detailed chart might look like the one below.(Comments are written by each box.)



X is used to point to R(5)

R(5) is set up to the start of the table of numbers.

X point to R(5) where the address of the byte to be displayed is held. This is then output and the address in R(5) incremented to point to the next byte to be displayed.

delay

This checks whether the contents of 2F have just been displayed - if so, R(5) will contain 30 - the address of the location following 2F. If not, a branch to the display instruction occurs.

If 30 has been reached, then the program ends.

Note that the " = " sign is used as an instruction, in the above. Thus " X=5 " means"set X to the value 5".

Even th is flowchart does not contain all the details, but it is very close. It can now be"coded" - i.e. converted into machine code. Each box can be coded easily, for instance the instruction [X=5] has op-code E5 . However, [R(5)=20] is a double instruction, as seen in a previous program. See if you can code this flowchart before reading on. Start the program from 00 as usual. The coded result follows.

```
00    E5                  SEX 5              Set X to 5
01    F8 20 A5            LDI20 & PLO 5      R(5)=20
04    64                  OUT 4              Display M(R(5)) and inc. R(5)
05    F8 00 FC 01 3A 07       This is the usual delay routine
0B    85                  GLO 5              R(5).0 → D
0C    FB 30               XRI 30             Compare D to 30
0E    3A 04               BNZ 04
10    00                  IDL                End
```


See if you can modify this program to show the program itself from 00 to 10.

In general, it is well worth while, when programming in machine code, to draw a flowchart first. It enables others to understand the program, lays the logic out in an easily modified form, and, most important, reminds you of how the program works if you have left it for a week or two.

The following programs are samples for you to try out, modify and examine carefully for tricks to use in your own programs.

(1) This program counts through the Hex number system to help you gain familiarity.

```
00    E5                  SEX 5              Location 20 (pointed to by R(5)) stores the count.
01    F8 20 A5            R(5)=20
04    F8 00               D=00               The count is started at 00
06    55                  PLO 5 STR 5        D's contents are displayed via 20 - this
07    64 25               OUT4 & DEC 5       increments R(5) - R(5) is then DECed.
09    F8 00               LDI 00
0B    FC 01 3A 0B         }                  Two delays
0F    FC 01 3A 0F         }
13    05                  LD5                The current contents of 20 are loaded
14    FC 01               ADI 01             into D and incremented.
16    30 06               BRA 06             The program then displays the new count.
```

(2) Programs and data do not have to start from 00. The following program allows you to load information higher up in memory without the necessity for pressing "In" dozens of times to reach those locations.

```
00    F8 04 AF            R(F)=04            This program uses R(F) as the
03    DF                  SEP F              Program Counter and sets R(0)
04    F8 XY A0            R(0)=XY            to address XY then waits for
07    00                  IDL                a LOAD to occur.
```

In general, when RESET is entered, followed by RUN, the Program Counter is R(0), and is set to 00. Thus the contents of 00 are fetched and executed. Here, the resulting program turns R(F) into the Program Counter (set initially to 04) causing a jump to 04. At this location, R(0) is set to some address XY, chosen by the user, and the IDLE state is entered. In this state, the computer outputs the address stored in R(0) - i.e. XY - and waits for a byte of data from the keyboard to store in XY. After running this program, therefore, use the keyboard and "In" key as usual to store information from address XY onwards. To execute a program stored from XY onwards, store 30 XY in addresses 00 and 01 respectively.

(3) To play several musical notes through a loudspeaker, the following program uses a "look up" table. The bytes stored in this block of memory are fetched, one by one , and used to determine the pitch of each successive note. The table of notes must end with 00. When this is reached, the computer goes back to the beginning and starts again. The table starts at location 16. When you have loaded the program, experiment with different numbers in the table until you can tie number up with pitch.

This program is a very simple example of its kind and suffers from several drawbacks. For example, high pitch notes last a shorter time than low. Try and correct some of the defects by writing a more sophisticated routine. The highest pitch of note attainable depends upon the frequency of the computer's clock. This may be increased by connecting a 1K resistor in parallel with R20.

```
00    F8 00 A6            R(6).0=00
03    F8 15 A5            R(5).0=15
06    15                  INC R(5)
07    05                  D=M(R(5).0)
08    32 03               BEQ 03
0A    7B                  SEQ
0B    FC 01               INC D
0D    3A 0B               BNE 0B
0F    7A                  REQ
10    16                  INC R(6)
11    86                  D=R(6).0
12    3A 07               BNE 07
14    30 06               BRA 06
16    The lookup table of notes starts here.
```

To understand the workings of this program, you should draw a "dry run" table and then a flowchart. You should then be able to experiment with some modifications of your own to effect an improvement to the design.

Final Hints

You should always check your program through before running it. Remember that as you step through, the Am key may be used to correct any byte being displayed as it comes up. You will also find it advantageous to keep a note-book of all programs and changes during their development. Even "bugs" should be written up to help in the future.

CHAPTER 5
HARDWARE, Control, Troubleshooting

INTRODUCTION

Two areas are described in this chapter. Firstly, methods of using the EDUKIT for controling external devices, and secondly a brief description of the hardware of the EDUKIT itself with special reference to troubleshooting.

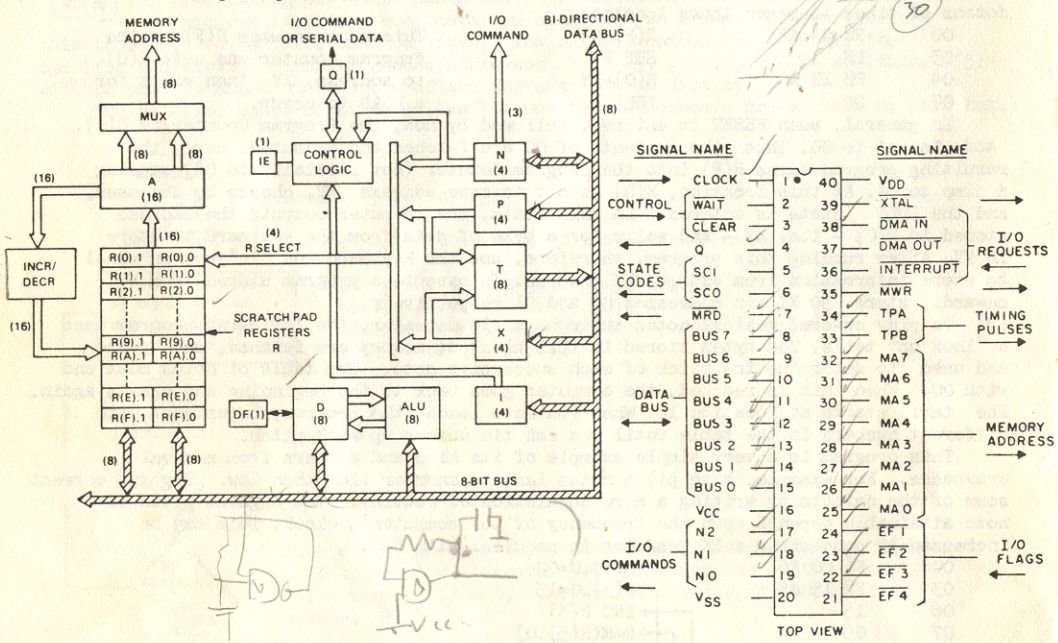
Neither of these areas is as amenable to a short discussion from scratch as is the Software of the last chapter. Thus, it is assumed, here, that the reader has a basic understanding of electronic logic devices. If this is not so, Don Lancaster's excellent "TTL Cookbook" referred to in Appendix I, will help considerably.

The 1802 User's manual (see Appendix I) is a very important supplement to this chapter and should be used for continual reference.

1802

The 1802 is an interesting and unique type of 8-bit MPU. Not only does it have many specially dedicated instructions and lines for I/O, but it has a Program Loader and execution function built in. There is also a simple but very useful Direct Memory Access (DMA) facility on the chip which allows an external device to Read from or Store at the memory location pointed to by R(0). In addition, the processor is a CMOS device and consumes very little power.

The following diagrams show the 1802's achitecture and pin-out.

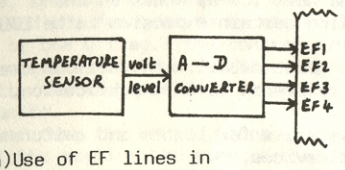


External Flag Lines

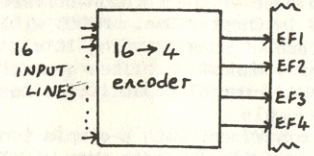
As explained in the previous chapter, there are Condtional Branch Instructions which are executed depending upon the state of the "EF" lines - pin numbers : 21, 22, 23 and 24. These pins are brought out to pads next to the processor for convenience. These "External Flags", as they are called, are normally used in conjunction with an interrupt, to identify the interrupting device. However, for a small Control System having a few switches or sensors which must be read and acted upon, the EF lines are ideal. These 4 lines allow 2^4 = 16 different switches, with some external logic, to be scanned. Alternatively, just four switches may be attached with no external logic at all.

Another interesting use of thse lines would be with a 16-level Analogue to Digital (A-D) Converter. A control activity may depend upon the temperature of a room - different actions being required for different bands of temperature. An electronic temperature gauge would be fed through an A-D Converter giving binary patterns on 4 lines connected to the EF lines of the EDUKIT. The program would need to decode the patterns of bits into some action. The EF lines must be scanned continually to ensure that a change in

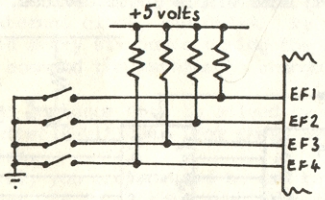
temperature is noticed quickly. However, the program for decoding the bit patterns into activities is rather cumbersome, and more efficient methods of achieving this effect will be seen below. Before proceeding, some circuit suggestions are given for use of the External Flags as described above.



(a)Use of EF lines in analogue sensing.



(b)16 lines may be sensed by the EF inputs.



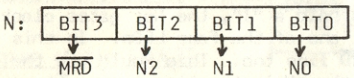
(c) Alternatively, the EF lines provide a very neat and simple method of reading the state of one of four lines.

Direct Bus Access

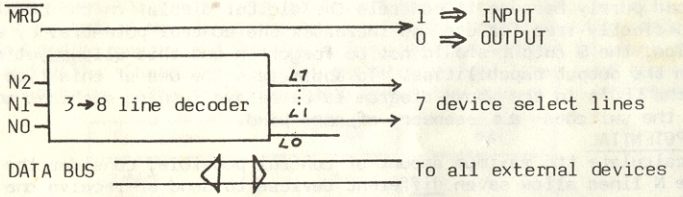
There are some special Input/Output instructions which allow up to seven devices to send and receive one byte of information via the Data Bus of the EDUKIT. They are the "Input-Output Byte Transfer" group one of which has already been introduced to display Hex digits on the EDUKIT. The N register is used to select an external device as well as to signal the direction of the Byte to be transferred.

To transfer a byte to or from an external device, the 6N instruction (see table of op-codes) is used. Thus I = 6 and N is some Hex digit. The value of N determines both data direction and device selected using the $\overline{MRD}$ pin and N0, N1 and N2 pins, all brought out to pads next to the 1802 (see component overlay).

During the instruction's execution, the bits of N appear on the above lines according to the following diagram:



It is up to some external logic to decode four control lines to activate the appropriate device.



The different binary numbers on N0 - N2 select L1 to L7. L0 is selected when N0 - N2 are all zero, as occurs when no I/O instruction is being executed.

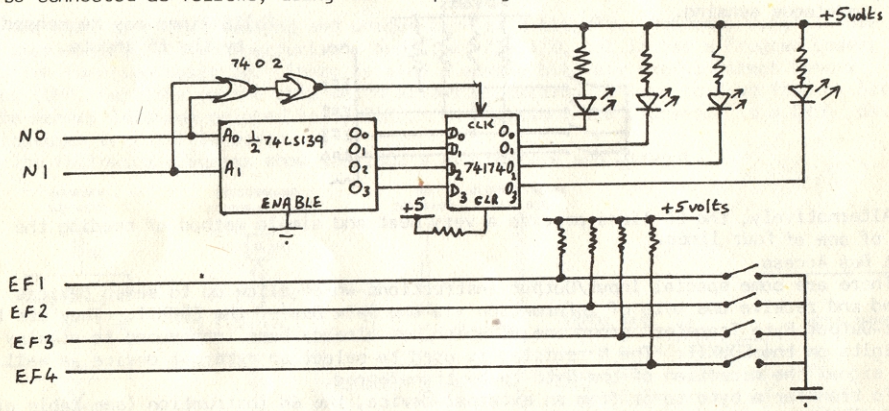
WhenN is between 9 and F, Bit 3 in N is at a 1 and this appears on $\overline{MRD}$, to signal an Input to memory from an external device. When N is between 1 and 7, Output from memory is implied. During Input, the X register is used to point the R(X) which contains the memory address into which the byte is transferred. D is also loaded with the input byte simultaneously. X and R(X) must be set correctly before the transfer is effected. During Output, R(X) is again used to hold the byte to be transferred. In this case, the contents of R(X) appear on the Bus, $\overline{MRD}$ is set to "0", the N lines set up and then R(X) is incremented. The last operation is very convenient if several bytes, stored consecutively in memory, are to be output one after the other.

All the external devices connect to the EDUKIT's Data Bus and are activated when appropr iate. It is essential, therefore, for devices to be TRI-STATE if they are to place data on the Data Bus for Input to the EDUKIT. It is important, also, to buffer the data lines from the EDUKIT for external use. The Data Bus lines may be picked up from some of the through-correction pins near IC7 as shown on the component overlay in Chapter One. MODUS will be making certain expansion tothe EDUKIT to plug in, in place of some existing IC's.

The 1802 User's Manual describes general I/O in great detail. Any serious use of the EDUKIT's full control capabilities must make reference to this publication.

A Small control example

In order to experiment with a simple Control System, a few lights and switches can be connected as follows, using some simple logic devices.



This suggested circuit does not use the Data Bus - but controls one of 4 outputs and four inputs, directly. There is a unique machine-code instruction for each light and switch.

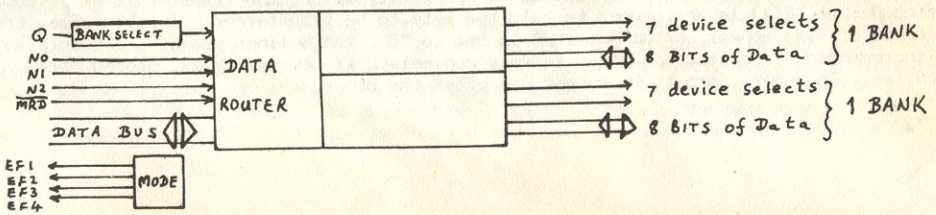
Only three integrated circuit packages are required. The N0, N1 lines are decoded to select one of four lines via a 2 to 4 decoder (74 LS 139).

When one or both N lines hold a "1", the Nor gates clock the latch (74174) to hold the selected output and light one of the four LEDs. In this way, eight outputs may be controlled by decoding the MRD line too. This would use the other half of the 74LS139.

To control the circuit, the 6N instruction is used to turn any LED on and the Short Branch Instructions used to sense and act upon the states of the switches. N2 is left unused purely because it controls the digital display on the EDUKIT. However, the line is perfectly free for use and increases the control potential by a factor of two. In addition, the Q output should not be forgotten and this allows yet another factor of two in the output capabilities. To appreciate the use of this type of control, imagine the LED's in the above diagram to be relays turning machines on and off, and think of the switches as sensors of some kind.

MAXIMUM POTENTIAL

To calculate the maximum amount of control possible, consider the following - The three N lines allow seven different devices to send or receive one byte of information - depending upon the state of MRD. Each byte has eight bits and each bit could be the state of a switch. Thus 7 x 8 = 56 switches can be monitored at once. Output is similar - 56 relays (or equivalent devices) can be controlled. In addition, the four EF lines can be used for very fast servicing of four input lines - these could be used as an overiding "mode" control, best described by the example below. Finally, everything can be simply doubled by using the Q flag to switch from one set of lines to another.



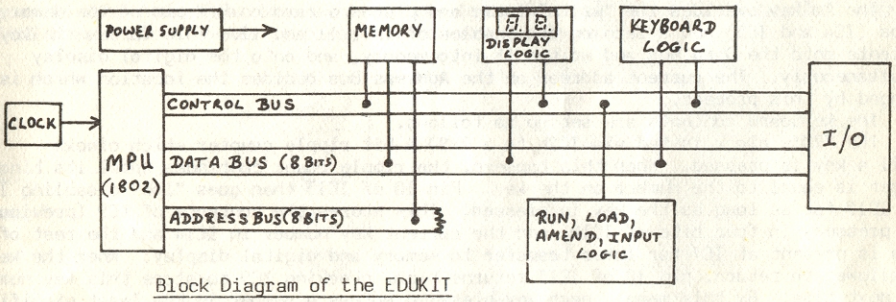
The above diagram shows the sort of potential possible. Suppose that the system is used to control the security of a building. There could be fourteen rooms, say, each with eight sensors for temperature, smoke, intruder etc, and eight relays for lights, heaters, security cameras etc. etc. Each room is called a "device" in the block diagram above, and the fourteen rooms are divided into two banks, each serviced for one of the states of the Q flag. The control program continually fetches data from the building, examines the bytes thus retrieved using, perhaps, the shift instructions to examine each bit. At the same time (apparently) it sends data out in bytes to control the slave units.

Overiding the entire process, is the state of the EF lines. These could be set to tell the machine the time of the day. For instance, each of four six hour periods could be signal ed, by an external clock, via EF1-4. By this means, the program controling the building could be changed every six hours during the day. Alternatively, light sensors could feed EF1-4 to command the machine to change from day-time operation to night.

The problem of gathering analogue data is solved by using an A-D Converter and using the above system to gather 8-bit data from these devices directly into the computer's memory for analysis and action.

If you are new to Control, you are advised to use the simpler facilities of the EDUKIT to learn. In many ways, it is in this area that the major advances in micro-computing are waiting to be made, and a few lights and switches connected to the EDUKIT will teach you more of an immediately practical nature than any number of books.

Hardware of the EDUKIT and Troubleshooting



Block Diagram of the EDUKIT

The block diagram shows the basic ingredients, and the circuit diagram fills in the details for those interested in modifying the basic unit, or troubleshooting malfunctions.

The 1802 allows Loading memory and Running a program from address 00 using two Mode Select lines $\overline{\text{CLEAR}}$ and $\overline{\text{WAIT}}$. There are four bit patterns on two lines, and the other two modes are RESET and Pause. The following table gives the states for all these functions:

Function	$\overline{\text{CLEAR}}$	$\overline{\text{WAIT}}$
LOAD	0	0
RESET	0	1
PAUSE	1	0
RUN	1	1

When Run is entered, from RESET, R(0) is used as the Program Counter and set to 00. Pause simply suspends operation. Load is used with the Direct Memory Access (DMA) function of the 1802 to load memory sequentially from 00 upwards. The appropriate state is set on $\overline{\text{CLEAR}}$ and $\overline{\text{WAIT}}$ by the flip-flops in IC3 as the (capacitively debounced) R & L buttons are operated.

To understand the DMA functions of the 1802, the state - Code outputs SC0 and SC1 must be described. They give access to the type of cycle being performed, at any time, by the Processor, according to the following table (refer to 1802 user manual).

STATE	SC1 (pin 5)	SC0 (pin 6)
FETCH	0	0
EXECUTE	0	1
DMA	1	0
INTERRUPT	1	1

When the $\overline{\text{DMA IN}}$ line on the 1802 is taken "low", the processor uses (R(0) to point to a location to which the Data Bus Contents are immediately loaded. It is up to an external device to ensure that the correct Bus Contents are present. In addition, SC1 goes high and R(0) is incremented to allow a series of memory locations to be loaded. The address in R(0) is present on the Address Bus from the 1802 to ensure that the correct locations are loaded. In the LOAD mode, the processor waits for a DMA. In order to ensure that the $\overline{\text{DMA IN}}$ line (controlled by the In key) does not stay low too long, a flip-flop (IC2a) is set by the In key when a DMA is needed, and reset again by SC1's going to "1", via IC6a. Thus each time "In" is pressed, you can watch the address on the Address Bus increment - for checking purposes. At the same time, $\overline{\text{MWR}}$ pulses low to place the memory into write mode via IC6c and IC6d. Memory Protect simply forces $\overline{\text{MWR}}$ to a permanent "1" preventing memory from being overwritten. This also puts memory into the Read state and allows the contents of memory to appear on the Data Bus and hence on the digital display through IC11 and IC12.

The "Am" key overides the "write" logic and places a zero on R/W pin of the memory chips (IC4 and IC5) and a zero on the Enables of IC7, 11 and 12. This forces the keyboard contents onto the Data Bus and writes it into memory, and onto the digital display simultaneously. The current address on the Address Bus decides the location which is changed by this process.

The keyboard contents are set up as follows.

The MPU's clock is fed via IC2b to a 7493 4-bit ripple counter which clocks until a key is pressed. When this happens, the ripple count continues until its binary number is equal to the number on the key. Pin 10 of IC13 then goes "low" disabling IC2b and IC15 for as long as the key is pressed. This stores the contents of IC9 (previous key pressed) in four bits of IC10, and the current key number in IC14 and the rest of IC10. This is present at IC7 for later transfer to memory and digital display. When the key is allowed to return, pin 10 of IC13 returns high, clocking IC9 to store this key number for next time. By this means, each key-pressing enters a number in the least significant position after shifting the previous LS digit to the most significant place. The number thus set up is only allowed "out" when Am or In keys are pressed. "In" advances to the next location before storing the current keyboard number, "Am" stores at the current address.

The MPU is clocked by an R - C oscillator using the Schmitt trigger IC6. This is the first unit to be checked if the processer does not work.

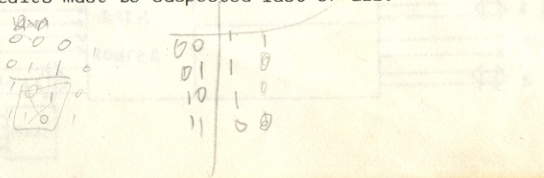
TROUBLESHOOTING

When power is applied, a digital number will appear on the display. If not, check that power is reaching IC11 and IC12 and that the DI1 and DI2 are connected to Ground Correctly.

If the R and L lights do not turn on as R and L are pressed, check that operating R and L affects pins 1 and 5 of IC3 respectively. A meter, or even an LED in series with a 270 Ohm resister will suffice for these checks. Check that IC3 is receiving power, then check that all its pins are soldered or pushed into its socket properly.

To check any further, an oscilloscope is reasonably essential. Howver, in our experience, between 96% and 98% of all units sent back for repairs have a dry joint, a pin-through unsoldered or missing, a reversed diode or a solder bridge. It is, therefore, well worth while checking again to be sure!

The incidence of "duff" chips is lower than 0.5% and is far outweighed by assembly errors. Unless you have reversed or overloaded the power supplies at some time, the integrated circuits must be suspected last of all.



If a scope is available, check that the clock is oscillating and that $\overline{\text{MWR}}$ goes low for a short time when "In" is pressed in the Load mode with M/P off. Then enter the Run mode - this may make the Address and Data Buses oscillate. If not, press the L light on and off with R on - this usually has a noticeable effect on Address and Data Bus if the processer is working. If not, check power and $\overline{\text{CLEAR}}$ and $\overline{\text{WAIT}}$.

If the keyboard circuit is not displaying keys pressed when "Am" is pressed, check that pins 1 and 19 of IC7 go low when "Am" is pressed and when "In" is pressed in the LOAD mode.

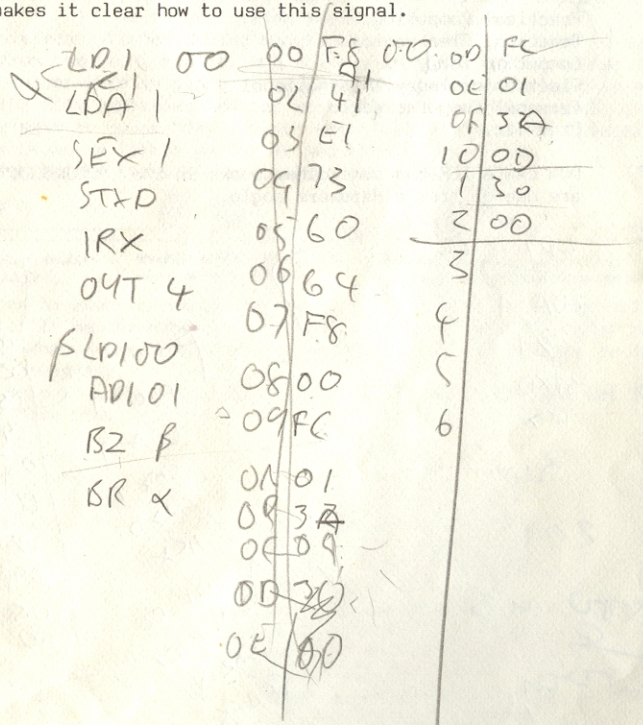
Interrupts

when the Processor's INTERRUPT pin is brought low, execution of the current program ceases, and the MPU saves the current X Register and Program Counter. The next instruction is fetched from the address stored in R(1), which must,therefore, be initialised before any interrupts are met. The processor's response to interrupts may be prevented by the use of a machine code instruction. In order to use interrupts correctly, it is essential to refer to the 1802 User's Manual.

Interrupt techniques are used primarily to allow external devices to signal that they need servicing by the computer.

Memory Expansion

The memory of the EDUKIT is by no means restricted to the small amount of on-board memory provided. It is perfectly possible to expand the computer up to 64K by bringing the TPA signal out to an external logic board. The extra sixteen Bits of Address information are multiplexed onto Address Bus, and TPA signals that the upper 16 Bits have just been sent and the lower Bits are about to be present. Again, the 1802 User's Manual makes it clear how to use this signal.



APPENDIX I

BIBLIOGRAPHY

(a) User Manual for The CDP1802 COSMAC Microprocessor, published by RCA.

(b) A Short Course In Programming, By TOM PITTMAN, published by Netronics R and D Ltd.
This book is written for another system which uses the 1802, but it describes the machine code very fully for any 1802 based machine.

(c) TTL COOKBOOK, By DON LANCASTER, published by SAMS

(d) The OSBORNE books are very good for an introduction to Microcomputing. These are available from any good bookseller advertising in the magazines suggested below.

(e) Monthly publications such as:

Practical Computing
Practical Electronics
Computing Today
Electronics Today International
Personal Computer World
Etc. etc.

(f) DON LANCASTER has some other books in the "COOKBOOK" series which are useful from a Hardware angle.

APPENDIX II

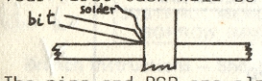
SOLDERING

In order to form a good solder joint, a hot clean iron, clean work and a good-quality flux-cored solder is required. During soldering, two pieces of metal are heated up to a temperature sufficiently high to melt a third metal (solder). This runs into the joint, cools, solidifies and joins the two pieces of metal together. The probem is that the metal surfaces to be joined must remain clean during the process or the solder does not join to the metal itself. When copper, and most metals, are heated , a layer of oxide forms which makes soldering difficult. It is this layer which the special resin-based flux in the solder is supposed to prevent.

If yoursoldering iron has never been used before, make sure you "tin" it by applying solder to the bit while it heats up for the first time. This will clean the bit and prevent its iron coating from oxidising. The iron bit should never be filed, cleaning should be done while hot, with a wet sponge or piece of cardboard followed by a coating of solder.

Practice joining two pieces of wire together. Remove the plastic insulation and tin each piece of wire by holding the soldering iron to each piece separately, and running solder into the iron/wire interface. Then twist the wires together and apply the iron along with some solder. The whole joint should run cleanly with solder. In general, applying solder while the iron is held against the work, greatly assists heat transfer as well as bringing flux onto the surfaces. Next, practice using minimum solder possible to effect a good strong joint - this is very important when assembling electronic boards to prevent solder bridging to nearby tracks.

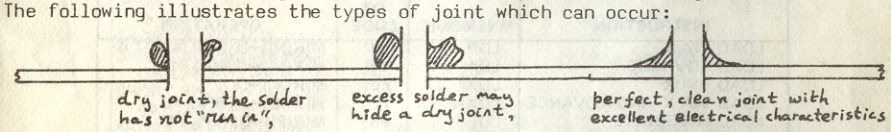
Your first task will be to solder pins through the board.



The pins should be handled as little as possible as finger grease impedes the process.

The pins and PCB are already tinned during manufacture, and all you have to do is shake any excess solder from the iron, tin it, and quickly apply it to the corner between pin and PCB with solder held in place to run into the joint. The process must be completed swiftly or the flux vaporises and the work begins to oxidise making the soldering "dry". This is said to cause "dry" joints which will normally be intermittent at best. Again, practice using as little solder as possible on the joint.

The following illustrates the types of joint which can occur:



The soldering-iron can be used to remove solder, by shaking the old solder off it and applying to the joint. It may be necessary to invert work and let the solder run downwards on to the iron. When excess solder has been removed, it is possible to inspect the joint and resolder if necessary.

Do not put too much solder on the iron, but tin it before each joint and act fast so that the flux does not have time to burn off.

Appendix II
Instruction Summary

The COSMAC instruction summary is given in Table I.

Hexadecimal notation is used to refer to the 4-bit binary codes.

In all registers bits are numbered from the least significant bit (LSB) to the most significant bit (MSB) starting with 0.

R(W): Register designated by W, where W=N or X, or P

R(W).0: Lower-order byte of R(W)
R(W).1: Higher-order byte of R(W)
N0 = Least significant Bit of N Register

Operation Notation

M(R(N)) → D; R(N) + 1

This notation means: The memory byte pointed to by R(N) is loaded into D, and R(N) is incremented by 1.

TABLE I – INSTRUCTION SUMMARY
by Class of Operation

Register Operations

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
INCREMENT REG N	INC	1N	R(N) +1
DECREMENT REG N	DEC	2N	R(N) –1
INCREMENT REG X	IRX	60	R(X) +1
GET LOW REG N	GLO	8N	R(N).0→D
PUT LOW REG N	PLO	AN	D→R(N).0
GET HIGH REG N	GHI	9N	R(N).1→D
PUT HIGH REG N	PHI	BN	D→R(N).1

Memory Reference

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
LOAD VIA N	LDN	0N	M(R(N))→D; FOR N NOT 0
LOAD ADVANCE	LDA	4N	M(R(N))→D;R(N) +1
LOAD VIA X	LDX	F0	M(R(X))→D
LOAD VIA X AND ADVANCE	LDXA	72	M(R(X))→D; R(X) +1
LOAD IMMEDIATE	LDI	F8	M(R(P))→D; R(P) + 1
STORE VIA N	STR	5N	D→M(R(N))
STORE VIA X AND DECREMENT	STXD	73	D→M(R(X)); R(X) –1

Logic Operations♦♦

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
OR	OR	F1	M(R(X)) OR D→D
OR IMMEDIATE	ORI	F9	M(R(P)) OR D→D; R(P) +1
EXCLUSIVE OR	XOR	F3	M(R(X)) XOR D→D
EXCLUSIVE OR IMMEDIATE	XRI	FB	M(R(P)) XOR D→D; R(P) +1
AND	AND	F2	M(R(X)) AND D→D
AND IMMEDIATE	ANI	FA	M(R(P)) AND D→D; R(P) +1
SHIFT RIGHT	SHR	F6	SHIFT D RIGHT, LSB(D)→DF, 0→MSB(D)
SHIFT RIGHT WITH CARRY	SHRC	76♦	SHIFT D RIGHT, LSB(D)→DF, DF→MSB(D)
RING SHIFT RIGHT	RSHR		
SHIFT LEFT	SHL	FE	SHIFT D LEFT, MSB(D)→DF, 0→LSB(D)
SHIFT LEFT WITH CARRY	SHLC	7E♦	SHIFT D LEFT, MSB(D)→DF, DF→LSB(D)
RING SHIFT LEFT	RSHL		

♦NOTE: THIS INSTRUCTION IS ASSOCIATED WITH MORE THAN ONE MNEMONIC. EACH MNEMONIC IS INDIVIDUALLY LISTED.
♦♦NOTE: THE ARITHMETIC OPERATIONS AND THE SHIFT INSTRUCTIONS ARE THE ONLY INSTRUCTIONS THAT CAN ALTER THE DF.

Arithmetic Operations♦♦

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
ADD	ADD	F4	M(R(X)) +D→DF, D
ADD IMMEDIATE	ADI	FC	M(R(P)) +D→DF, D; R(P) +1
ADD WITH CARRY	ADC	74	M(R(X)) +D +DF→DF, D
ADD WITH CARRY, IMMEDIATE	ADCI	7C	M(R(P)) +D +DF→DF, D R(P) +1
SUBTRACT D	SD	F5	M(R(X))–D→DF, D
SUBTRACT D IMMEDIATE	SDI	FD	M(R(P))–D→DF, D; R(P) +1
SUBTRACT D WITH BORROW	SDB	75	M(R(X))–D–(NOT DF)→DF, D
SUBTRACT D WITH BORROW, IMMEDIATE	SDBI	7D	M(R(P))–D–(NOT DF)→DF, D, R(P) +1
SUBTRACT MEMORY	SM	F7	D–M(R(X))→DF, D
SUBTRACT MEMORY IMMEDIATE	SMI	FF	D–M(R(P))→DF, D; R(P) +1
SUBTRACT MEMORY WITH BORROW	SMB	77	D–M(R(X))–(NOT DF)→DF, D
SUBTRACT MEMORY WITH BORROW, IMMEDIATE	SMBI	7F	D–M(R(P))–(NOT DF)→DF, D R(P) +1

Branch Instructions – Short Branch

SHORT BRANCH	BR	30	M(R(P))→R(P).0
NO SHORT BRANCH (SEE SKP)	NBR	38♦	R(P) +1
SHORT BRANCH IF D=0	BZ	32	IF D=0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF D NOT 0	BNZ	3A	IF D NOT 0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF DF=1	BDF	33♦	IF DF=1, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF POS OR ZERO	BPZ		
SHORT BRANCH IF EQUAL OR GREATER	BGE		
SHORT BRANCH IF DF=0	BNF	3B♦	IF DF=0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF MINUS	BM		
SHORT BRANCH IF LESS	BL		
SHORT BRANCH IF Q=1	BQ	31	IF Q=1, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF Q=0	BNQ	39	IF Q=0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF1=1 (1 = V_{SS})	B1	34	IF EF1=1, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF1=0 (0 = V_{CC})	BN1	3C	IF EF1=0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF2=1 (1 = V_{SS})	B2	35	IF EF2=1, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF2=0 (0 = V_{CC})	BN2	3D	IF EF2=0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF3=1 (1 = V_{SS})	B3	36	IF EF3=1, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF3=0 (0 = V_{CC})	BN3	3E	IF EF3=0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF4=1 (1 = V_{SS})	B4	37	IF EF4=1, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF4=0 (0 = V_{CC})	BN4	3F	IF EF4=0, M(R(P))→R(P).0 ELSE R(P) +1

Component List:

Integrated Circuits and Sockets

IC1	1802
IC2, IC3	7473 (2 off)
IC4, IC5	2111 (2 off)
IC6	74LS 132
IC7	74LS 244
IC8	7402
IC9, IC10	74174 (2 off)
IC11, IC12	9368 (2 off)
IC13	74150
IC14	7474
IC15	7493
40-pin socket for IC1	

Capacitors

C1,C2,C3	22 mF (typically) Electrolytic
C4,C5,C6	10 nF ceramic
C7	100 nF ceramic

Miscellaneous

20 Keyswitches
Subminiature on/off switch
Vero pins (200 off)
PCB
EDUKIT manual
Card of keyboard legends.

Resistors (¼ Watt 5%)

R1-R17	1K (17 off)
R18-R22	2K2 (5 off)
R23-R25	100 (3 off)
R26, R27	10K (2 off)
R28	18K

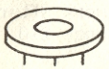
Diodes, Transistors, LED's

D1	IN4001
D2-D8	IN4148 (Typically) (7 off)
LD1-LD3	Red Led's (3 off)
TR1	2N2926 (typically)
DI1,DI2	FND 500 (2 off)

COMPONENT IDNTIFICATION

There are several different types of component - they are listed and described below. Refer, throughout, to the Component overlay.

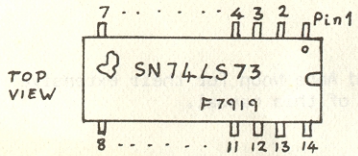
KEY SWITCHES

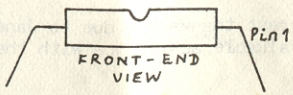


The three pins on the switches may need cleaning before soldering - they fit their holes closely and must not be tinned before insertion.

INTEGRATED CIRCUITS AND SOCKETS

The black plastic packages with pins sticking out are the integrated circuits (IC's) themselves, sometimes called "chips" (or even bugs). These packages are of the "dual in line" type (DIL) whereby each has two identical rows of pins - one on each side of the package.





Various markings are to be found on the top of the DIL packages. In order to identify a given device from its IC number in the component list, first read the device type number in the list, e.g., 9368, 7402 or 74LS244 etc. This combination of numbers and letters should appear on the chip somewhere, sometimes with various other letters and numbers. E.G, 7473 may appear (as shown) with an SN in front and an LS in the middle. Identify each device before proceeding.

Next, note that in addition to various dents in the upper surface, one end ha notch cut into its upper surface as shown above. This notch is used to ensure that package is inserted in position the correct way round. Failure to observe the correc orientation will totally destroy the IC's. The notch must tie up with the component overlay diagram.

Sometimes, instead of, or in addition to, the notch, a small hole or bump will appear on the upper surface of the IC to identify Pin 1. The pins are numbered, on all chips, anticlockwise around the chip, as shown, when looking at the top surface. The largest IC, the Microprocessor itself, is a 40-pin device, and a socket is included, for it, in the kit. Do not remove the IC from its packing until absolutely necessary.

CAPACITORS

There are just two types of capacitor :

a) Electrolytic : C 1, C 2, C 3 - all identical.

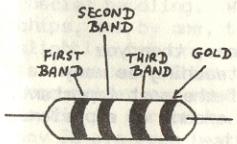
These devices have two pins - one of them is "+" and one "-", as shown on the body of the device. The + side is marked on the PCB so that these capacitors can be inserted the correct way round.

b) Ceramic : C 4, C 5, C 6, C 7.

These are flat devices with two pins and can be inserted any way round - C4, C5, C6 are marked "10n" and C7 is marked "100n" - perhaps with some other letters, depending upon the type supplied.

RESISTORS

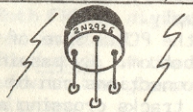
All resistors have red bodies and are similar, apart from coloured bands which identify the value of the resistor according to the following table:



Resistor	Value	Bands		
		First	Second	Third
R1-R17	1K	BROWN	BLACK	RED
R18-R22	2K2	RED	RED	RED
R23-R25	100	BROWN	BLACK	BROWN
R26, R27	10K	BROWN	BLACK	ORANGE
R28	18K	BROWN	GREY	ORANGE

Resistors may be inserted any way round.

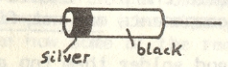
DIODES, TRANSISTORS, LED's.



The transistor (TR1) has 3 leads and 2N2926 written on it. It must be inserted, as shown here, with correct pins in correct holes.

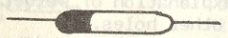
There are three types of diode. They must all be inserted the correct way around.

a) IN4001 (D1) has a black body and a silver ring around one end.

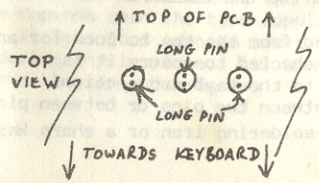


The ring end must tie up with the black stripe on the Component Overlay.

b) Small signal diodes: D2 - D8. Again, one end is striped - with Black or yellow - and must tie up with the striped end shown on the Component Overlay.



c) Light-Emitting Diodes (LED's) LD1, LD2, LD3.



These are translucent red objects with two pins - one longer than the other. The long pins must be inserted in the holes as shown in the diagram, with PCB oriented accordingly.